Tutorial 8

Exercise 1: Robustness

We want to add another "direction" for the reading head of a Turing machine, namely "0" for "stay." Thus, a transition of the form $\delta(q, a) = (q', b, 0)$ updates the state to q' and changes the symbol at the current cell to b but does not move the head.

Argue that every DTM with the "stay" direction can be outcome-simulated by a standard DTM (i.e., without the "stay" direction). A formal proof is not required, but try to describe the construction.

Solution:
The idea is to simulate a "stay" by going right once and then going left again, thereby ending at the same cell again.[1] More formally, a transition of the form $\delta(q, a) = (q', b, 0)$ will be simulated as follows:

- Replace a by b and go one cell right. Also update the state to remember that we need to go left next and reach state q' then; this is done by changing to a new auxiliary state q'_L.
- When in state q'_L, leave the tape unchanged, move to the left, and go to state q'.

Formally, given a DTM $M = (Q, \Sigma, \Gamma, s, t, r, \delta)$ where $\delta\colon (Q \setminus \{t, r\}) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$, we construct the machine $M' = (Q', \Sigma, \Gamma, s, t, r, \delta')$ with

- $Q' = Q \cup \{q_L \mid q \in Q\}$ (i.e., we add a copy q_L of each state $q \in Q$), and
- for all $q \in Q$ and $a \in \Gamma$, $\delta'(q, a) = \begin{cases} \delta(q, a) & \text{if } \delta(q, a) = (q', b, -1) \text{ for some } q' \in Q, b \in \Gamma, \\ \delta(q, a) & \text{if } \delta(q, a) = (q', b, +1) \text{ for some } q' \in Q, b \in \Gamma, \\ (q'_L, b, +1) & \text{if } \delta(q, a) = (q', b, 0) \text{ for some } q' \in Q, b \in \Gamma, \end{cases}$

  and $\delta'(q_L, a) = (q, a, -1)$.

Then, one can show by induction over $n \in \mathbb{N}$ the following correctness statement:

> Let α_w be the initial configuration of M on w (which is also the initial configuration of M' on w) and let β be an arbitrary configuration of M (which is also a configuration of M'). Then, $\alpha_w \vdash_M^n \beta$ implies $\alpha_w \vdash_{M'}^* \beta$.

The induction is straightforward (moving the reading head of M to the left or to the right can directly be simulated by M'; not moving the head of M can be simulated by two moves of M'), so let us argue that M' outcome-simulates M:

- If M halts on w, then $\alpha_w \vdash_M^n \beta$ for some halting configuration β. Then, by the correctness statement above, we have $\alpha_w \vdash_{M'}^* \beta$, i.e., M' also reaches a halting configuration when started with input w. Hence, M' halts on w.
- If M accepts w, then $\alpha_w \vdash_M^n \beta$ for some accepting configuration β. Then, by the correctness statement above, we have $\alpha_w \vdash_{M'}^* \beta$, i.e., M' also reaches an accepting configuration when started with input w. Hence, M' accepts w.

Note: This is just one example of how this problem can be solved. Solutions different from the above presented one can still be correct.

[1] Bonus question: Why not left and then right?

Exercise 2: Challenge 1

In the lecture, we have introduced Turing machines with tapes that are infinite in one direction. Sometimes, it is useful to consider Turing machines whose tape is infinite in both directions (doubly-infinite).

1. Formally define such a machine, its configurations, and the notion of successor configuration.
2. Argue that every DTM with a doubly-infinite tape can be outcome-simulated by a DTM with a singly-infinite tape.
3. Argue that every DTM with a singly-infinite tape can be outcome-simulated by a DTM with a doubly-infinite tape.

Thus, DTMs are also robust with respect to changing the format of the tape.

Solution:

1. We begin by defining a Turing machine with doubly-infinite tape and its semantics formally, following the definitions for DTMs with singly-infinite tape as introduced in the lecture and the book.

   Formally, a DTM with doubly-infinite tape still has the form $M = (Q, \Sigma, \Gamma, s, t, r, \delta)$ as a standard DTM. However, a configuration now has the form $[q, \tau, \ell]$ where

   - $q \in Q$ is the current state,
   - $\tau \colon \mathbb{Z} \to \Gamma$ is the tape content (where we index cells with (possibly nonpositive) integers), and
   - $\ell \in \mathbb{Z}$ is the current position of the head (which is a (possibly nonpositive) integer).

   Furthermore, the initial configuration on a word $w_1 \dots w_k$ is $[s, \tau, 1]$ where

   $$\tau(n) = \begin{cases} w_n & \text{if } n \in \{1, \dots, k\} \\ \text{␣} & \text{otherwise.} \end{cases}$$

   Accepting, rejecting, and halting configurations are defined as before.

