Exercise VII, 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.

Problems on Mapping Reductions

1 Show that $\leq_m$ is a transitive relation on languages, i.e., if $A \leq_m B$ and $B \leq_m C$ then $A \leq_m C$.

Solution: Suppose $A \leq_m B$ and $B \leq_m C$. That is, there are computable functions f, g with

$$x \in A \iff f(x) \in B \quad \text{and} \quad y \in B \iff g(y) \in C.$$

Now let $h(x) = g(f(x))$ and note that $x \in A \iff h(x) \in C$.

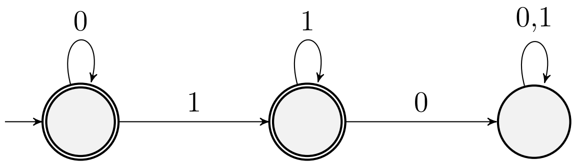
It remains to prove that h is computable by building a Turing machine that computes h. By assumption there are Turing machines F and G that compute f and g, respectively. Now consider the Turing machine that, on input x, does the following: first, simulates F on x and store the output y. hen simulate a G on y and return the result. This machine thus computes $h(x) = g(f(x))$. Hence, h is computable, so $A \leq_m C$.

2 Show that if A is a Turing-recognizable language and $A \leq_m \overline{A}$, then A is decidable.

Solution: Since $A \leq_m \overline{A}$, we also have $\overline{A} \leq_m A$ (see exercise 1 from last week). Since A is recognizable and $\overline{A} \leq_m A$, we have that $\overline{A}$ is recognizable as well. Since A and $\overline{A}$ are both recognizable, A is decidable.

3* Show that a language $A \subseteq \{0,1\}^*$ is decidable if and only if $A \leq_m \{0^a1^b \mid a, b \geq 0\}$.

Solution: We prove the two directions separately. Suppose that $A \leq_m \{0^a1^b \mid a, b \geq 0\}$. We know that $\{0^a1^b \mid a, b \geq 0\}$ is a decidable language, since it is accepted by the following DFA:



Thus, A can be reduced to a decidable language, and is therefore also decidable.

Now suppose that A is decidable and let M be the Turing machine that decides A. Consider the following algorithm.

On input w, do:

1. Run M on input w.

2. If the computation accepts, output 01. Otherwise, output 10.

This algorithm clearly halts on every input and thus defines a computable function f. Moreover, we show that $w \in A \iff f(w) \in 0^*1^*$.

- If $w \in A$, then $M(w)$ accepts. Hence, $f(w) = 01 \in 0^*1^*$.
- If $w \notin A$, then $M(w)$ rejects. Hence, $f(w) = 10 \notin 0^*1^*$.

Therefore, $A \leq_m 0^*1^*$ as desired.

4 A *useless state* in a Turing machine is a state that is never entered on any input string. Consider the problem of determining whether a Turing machine has any useless states. Formulate this problem as a language and show that it is undecidable.

Solution:
We define the language

$$\textsc{Useless} = \big\{\langle M\rangle : M \text{ has a useless state}\big\}.$$

We show that $\textsc{Useless}$ is undecidable by showing that $\overline{A_{TM}} \leq_m \textsc{Useless}$. We construct the corresponding computable function f via the following algorithm.

```
On input $\langle M, w\rangle$, do:
  1. Construct a Turing machine $T$ which on input $x$, does:
     (a) If $x = \varepsilon$, then $T$ **rejects**.
     (b) Else, $T$ simulates $M(w)$.
     (c) If $M(w)$ accepts, then $T$ also **accepts**.
     (d) If $M(w)$ rejects, then $T$ also **rejects**.
  2. Output $T$.
```

