EPFL

Exercise IV, 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 Are Turing-decidable languages closed under the following operations? Give informal proofs.

1a Intersection

1b Concatenation

1c Complementation

Do these arguments work for Turing-recognisable languages? If not, can you adapt them?

Solution: We argue that Turing-decidable languages are closed under all three operations.

1a Let M_1 and M_2 be Turing machines that **decide** the languages L_1 and L_2 respectively. We construct the Turing machine M as follows.

On input x, do:
1. Run $M_1(x)$.
2. Run $M_2(x)$.
3. Accept if both M_1 and M_2 accepted.
4. Reject.

Since M_1 and M_2 **decide** the languages, they halt on any input. Thus, M also halts on any input. Moreover, M clearly accepts an input x if and only if both M_1 and M_2 accept x. Thus,

$$L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2.$$

The same algorithm works for Turing-recognisable languages too. The only difference is that M might not halt on input $x \notin L_1 \cap L_2$. But this is okay, since in this case we only need M to **recognises** $L_1 \cap L_2$ by accepting all $x \in L_1 \cap L_2$.

1b Again, let M_1 and M_2 be Turing machines that decide the languages L_1 and L_2 respectively. We now construct the Turing machine M as follows.

On input x, do:
1. For all possible decompositions $x = x_1x_2$:
   (a) Run $M_1(x_1)$.
   (b) Run $M_2(x_2)$.
   (c) Accept if both $M_1(x_1)$ and $M_2(x_2)$ accepted.
2. Reject.

This algorithms always halts, since for any input x there are only finitely many decompositions $x = x_1x_2$ to check, and both M_1 and M_2 always halt since the **decide** L_1 and L_2. Since $x \in L_1L_1$ if and only if there is a composition $x = x_1x_2$ with $x_1 \in L_1$ and $x_2 \in L_2$, it is clear that M decides L_1L_2.

For recognisable languages this algorithm does not work directly. This is because M_1 or M_2 could get stuck in an infinite computation while checking a decomposition $x = x_1x_2$ that does not work. Thus we might never reach the decomposition which witnesses $x \in L_1L_2$. We can however adapt the algorithm slightly to avoid this issue. Given Turing machines M_1 and M_2 that **recognise** L_1 and L_2, define M as follows.

On input x, do:
1. For all $l \in \mathbb{N}$:
   For all possible decompositions $x = x_1x_2$:
   (a) Run $M_1(x_1)$ for at most l steps.
   (b) Run $M_2(x_2)$ for at most l steps.
   (c) Accept if both $M_1(x_1)$ and $M_2(x_2)$ accepted.
2. Reject.

We argue that M **recognises** $L = L_1L_2$.

- If $x \in L$, then there exists a decomposition $x = x_1x_2$ with $x_1 \in L_1$, and $x_2 \in L_2$. Therefore, there is some finite k such that both $M_1(x_1)$ and $M_2(x_2)$ accept within k steps. Note that each iteration of the outer loop only takes finitely many steps, and therefore M will eventually reach a value of l for which both $M_1(x_1)$ and $M_2(x_2)$ accept. Thus, $M(x)$ accepts.

- If $x \notin L$: Then for all possible decompositions $x = x_1x_2$, at least one of $M_1(x_1)$ and $M_2(x_2)$ does not accept after l steps. Hence, $M(x)$ does not accept.

1c If L is a language **decided** by the Turing machine M, then switching the accept and reject states in M will yield a Turing machine which decides the complement of L.

If M just **recognises** L, the story is much more complicated. Intuitively, to construct a machine which accept all strings not accepted by machine M, one would need to deal with the inputs on which M fails to halt. These inputs should all be accepted by the new machine for recognizing the complement.

Can we algorithmically check whether a Turing machine terminates when given a certain input? As we prove in the next lecture, this is not possible! The result of the next lecture will allow us to formalize these intuitions and provide a precise proof that recognizability is not closed under complementation.