   The notion of successor configurations has to be adapted: Now we do not have to worry about the special case of the head being stuck at the left end of the tape. Hence, given a configuration $[q, \tau, \ell]$, let $\delta(q, \tau(\ell)) = (p, a, d)$. Then the unique successor configuration of $[q, \tau, \ell]$ is the configuration $[p, \tau', \ell + d]$ with

   $$\tau'(n) = \begin{cases} a & \text{if } n = \ell, \\ \tau(n) & \text{otherwise.} \end{cases}$$

   The definitions of runs, accepting runs, and rejecting runs are as for standard machines. This completes the definition of the new machine model.

2. Given a DTM M with a doubly-infinite tape, we construct a (standard) DTM M' with a singly-infinite tape that outcome-simulates M, i.e., if M halts on (resp. accepts) an input w, then so does M'. Our idea is based on the fact that even a DTM with doubly-infinite tape can only visit finitely many cells in finitely many steps, i.e., only finitely many cells are ever nonempty. This finite region of the tape can therefore also be stored in a singly-infinite tape. If we need one more cell to the left, we can just move the finite region on the tape one cell to the right.

   To delimit this region, M' has two additional tape symbols ℓ and r that are not in the tape alphabet of M, which we will use to mark the left end of the tape and the right end of the tape used so far. Then, whenever M wants to move left to a new cell (which M' may not always be able to do on its left-bounded tape), if M' reaches the leftmost cell, which is marked by ℓ, M' pauses the simulation

of M and shifts the whole tape content one position to the right, thereby freeing a new empty cell at the left. In order for the shifting to terminate, we use the right-end marker r, i.e., we only shift up to that position. Similarly, when new empty cells are required at the end of the finite region, we have to move the right-end marker r. Also, note that while M' moves the markers to create new empty cells, it has to remember the current state and head position when the simulation was interrupted. This can easily be done by either writing this information on the tape or by storing it in the states used to create the new empty cells.

More precisely, on input w, the DTM M' with singly-infinite tape does the following:

(1) First, replace the input w by $\ell w r$ on the tape so that ℓ is in the first cell.
(2) Move head to first position of w (cell 2).
(3) If the accepting state of M is reached, accept. If the rejecting state of M is reached, reject.
(4) Otherwise, simulate one transition of M.
(5) If the head now points to a cell containing ℓ, go to step 6. Otherwise, if the head now points to a cell containing r, go to step 9. Otherwise, go to step 3.
(6) Shift the content of the tape (all except the leftmost cell marked with ℓ) up to (and including) the "rightmost" cell marked r one cell to the right, thereby creating a new blank cell right after the position marked with ℓ.
(7) Move the head to that new blank cell and switch back to the state where we interrupted the simulation.
(8) Go to step 3.
(9) Replace the r by a $\sqcup$, write r in the next cell to the right, and move the head to the new blank.
(10) Go to step 3.

The resulting DTM M' outcome-simulates M, which can be shown using a correctness statement similar to the one presented in Exercise 1.

3. Now, we show how to outcome-simulate a DTM with a singly-infinite tape by a DTM with a doubly-infinite tape. This is rather straightforward, as a doubly-infinite tape contains a singly-infinite tape. We just need to implement the special case where the head of a DTM with a singly-infinite tape stays at the first cell if it moves left in the first cell. To this end, we mark cell zero (the first cell to the left of valid indices of a singly-infinite tape) by a special symbol $\#$ that is not in the tape alphabet of M. Whenever this cell is reached, the head is moved one cell to the right to correctly implement the special case.

   Formally, given a DTM $M = (Q, \Sigma, \Gamma, s, t, r, \delta)$ with a singly-infinite tape, we construct a DTM $M' = (Q', \Sigma, \Gamma \cup \{\#\}, s', t, r, \delta')$ with a doubly-infinite tape that outcome-simulates M as follows:

   - $Q' = Q \cup \{s', q_\#\}$ (where s' and $q_\#$ are not in Q),
   - δ' is defined as follows:
     - $\delta'(s', a) = (q_\#, a, -1)$ for $a \in \Gamma$: at the start, leave the first cell unchanged and move the head to the left.
     - $\delta'(q_\#, \sqcup) = (s, \#, +1)$: write a $\#$ in the empty cell to the left of the input, move the head to the first letter of the input, and go to the initial state of M.[2]
     - $\delta'(q, a) = \delta(q, a)$ for $q \in Q$ and $a \in \Gamma$: simulate M.
     - $\delta'(q, \#) = (q, \#, +1)$ for $q \in Q$: if the head reaches the cell marked with $\#$ (i.e., the special case is triggered), move the head back to the right without changing the state.

   The resulting DTM M' outcome-simulates M, which can be shown using a correctness statement similar to the one presented in Exercise 1.

Note that there are again multiple ways this problem can be solved.

[2] $\delta(q_\#, a)$ for $a \neq \sqcup$ can be defined arbitrarily, as it will never be reached; but it needs to be defined, e.g., $\delta'(q_\#, a) = (q_\#, a, +1)$, since δ must be a total function.

Exercise 3: Test your understanding

Which of the definitions of nondeterministic Turing machines (NTMs) given below is correct?

