Tutorial 4

Exercise 1 (compulsory)

Consider the following two claims:

> **Claim 1:** Every language L which is a subset of A_{TM} ($L \subseteq A_{TM}$) is undecidable.
>
> **Claim 2:** Every language L which is a superset of A_{TM} ($A_{TM} \subseteq L$) is undecidable.

Which of these claims are true? Provide the right arguments or give counter-examples.

Solution:
Both claims are wrong. For Claim 1 consider the language $\emptyset$. This is surely a decidable language (try to find a TM which does not accept any string, it is easy) and at the same time it is a subset of A_{TM}. For Claim 2 consider the language Σ^*. It contains the language A_{TM} but it is decidable (again try to find a TM which accepts all strings, this is easy).

Exercise 2 (compulsory)

Which of the following languages are decidable? If you claim that a particular language is decidable, provide a decider for the language.

- $L_1 \stackrel{\text{def}}{=} \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFA and } L(A) \cap L(B) = \emptyset\ \}$
- $L_2 \stackrel{\text{def}}{=} \{\langle M\rangle \mid M \text{ is a TM and } M \text{ contains more than 5 states}\ \}$
- $L_3 \stackrel{\text{def}}{=} \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w \text{ in less than 1000 computational steps}\ \}$
- $L_4 \stackrel{\text{def}}{=} \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w \text{ in a finite number of computational steps}\ \}$
- $L_5 \stackrel{\text{def}}{=} \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ on } w \text{ halts either in accept or reject state}\}$

Solution:

- The language L_1 is decidable. A decider for L_1 would work as follows:

  > ”On input $\langle A, B\rangle$:
  > 1. Construct a DFA C such that $L(C) = L(A) \cap L(B)$.
  > 2. Test whether $L(C) = \emptyset$ or not. If yes then accept else reject.”

  The step 1. is surely algorithmic using the classical product construction (covered in the formal automata course). Step 2. is also algorithmic because we already discussed that the emptiness problem E_{DFA} is decidable for DFA.

- The language L_2 is decidable. A decider M_2 for L_2 would on the input $\langle M\rangle$ do the following. It would inspect the description of M present on the first tape and count the total number control states in M. Should the number be greater than 5 then M_2 would accept, otherwise M_2 would reject. This algorithm will surely terminate and hence we have a decider M_2 for L_2.

- The language L_3 is decidable. Consider the following Turing machine M_3 deciding L_3:

M_3 = "On input $\langle M, w\rangle$:
1. i:=1;
2. Simulate one step of M on w.
3. If M accepted w then M_3 accepts.
   If M rejected w then M_3 rejects.
   If i $\geq$ 1000 then M_3 rejects.
4. Else i:=i+1; goto step 2."

Clearly the machine M_3 is decider as the main loop will run at most 1000 times and $L(M_3) = L_3$.

- The language L_4 is equal to the language A_{TM} and hence we know that it is undecidable.
- The language L_5 is undecidable. A proof of it is given i Exercise 4.

Exercise 3 (compulsory)

In the proof of Theorem 4.11 we construct a machine D and reach a contradiction by running D on the input $\langle D\rangle$. Why could we not instead of constructing D continue the proof as follows?

Construct the machine D_1 :

D_1 = "On input $\langle M\rangle$, where M is a TM:
1. Run H on $\langle M, \langle M\rangle\rangle$.
2. Return the answer of H, i.e. if H accepted, then D_1 accepts, if H rejected, then D_1 rejects."

We see that D_1 accepts $\langle D_1\rangle$ if and only if D_1 accepts $\langle D_1\rangle$. This is obviously true and therefore there is no contradiction and there H must exist. Consequently, A_{TM} must be decidable.

Justify your answer.

Solution:
The attempted line of reasoning has two problems:

1. A contradiction will not disappear, simply because you deliberately do not take the steps that you know will cause it to appear.
2. If we wanted to show that A_{TM} were decidable, we would need to construct a decider for A_{TM}. From the fact that D_1 accepts $\langle D_1\rangle$ if and only if D_1 accepts $\langle D_1\rangle$ we cannot of course conclude the existence of such a decider H.

Exercise 4 (not compulsory but highly recommended!)

Using the diagonalization method show that the language

$$HALT_{TM} \stackrel{\text{def}}{=} \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ on } w \text{ halts either in accept or reject state}\}$$

is undecidable. Hint: modify slightly the proof of undecidability A_{TM}.

Solution:
By contradiction assume that there is a decider H for $HALT_{TM}$:

$$H(\langle M, w\rangle) = \begin{cases} \underline{\text{accept}} & \text{if } M \text{ halts on } w \\ \underline{\text{reject}} & \text{if } M \text{ loops on } w \end{cases}$$

From H we can build the following Turing machine D.

D = "On input $\langle M \rangle$
1. Run H on $\langle M, \langle M \rangle \rangle$.
2. If H accepted then D will enter an infinite loop. If H rejected then D accepts."

What happens if we run D on $\langle D \rangle$?

1. D halts on $\langle D \rangle$, but then H rejected $\langle D, \langle D \rangle \rangle$ and hence D looped on $\langle D \rangle$, contradiction!
2. D loops on $\langle D \rangle$, but then H accepted $\langle D, \langle D \rangle \rangle$ and hence D halted on $\langle D \rangle$, contradiction!

Clearly either 1. or 2. has to happen but in both cases we get a contradiction. This implies that D cannot exist, and so H cannot exist either (D was built from H). This means that $HALT_{TM}$ is undecidable.

Exercise 5 (optional)

Let us call a Turing machine *repetitive* (abbriviated by RTM), if it *only* loops forever if we encounter the same configuration C more than once during a computation. Is the acceptance problem decidable for the class of repetitive Turing machines? First, define explicitly the language you want to show is decidable/undecidable and then give the arguments.

Solution:
The answer depends little bit on how you formally define the problem. One option is to define the following language:

$$A_{RTM} \stackrel{\text{def}}{=} \{\langle M, w \rangle \mid M \text{ is an RTM and } M \text{ accepts } w\}.$$

Now, the decider for A_{RTM} should consider any input string and accept only those that encode an RTM and a string accepted by it. However, the input $\langle M, w \rangle$ could also contain an ordinary TM, which the decider should reject if M is not an RTM. However, the problem whether a given TM is an RTM is undecidable, so A_{RTM} is undecidable too. You can try to prove that the language $\{\langle M \rangle \mid M \text{ is an RTM}\}$ is undecidable by reduction from e.g. $HALT_{TM}$.

On the other hand, one can understand the problem like this. Given an RTM M (that we know is an RTM), we define the language:

$$A_M \stackrel{\text{def}}{=} \{w \mid M \text{ accepts } w\}.$$

Now the language A_M is decidable for any given RTM M. We can detect, given an RTM M, when on the given input w the machine enters an infinite loop. We run M on w and save all the configurations that we encounter on an additional tape. After each step we check if the recent configuration has occurred previously; if it did, we know that M has entered an infinite loop and that consequently w will be rejected.

The following decider for A_M uses three tapes:

"On input w:

1. Place w on tape 2.
2. Copy the current configuration on tape 2 to tape 3 and write a # as separator.
3. Simulate one step of M on tape 2.
4. If the resulting configuration is accepting, then accept.
5. Else, if the new configuration is rejecting, then reject.
6. Else compare the current configuration with the configurations on tape 3. If the new configuration already appears on tape 3, then reject. Otherwise, go to step 2."