2 Construct a Turing machine which recognizes the language

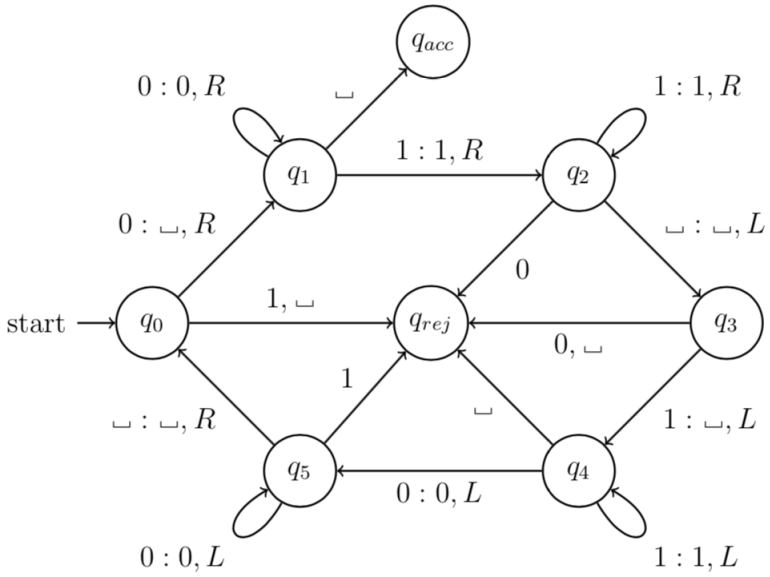
$$L = \{0^i 1^j \mid i > j \geq 0\}.$$

Solution: This problem asks to construct a Turing machine. It is useful to first think of a good strategy to recognize the language L which is easily convertible into a Turing machine procedure.

One approach is to repeatedly match a symbol 0 from the very start of the string with a symbol 1, from the very end. If we remove the matched symbols, then we recover a smaller instance of the problem. If the matching fails, then we just need to check that the remaining string is of the form 0^k for some $k \geq 1$.

On a high level, the operation can be described as follows. The machine starts from q_0, removes the first 0 it sees and moves right past all the 0s. Then it moves past all the 1s. Once it reaches the (signified by $\sqcup$), it turns back and crosses the last 1. Then it starts travelling back to the beginning of the string, passing all the 1s and then all the 0s. Once it gets to the beginning of what remains (signified by $\sqcup$), it turns around and returns to state q_0. The machine now continually repeats this procedure of removing the first 0 and last 1.

In case any of our assumptions is broken (for instance if there is a 0 after a 1), then the machine rejects. If at some point there are only 0s left (and at least one of them), the machine accepts. To get a better intuition of procedures like this and to convince yourself that this machine actually decides L, it is always advisable to run them on some sample inputs such as $\varepsilon, 0, 0011, 00000111, 0101$. Illustrated below is the state diagram of a Turing machine implementing this strategy.



Note that there are many other strategies and concrete implementation choices that also work. In fact, later in the course we will see a result that implies it is uncomputable to check if a solution to this exercise is correct!

3 Let L be the language over the singleton alphabet $\{1\}$ consisting of all the strings whose lengths are prime numbers. Thus, $L = \{11, 111, 11111, 1111111, \ldots\}$. Is L Turing-decidable?

Solution: Yes, L is indeed Turing-decidable. This claim should not be surprising, as Turing machines capture all algorithmic concepts, and we know quite a few algorithms to decide whether a given number is prime or not. We now construct a Turing machine M which decides L.

On input x, do:
1. For $2 \leq i \leq |x| - 1$:
   (a) Make a copy x' of x, somewhere else in memory.
   (b) Repeatedly erase i symbols from x' until $|x'| < i$.
   (c) If $|x'| = 0$, reject.
2. Accept.

Clearly this procedure is finite and thus M always halts. We now show that M decides L.

- If $|x|$ is prime, then there is no $2 \leq i \leq |x| - 1$ which divides $|x|$. Therefore, every iteration of the loop will end up with $|x'| > 0$ and thus $M(x)$ accepts.
- If $|x|$ is not prime, then there is at least one value $2 \leq i \leq |x| - 1$ which divides $|x|$. The corresponding iteration of the loop will thus make $M(x)$ rejects.

4 Determine all the languages over the unary alphabet $\{1\}$ which are decided by a Turing machine that only has three states (including the accept and reject states).

Solution: We prove that the complete list of such languages is

$$\varnothing, \quad \{\varepsilon\}, \quad \{1^n \mid n \geq 1\}, \quad \{1\}^*.$$

It is not hard to see that all four of these languages are indeed decided by a Turing machine of the desired form. Conversely, let M be a Turing machine with states $Q = \{q_{acc}, q_{rej}, q\}$ that decides the language L. If the start state of M is q_{acc}, then $L = \{1\}^*$. If instead the start state of M is q_{rej}, then we have $L = \varnothing$. Thus, we can assume that q is the start state of M.

