Self-Study Exercises 3

Exercise 1: Test your understanding

Assume $A \leq_m B$, i.e., that language A is reducible to language B. Which of the following claims hold?

1. A halting DTM M for the language A (i.e., $L(M) = A$) can be used to show that the language B is computable.
2. If A is computable, then B is computable too.
3. If A is not computable, then B is not computable too.

Solution:

1. WRONG: It should be the other way around: "A halting DTM for the language B can be used to show that the language A is computable." See also the next item.
2. WRONG: Notice that "A is computable" is equivalent to there being a halting DTM for A; so this item is almost identical to the previous item. It should again be the other way around.

   Consider, e.g., the case where A is the empty language (which is clearly computable) and B is the halting problem HP. The empty set is reducible to the halting problem HP (make sure you understand why this is the case!), but HP is not computable.
3. RIGHT.

Exercise 2: Computably-enumerable languages

Which of the following languages are computably-enumerable?
If a language is computably-enumerable, (informally) describe a corresponding Turing machine. Otherwise, you do not have to give a proof.

1. $L_1 = \{\ulcorner M \urcorner \mid M \text{ accepts at least one word}\}$
2. $L_2 = \overline{L}_1$ (i.e., the complement of L_1)
3. $L_3 = \{\ulcorner M \urcorner \mid M \text{ accepts at most one word}\}$
4. $L_4 = \overline{L}_3$ (i.e., the complement of L_3)
5. $L_5 = \{\ulcorner M \urcorner \mid M \text{ accepts exactly one word}\}$
6. $L_6 = \overline{L}_5$ (i.e., the complement of L_5)

Side note: Some of these languages may remind you of (safety and liveness) properties (from the course *Models and tools for cyber-physical systems*).

Solution:
For languages that are not computably-enumerable, we give an intuitive argument. We would, however, need to use a formal proof (e.g., by reduction) for a proper argument.

1. Computably-enumerable.

   We construct a DTM M_1, which works as follows. First, it checks that its input is the encoding of a DTM M. Then, it enumerates all pairs (w, n) of input words w and natural numbers n, and for each such pair, it simulates M on w for n steps. Finally, if M accepts, then M_1 also accepts; if M rejects, M_1 continues the search.

   We argue why $L(M_1) = L_1$. Clearly, M_1 only accepts valid encodings of DTMs M. If there is a word w such that M accepts it, that will happen after a finite number of steps, say, n_0, and so the pair (w, n_0) will eventually be enumerated and M_1 will accept.

2. Not computably-enumerable.

   Argument: If L_2 was computably-enumerable, then L_1 would be computable. While we have not proven that this is not the case, there is a simple reduction from the acceptance problem AP (bonus exercise: show this reduction!).

3. Not computably-enumerable.

   Argument: To accept a word in L_3 (i.e., a DTM that accepts at most one word), we would have to simulate it on all inputs, which is not possible.

4. Computably-enumerable.

   We construct a DTM M_4, which works as follows. First, it checks that its input is the encoding of a DTM M; if not, it accepts (because invalid encodings are part of L_4). Otherwise, M_4 follows almost the same algorithm as M_1. But instead, when M accepts a word, M stores this information and continues with the next pair. If M ever accepts a second word, M_4 accepts (because then $\ulcorner M \urcorner \in L_4$).

   We argue why $L(M_4) = L_4$. Clearly, M_4 accepts all invalid encodings of DTMs. Moreover, if the input is an encoding of a DTM M, if there are at least two words w_1 and w_2 such that M accepts them, that will happen after a finite number of steps, say, n_1 and n_2, and so the pairs (w_1, n_1) and (w_2, n_2) will eventually be enumerated and M_4 will accept.

5. Not computably-enumerable.

   Flawed argument: It is tempting to say that $L_5 = L_1 \cap L_3$, and so we would need to solve L_1 and L_3, but L_3 is not computably-enumerable. But such an argument is generally flawed (why?)!

   Argument: We cannot come up with an enumeration algorithm, since we would not know when to accept. We can search for the first word that is accepted, but then there may always exist another word that is also accepted; hence, we can never terminate the search.

6. Not computably-enumerable.

   Argument: By de Morgan's rule, we have that $L_6 = L_2 \cup L_4$. Similarly to the argument for L_5, we cannot come up with an enumeration algorithm, since we would not know when to accept – it may always be the case that there exists a word that is accepted; hence, we can never terminate the search.

Side note: Notice that L_5 is a strictly harder problem than those we saw before, since neither L_5 nor its complement are computably-enumerable.

Exercise 3: Reduction relation $\leq_m$

Show that the reduction relation $\leq_m$ is reflexive and transitive. In other words, show that for all languages $L_1, L_2, L_3 \subseteq \Sigma^*$ the following holds:

1. $L_1 \leq_m L_1$
2. If $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$.

Solution:
Recall that $A \leq_m B$ means that there is a *computable function* f such that $w \in A \iff f(w) \in B$.