Note that this algorithm always halts. We can assume that T has no useless states, possibly with the exception of the accept and reject states. Without loss of generality, assume that the input alphabet of T is $\{0, 1\}$, has start state q_1, and the ability to not move the head after a transition. We now alter T by adding two new tape symbols s_0 and s_1, as well as a new start state q_0. Let $q_0, q_1, q_2, \ldots, q_k$ be an enumeration of the non-terminal states of T. On some (non-empty) input, T now does the following: It replaces the first symbol a by s_a on the tape and goes to state q_2 without moving the head. Then, reading s_a from the tape, it loops though the states $q_2, q_3, \ldots, q_k$, without changing the tape symbol or moving the head. Finally, from state q_k, it changes the first symbol of the tape back from s_a to a and goes to state q_1, without moving the head. After that, the machine operates exactly like the original T would. It is not hard to see that T has no non-terminal useless states and agrees with the original machine on all inputs.
It remains to show that $\langle M, w\rangle \in \overline{A_{TM}} \iff T \in \textsc{Useless}$.

- Suppose $\langle M, w\rangle \in A_{TM}$. Then $M(w)$ accepts and thus T accepts all inputs besides the empty string. Hence, T has no useless states and $T \notin \textsc{Useless}$
- Suppose $\langle M, w\rangle \notin A_{TM}$. Then T rejects or loops on all inputs. Hence, the accept state of T is a useless state and $T \in \textsc{Useless}$

This concludes the proof of the reduction. Since $\overline{A_{TM}}$ is undecidable, so is $\textsc{Useless}$. Note that this argument even shows that $\textsc{Useless}$ is unrecognisable.

Problems on Time Complexity

5 Arrange the following functions in increasing order according to asymptotic growth.

$$2^n, \quad \binom{n}{5}, \quad (\log\log n)^{10}, \quad n^{300}, \quad \sqrt{n}, \quad (\log n)^n, \quad n/\log n, \quad 2^{2^n}, \quad n^{\sqrt{n}}, \quad n!, \quad \log n$$

Solution:

$$(\log\log n)^{10} < \log n < \sqrt{n} < n/\log n < \binom{n}{5} < n^{300} < n^{\sqrt{n}} < 2^n < (\log n)^n < n! < 2^{2^n} \ .$$

6* We call an undirected graph $G = (V, E)$ *Eulerian* if there is a closed walk (i.e. returning to where it starts) on G that uses each edge exactly once. Show that the language

$$\textsc{Eulerian-Graph} = \{\langle G\rangle : G \text{ is Eulerian}\}$$

is in $\mathbf{P}$ by giving an algorithm and bounding its running time as a polynomial in $|V|$ and $|E|$.

Hint: *First prove that a connected graph is Eulerian iff every vertex has even degree.*

Solution: We first prove the correctness of the claim in the hint: A connected graph is Eulerian iff every vertex has an even degree. We show the two directions separately.

- Suppose G is Eulerian and consider a closed walk on G that uses every edge exactly once. For any vertex v, the walk must enter and exit v an equal number of times. Therefore, the degree of v is even.
- Assuming the other direction is false, let G be the connected graph with the fewest edges that is not Eulerian, but in which every vertex has even degree. Since G is connected and has at least two vertices, every vertex of G has degree at least 2, which means G is not a tree. Hence, there must be some cycle C in G. Let $H = G \setminus C$ and let $H_1, H_2, \ldots, H_k$ be the connected components of H. Note that for all i, H_i is connected and every vertex in H_i has even degree. Moreover, each H_i has strictly fewer edges than G, so must be Eulerian by assumption. Let W_i be a closed walk on H_i that uses each of its edges exactly once. Since G was connected, each W_i must share a vertex with C. Now consider the closed walk on G that goes around C, while taking each detour W_i exactly once. This walk visits every edge of G exactly, which means that G is Eulerian, a contradiction.

Now we give an algorithm that decides Eulerian-Graph using the above characterization.

```
On input $\langle G\rangle$, do:
  1. Check if $G$ is connected (e.g. by breadth-first-search from an arbitrary vertex).
  2. If $G$ is disconnected, reject.
  3. Else, iterate over all vertices $v$ of $G$ and do:
     (a) If $v$ has odd degree, reject.
  4. Else, if all degrees are even, accept.
```

Note that this algorithm takes at most $O(|V| + |E|)$ many fundamental steps. Hence, Eulerian-Graph is in $\mathbf{P}$ as desired.