Since we want M to **decide** the language L, the machine M must halt on every input. Note that if $\delta(q, \sqcup)$ maps to state q, then M will not halt on input ε. Thus, M halts as soon as the head encounters an empty cell on the tape.

If $\delta(q, 1) = (q)$ maps to state q, then no matter what symbol is written in the place of 1, and no matter which direction the head moves in afterwards, the head will just move off the beginning or end of the input string. There it will encounter a $\sqcup$ and halt with the same outcome for all inputs. If instead $\delta(q, 1)$ maps to a halting state, then membership in L just depends on whether the input is empty. We conclude that the list given above is exhaustive.

5* Consider a variant of the Turing machine whose head can move right or stay put. In particular, the transition function δ of such a machine takes the form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{P,\ R\},$$

where P corresponds to the head staying in place after changing the state and tape symbol.

Argue that this model of computation is not equivalent to our standard Turing machine model, and describe the class of languages it recognises.

Hint: *First show this model is equivalent to the model where the head can only move right.*

Solution: Intuitively, not being able to move to the left means that the machine can only do one pass over the input string. This reminds us of the way DFAs run computations. In fact, we will show that these machines recognize exactly the regular languages.

It is easy to see that we can simulate any DFA on such a machine by moving to the right at every step, reading the symbols one at a time, and halting accordingly as soon as we read a blank symbol indicating the end of the input. More concretely, our Turing machine variant will have the same set of states as the DFA, with an added "accept" and "reject" state. While reading symbols of the input alphabet, we emulate the exact behaviour of the DFA. As soon as we read the first blank symbol, we transition to either the "accept" or "reject" state, depending on whether our current state was accepting or rejecting in the DFA. The start state will be the same as in the DFA. Note that it does not matter what we write on the tape, as we never read from the same tape position twice.

It remains to show that we can construct an equivalent DFA for such a Turing machine variant M. Note that if there are any transitions of M where the head stays put, then in the next step, the machine can only read the symbol it just wrote down. Thus, its behaviour in the next step is already predetermined. Therefore, any **finite** sequence of transitions where only the last one move to the right can be performed in a single step, by directly moving to the right and going to the appropriate state. Compressing all the transitions of M in this manner, we obtain a new machine M' that behaves exactly like M but does not make use of the "stay put" option. Thus, M' moves right at each step. But such a transition function is essentially equivalent to the transition function of a DFA, since writing symbols on the tape has no effect, given they cannot be read again by M.

There is one subtle caveat to this construction. It might be that there is a sequence of transitions in M that all stay put and correspond to a loop in the state diagram. But note that if we were to take any such transitions, we would get stuck in an infinite loop. In the DFA we can thus directly move into a dead rejecting state.

We showed that any DFA can be simulated by this Turing machine variant and that for any such Turing machine there is a DFA that accepts the same language. Thus, we conclude that, unlike standard Turing machines, the variant of Turing machines that can only stay put or move right can recognize exactly the regular languages. Since we know that there are non-regular languages that are recognised by a Turing machine, this proves that the given Turing machine model is strictly weaker than the usual model.

6* Construct a Turing machine which decides the language

$$L = \{ww^R \mid w \in \{0,1\}^*\},$$

where w^R denote the reverse of the string w.

Hint: *Repeatedly reduce the length of the input while making some checks.*

Solution: The following Turing machine recognizes the given language. The idea is to start from the beginning, read and remove the first symbol, move to the end and check whether the last symbol is equal to the first symbol, before removing it too. If not, reject the string. If they are equal, return to the beginning and do the same thing again for the shorter string. In each iteration, the machine needs to remember whether it saw a 0 or a 1 at the beginning. This is achieved by having separate states for these two cases: If it sees a 0 at the beginning, it takes the upper path ($q_0 \to q_1$ ...) and checks for a 0 at the end. If instead it sees a 1 at the beginning, it takes the lower path ($q_0 \to q_2 \dots$) and checks for a 1 at the end. Once it sees a blank symbol, it knows that it has matched all the 0s and 1s correctly; hence it accepts the string.