Exercise V, Theory of Computation 2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 Prove that the problem of checking whether a given DFA accepts a finite language is decidable.

Solution:

We claim that a DFA D accepts an infinite number of strings if and only if in the transition diagram of D there is a path P with the following properties:

- The path P starts at the start of D.
- The path P ends at an accepting state of D.
- The path P visits some state of D at least twice.

Note that one possible way for P to satisfy the third property is to contain a vertex that self-loops. The proof of this claim is quite simple. If there is such a P, then, just as in the proof of the pumping lemme, we can repeat the part of P between two visits to the same state arbitrarily often, giving rise to infinitely many accepting paths. Thus, $L(D)$ is infinite.

Conversely, if the language $L(D)$ is infinite, then it must contain a word w whose length is larger than the number of states in D. Thus, the accepting path for P must satisfy all three conditions above by the pigeonhole principle.

To solve the exercise, it therefore suffices to describe a Turing machine which decides if a given transition diagram contains a path with these three properties or not. As a technical aside, note that we can always reconstruct the transition diagram of a DFA from its encoding. Consider the following algorithm:

On input $\langle D \rangle$, do:

1. For all states q of D:
   (a) Check if there is a path from the start state of D to q.
   (b) Check if there is a (non-empty) path that starts at q and returns to q.
   (c) Check if there is a path from q to some accepting state of D.
   (d) If all three of these checks succeeded, then accept.
2. Reject

Note that each of the three sub-routines is clearly decidable, using simple path finding algorithms for directed graphs. By the Church-Turing thesis, the algorithms described above can be run on a Turing machine. We conclude that checking whether $L(D)$ is infinite is Turing-decidable. And since Turing-decidable languages are closed under complementation, checking whether $L(D)$ is finite is too.

As another technical aside, note that the two languages mentioned above are probably not really complimentary over the underlying alphabet which we use to encode the input DFA. This is because some input strings might not correspond to the encoding of a valid DFA at all. But we always assume that the Turing machine can decide if an input is invalid, and thus we can always assume that all inputs are valid encodings.

We remark that this exercise can also be solved by using the pumping lemma directly to show the following fact: The language $L(D)$ is infinite if and only if it contains a word w such that

$$(\text{number of states of } D) < |w| \leq 2 \times (\text{number of states of } D).$$

But this property can be easily verified by simulating the D on all the finitely many words whose length falls in that interval.

2 Prove that the language

$$L = \big\{\langle D\rangle : D \text{ is a DFA over alphabet } \{0,1\} \text{ that only rejects a single string}\big\}$$

is decidable.

Solution: We construct a Turing machine which decides L. Consider the following algorithm:

On Input $\langle D\rangle$, do:

1. If there is a rejecting path though D, extract $w \notin L(D)$, otherwise reject.
2. Construct a DFA D_w that accepts only the string w.
3. Construct a DFA D' that recognizes the language $L(D) \cup L(D_w)$.
4. If D' has a rejecting path, reject. Otherwise, accept.

We first remark that the two DFAs we construct in the procedure have simple descriptions in terms of w or the description of previous DFAs. Moreover, finding a rejecting path though a DFA can be achieved using a simple path finding algorithm. Thus, by the Church-Turing thesis the above algorithm can be implemented na Turing machine. We now prove that this algorithm decides L.

- If $\langle D\rangle \in L$, then step 1 succeeds and finds the unique $w \notin L(D)$. But now $L(D') = L(D) \cup L(D_w) = \{0,1\}^*$ does not have a rejecting path and the algorithm accepts.
- If $\langle D\rangle \notin L$, then either $L(D) = \{0,1\}^*$ and we reject at step 1, or we find some $w \notin L(D)$. Since $\langle D\rangle \notin L$, there must be some other $w \neq w' \notin L(D)$. But this implies $w' \notin D_w$ and thus $w' \notin L(D') = L(D) \cup L(D_w)$. Therefore, D' has a rejecting path and the algorithm rejects.

We remark that we could have also defined an equivalent algorithm that just operates on D, without constructing other DFAs.

3 Prove that the following questions about pairs of Turing machines (M, N) are undecidable.

3a Is $L(M) \cap L(N)$ empty?

3b Is $L(M) \cap L(N)$ finite?

Solution: Recall that the language

$$A_{\text{TM}} = \big\{\langle T, w\rangle : T \text{ is a Turing machine that accepts } w\big\}$$

is undecidable.

3a Let $L_\varnothing = \{\langle M, N\rangle : L(M) \cap L(N) = \varnothing\}$ be the language in question. We prove that if $L_\varnothing$ is decidable, then A_{TM} is decidable too, which gives us the desired contradiction.