7 Which of the following operations is the complexity class **P** closed under?

7a Union

7b Intersection

7c Complementation

7d* Kleene closure

Solution:

7a (Union) Let L_1 and L_2 be two languages in **P**, decidable by deterministic polynomial-time machines M_1 and M_2, respectively. We construct a deterministic polynomial-time machine M for $L_1 \cup L_2$ as follows.

```
On input $x$, do:
  1. Run $M_1(x)$. If $M_1(x)$ accepts, accept.
  2. Run $M_2(x)$. If $M_2(x)$ accepts, accept.
  3. Reject.
```

The machine M accepts x if either of the two machines accepts x, and rejects otherwise. Hence, M decides $L_1 \cup L_2$. Since both M_1 and M_2 run in polynomial time, M runs in polynomial time too. We can conclude **P** is closed under the union operation.

7b (Intersection) Let L_1 and L_2 be two languages in **P**, decidable by deterministic polynomial-time machines M_1 and M_2, respectively. We construct a deterministic polynomial-time machine M for $L_1 \cap L_2$ as follows.

```
On input $x$ do:
  1. Run $M_1(x)$. If $M_1(x)$ rejects, reject.
  2. Run $M_2(x)$. If $M_2(x)$ rejects, reject.
  3. Accept.
```

The machine M accepts x if both machines accept x, and rejects otherwise. Hence, M decides $L_1 \cap L_2$. Since both M_1 and M_2 run in polynomial time, M runs in polynomial time too. We can conclude **P** is closed under the intersection operation.

7c (Complementation) Let L be a language in **P**, decidable by a deterministic polynomial-time machine M. We construct a deterministic polynomial-time machine M' for $\overline{L}$ as follows.

```
On input $x$, do:
  1. Run $M(x)$.
  2. If $M(x)$ rejects, accept.
  3. If $M(x)$ accepts, reject.
```

The machine M' accepts x if and only if M rejects x. Hence, M' decides $\overline{L}$. Since M runs in polynomial time, M' runs in polynomial time too. We can conclude **P** is closed under the complementation operation.

7d (Kleene closure) Let L be a language in **P**, decidable by a deterministic polynomial-time machine M. We construct a deterministic polynomial-time machine M' for L^* as follows.

```
On input $w$, do:
  1. If $w = \epsilon$, accept.
  2. Let $w = w_1 w_2 \dots w_n$.
  3. Initialize an array $A$ of length $n$ with all 0's.
  4. For each $j = 1, 2, \dots, n$:
     (a) Simulate $M$ on input $w_1 \dots w_j$.
     (b) If $M$ accepted, set $A[j] = 1$.
     (c) For each $i = 2, 3, \dots, j$:
         i. Simulate $M$ on input $w_i \dots w_j$.
         ii. If $M$ accepted and $A[i-1] = 1$, set $A[j] = 1$.
  5. If $A[n] = 1$, accept. Otherwise, reject.
```

In the above construction, we use a technique called dynamic programming. The idea is to store solutions of smaller sub-problems to solve the entire problem. A word w is in L^* if and only if either w is in L, or if it can be written as $w = w'w_0$ with $w' \in L^*$ and $w_0 \in L$. We used an array A to store the results of the sub-problems asking if the prefixes of w are in L^*. Since there are only finitely many indices where we could split the word w, we can loop over all these possibilities. Note that we need to iterate over all possible split locations from smallest to largest, to ensure that we have already solved all of the required sub-problems we might need later on.

Now we argue that M' runs in polynomial-time. M' has two nested loops. Assuming that M runs in time $O(n^d)$ for some $d \in \mathbb{N}$, the complexity of M' is thus $O(n) \cdot (O(n^d) + O(n))$, which is polynomial. Hence, **P** is closed under the Kleene star operation.