1. Define $f := \mathrm{id}$ as the identity function, which is a computable function (e.g., by a DTM that immediately accepts). Then we have

$$w \in L_1 \iff \underbrace{f(w)}_{w} \in L_1.$$

2. Let f_1 and f_2 be the functions demonstrating $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$. Define $f := f_2 \circ f_1$, i.e., first apply f_1 to the input and then apply f_2 to the result. The composition of two computable functions is computable (e.g., by a DTM that simulates the DTMs for f_1 and f_2). Then we have

$$w \in L_1 \iff f_1(w) \in L_2 \iff \underbrace{f(w)}_{f_2(f_1(w))} \in L_3.$$

Exercise 4: Reduction 1

Recall the following exercise from Lecture 10: Define the languages (now over the binary alphabet)

$$E = \{w \in \{0,1\}^+ \mid w \text{ is even}\}$$
$$O = \{w \in \{0,1\}^+ \mid w \text{ is odd}\}$$

Show that $E \leq_m O$. Give a detailed (but not formal) description of the Turing machine and prove the equivalence $w \in E \iff f(w) \in O$.

Solution:
As we saw in the lecture, we can choose $f(w) = w + 1$. This function is computable, as demonstrated by the following DTM: If the input is empty, accept. Otherwise, perform binary addition of the input and 1.

We show $w \in E \iff f(w) \in O$ by a case distinction. If $w = \varepsilon$, then $w \notin E$ and $f(w) = \varepsilon \notin O$. So let us assume $w \neq \varepsilon$.

$$w \in E \implies w \text{ is even} \implies w + 1 \text{ is odd} \implies f(w) \in O$$

$$w \notin E \implies w \text{ is odd} \implies w + 1 \text{ is even} \implies f(w) \notin O$$

Exercise 5: Reduction 2

Consider the following decision problem:

”Does a given DTM M accept all strings over its input alphabet?”

1. Define this problem as a language U (for “universal”).

   (Assume that the input alphabet is $\{0,1\}$, as we have defined the encoding function only for such DTMs.)

2. Prove that U is not computable by a reduction from the acceptance problem AP.

Solution:

1. $U = \{\ulcorner M \urcorner \mid L(M) = \{0,1\}^*\}$
2. Given a DTM M and an input w for M, let $A_{M,w}$ be the DTM that behaves as follows:

   On input w':

   1. Remove w' from the tape and write w on the tape.
   2. Simulate M on w.
   3. Accept (reject) if M accepts (rejects) w.

Then, we have $\langle \ulcorner M \urcorner, w\rangle \in \text{AP} \iff \ulcorner A_{M,w} \urcorner \in U$. Note that this statement only talks about valid encodings of DTMs and inputs. As usual in a reduction, we also need to take care of words that do not encode such valid encodings.

The function

$$f(x) = \begin{cases} \ulcorner A_{M,w} \urcorner & \text{if } x = \langle \ulcorner M \urcorner, w\rangle \text{ for some DTM } M \text{ and some input } w \\ \varepsilon & \text{otherwise} \end{cases}$$

is computable and we have $x \in \text{AP} \iff f(x) \in U$. Hence, $\text{AP} \leq_m U$.

General remark: It is common to talk about multiple different words in reductions (here: w, w', x). The names can of course be interchanged, and we do not use a consistent naming scheme in the solutions.

Exercise 6: Reduction 3

Let $L = \{\ulcorner M \urcorner \mid M \text{ accepts the word } 101010\}$.

1. Show that $\text{AP} \leq_m L$. (Recall that AP is the acceptance problem.) Give a detailed (but not formal) description of the Turing machine and prove the equivalence $w \in \text{AP} \iff f(w) \in L$.
2. Can you conclude from $\text{AP} \leq_m L$ whether L is computable?
3. Can you conclude from $\text{AP} \leq_m L$ whether L is computably-enumerable?

Solution:

1. We show a reduction with the computable function

$$f(w) = \begin{cases} \ulcorner A_{M,w'} \urcorner & w = \langle \ulcorner M \urcorner, w'\rangle \text{ for some DTM } M \text{ and some input } w' \\ \varepsilon & \text{otherwise,} \end{cases}$$

   where A works as follows given an input x:

   (a) If $x = 101010$, then simulate M on w'. If M accepts (rejects), then $A_{M,w'}$ also accepts (rejects).

   (b) If $x \neq 101010$, any behavior works, e.g., $A_{M,w'}$ rejects.

   We show $w \in \text{AP} \iff f(w) \in L$ by a case distinction. If w does not have the right encoding $\langle \ulcorner M \urcorner, w'\rangle$, then $f(w) = \varepsilon \notin L$. So let us assume $w = \langle \ulcorner M \urcorner, w'\rangle$.

$$\begin{aligned} w \in \text{AP} &\implies w = \langle \ulcorner M \urcorner, w'\rangle \text{ and } M \text{ accepts } w' \\ &\implies f(w) = \ulcorner A_{M,w'} \urcorner \text{ and } A_{M,w'} \text{ accepts } 101010 \\ &\implies f(w) \in L \end{aligned}$$

$$\begin{aligned} w \notin \text{AP} &\implies w = \langle \ulcorner M \urcorner, w'\rangle \text{ and } M \text{ does not accept } w' \\ &\implies f(w) = \ulcorner A_{M,w'} \urcorner \text{ and } A_{M,w'} \text{ does not accept } 101010 \\ &\implies f(w) \notin L \end{aligned}$$

2. L is not computable. This is a direct consequence of the reduction.

3. We cannot conclude anything because AP is computably-enumerable.