To this end, assume that $L_\varnothing$ is decided by the Turing machine $M_\varnothing$. We construct an algorithm that decides A_{TM}:

> On input $\langle T, w\rangle$, do:
>
> 1. Construct a Turing machine M' that simulates the T on input w and outputs $T(w)$.
> 2. Run $M_\varnothing(\langle M', M'\rangle)$.
> 3. If the computation in step 2 accepted, then reject. Otherwise, accept.

We first note that this algorithm always halts. The machine $M_\varnothing$ was defined to be a decider of the language $L_\varnothing$ and thus always halts. Note that the machine M' we construct does not take an input. Also, there is no guarantee that M' halts, since T can be an arbitrary Turing machine. But this is fine, since we never actually run T or M', we just feed the description of M' to $M_\varnothing$.

By the Church-Turing thesis, there is a Turing machine M that implements the above algorithm. We now argue that M decides A_{TM}.

- If $\langle T, w\rangle \in A_{\text{TM}}$, then $T(w)$ accepts by definition. Thus, M' always accepts and $L(M') \cap L(M') \neq \emptyset$. This implies $M_\varnothing(\langle M', M'\rangle)$ rejects, and hence M accepts
- Consider $\langle T, w\rangle \notin A_{\text{TM}}$. Then $T(w)$ does not accepts (it might not halt) and thus M' never accepts. Therefore, $L(M') \cap L(M') = \emptyset$ and $M_\varnothing(\langle M', M'\rangle)$ accepts, making M reject.

We conclude that if $L_\varnothing$ is decidable, then so is A_{TM}. But we know this is not the case, hence $L_\varnothing$ is not decidable.

3b We can use the exact same solution as in part a, with $L_\varnothing$ replaced with $L_{<\infty}$. This is because the only time we used the definition of $L_\varnothing$ was to distinguish the language of all strings from the empty language. But $L_{<\infty}$ can do this job equally well.

4* We say that a Turing machine has property $\mathcal{U}$ if for all $n \in \mathbb{N}$, at most one string of length n is accepted by M. Assuming that $|\Sigma| > 1$, prove that the language

$$L = \{\langle M\rangle : M \text{ has property } \mathcal{U}\}$$

is undecidable. What happens if $|\Sigma| = 1$?

Solution: Assume for the sake of contradiction that L is decided by a Turing machine M_L. We use M_L to construct a decider M for A_{TM}, which gives us the desired contradiction:

> In input $\langle T, w\rangle$, do:
>
> 1. Construct a Turing machine M' that simulates the T on w and outputs $T(w)$.
> 2. Run $M_L(\langle M'\rangle)$.
> 3. If the computation in step 2 accepted, then reject. Otherwise, accept.

This algorithm always halts, since M_L does. We now show that this algorithm decides A_{TM}.

- If $\langle T, w\rangle \in A_{\text{TM}}$, then $T(w)$ accepts, which implies that M' accepts all strings. Hence, $M_L(\langle M'\rangle)$ rejects since M' accepts $|\Sigma| > 1$ strings of length 1 in particular. Thus, M accepts as desired.

- If $\langle T, w\rangle \notin A_{\text{TM}}$, then $T(w)$ never accepts and thus M' has property $\mathcal{U}$. Hence, $M_L(\langle M'\rangle)$ accepts and M rejects as desired.

Therefore, M decides A_{TM}, contradicting its undecidability. If $|\Sigma| = 1$, then every Turing machine has property $\mathcal{U}$ and thus L is decidable.

Note that while exercises 3 and 4 seem to be asking for very different things, the solutions are almost identical. This is no coincidence, and later in the course we might see a result which vastly generalises this argument.

5* Is the following language recognizable?

$$L = \big\{\langle M\rangle : M \text{ is a Turing machine that accepts at most 2025 strings}\big\}$$

Solution: No, L is not recognisable.

Assume for the sake of contradiction that L is recognised by the Turing machine M_L. We use this to construct an algorithm that recognizes $\overline{A_{\text{TM}}}$, which we know to be unrecognizable.

On input $\langle T, w\rangle$, do:

1. Construct a Turing machine M' that simulates T on w and outputs $T(w)$.
2. Output $M_L(\langle M'\rangle)$.

Now we show that this algorithm recognizes $\overline{A_{\text{TM}}}$.

- If $\langle T, w\rangle \in \overline{A_{\text{TM}}}$, then $T(w)$ does not accept by definition. Hence, M' accepts 0 strings, which is less than 2025. Thus, $M_L(\langle M'\rangle)$ accepts an so does our algorithm.
- If $\langle T, w\rangle \notin \overline{A_{\text{TM}}}$, then $T(w)$ accepts. Hence, M' accepts all infinitely many strings. Since that is more than 2025, $M_L(\langle M'\rangle)$ will not accept (either reject or not halt at all). Thus, that our algorithm also does not accept.

We conclude that the algorithm recognizes $\overline{A_{\text{TM}}}$, which contradicts with the fact that $\overline{A_{\text{TM}}}$ is unrecognizable. Thus, our assumption that L is recognisable must have been wrong.