1. An NTM M accepts input w if there exists more than one computation of M on w such that M halts in the state t.
2. An NTM M accepts input w if no computation of M on w halts in the state r.
3. An NTM M accepts input w if there exists a computation of M on w such that M halts in the state t.
4. An NTM M accepts input w if there exists exactly one accepting computation of M on w.
5. An NTM M accepts input w if for every computation of M on w we have that M halts in the state t.

Solution:

1. WRONG: An NTM accepts if there is at least one computation in which the NTM halts in state t, but there need not be more than one such computation.
2. WRONG: "No computation of M halts in r" still allows all computations to loop. In this case, no accepting configuration is reached and the NTM rejects.
3. RIGHT.
4. WRONG: There could be more than one accepting computation and our definition from the slides allows for that.
5. WRONG: Not *all* computations need to halt in the accepting state t, just at least one. It is irrelevant for acceptance how the other runs behave.

Exercise 4: Constructing NTMs

1. Give an NTM for the language of words over the alphabet $\{0, 1\}$ that contain the infix 110. Specify all components of the NTM.
2. A natural number $n > 1$ is composite if it is not prime, i.e., it is the product of two numbers greater than 1. So, the first composite numbers are $4 = 2 \cdot 2$, $6 = 2 \cdot 3$, , $8 = 2 \cdot 4$, , $9 = 3 \cdot 3$, etc.

   Describe (informally) a halting NTM accepting the language $\{a^n \mid n \text{ is composite}\}$ over the alphabet $\{a\}$.

Solution:

1. Note that this language is regular, so we construct an NTM that essentially behaves like an NFA for that language: we guess nondeterministically the place where the infix 110 starts and then check whether this guess was correct.

   If the infix is in the word, then one guess will be correct. If it is not in the word, then all guesses will be wrong.

   Formally, we define $M = (Q, \Sigma, \Gamma, s, t, r, \delta)$ with

   - $Q = \{s, t, r, q_1, q_{11}\}$ where we use the states q_1 and q_{11} to remember that we have seen the infix 1 and 11, respectively.
   - $\Sigma = \{0, 1\}$,
   - $\Gamma = \{0, 1, \sqcup\}$, and
   - δ is given by the following table:

δ	0	1	$\sqcup$
s	$\{(s,0,+1)\}$	$\{(s,1,+1),(q_1,1,+1)\}$	$(r,\sqcup,+1)$
q_1	$\{(r,0,+1)\}$	$\{(q_{11},1,+1)\}$	$\{(r,\sqcup,+1)\}$
q_{11}	$\{(t,0,+1)\}$	$\{(r,1,+1)\}$	$\{(r,\sqcup,+1)\}$

Note that the NTM accepts as soon as the infix 110 has been found, as there is no need to read the rest of the word.

It is instructive to construct the computation trees for some short inputs to recall how nondeterminism is used to accept this language.

2. Given an input a^n, use nondeterminism to guess two numbers $1 < n_0 \leq n$ and $1 < n_1 \leq n$ and then compute $n_0 \cdot n_1$. If this is equal to n, then accept, otherwise reject.

   If n is composite, there exists a guess such n_0 and n_1, and thus at least one computation is accepting. Hence, the NTM accepts n.

   If n is not composite, there are no such n_0 and n_1 and all computations are rejecting. Hence, the NTM rejects n.

   Note that the NTM is halting because there are only finitely many numbers to guess (bounded from above by n), and so the branching is finite.

Exercise 5: Challenge 2

In this exercise, we explore the origin of the name "computably-enumerable".

An *enumerator* E is a two-tape DTM that prints words by writing them on the second tape and entering a special state q_{print}. It may then empty the second tape and print the next word, etc. The language generated by an enumerator is the set of all words that are printed on the second tape.

An enumerator is a *lexicographical enumerator* if it prints words in the lexicographical order, meaning that once a word w is printed, all other words that are printed afterward are in lexicographical order strictly larger than w, in particular implying that their length is at least $|w|$.

1. Prove that the class of languages generated by lexicographical enumerators is equal to the class of computable languages.

2. Prove that the class of languages generated by enumerators is equal to the class of computably-enumerable languages.

Note: To prove that two classes of languages are equal, we have to prove that a language is in one class if and only if it is in the other class.

Solution:

1. We have to show two directions: a) if L is a language generated by a lexicographical enumerator, then L is a computable language, and b) if L is a computable language, then L is generated by some lexicographical enumerator.

   To prove a), we consider two cases. Case 1: If L is a finite language, then it is clearly a computable language (see the first lecture). Case 2: Let L be an infinite language generated by a lexicographical enumerator E. We construct a halting multi-tape DTM M for the language L in order to prove that L is computable:

   "On input w:

   1 Run the enumerator E (on two tapes) until it prints a word w'.
   2 If $w = w'$, **accept**.
   3 If $|w| < |w'|$, **reject**.
   4 Return to step 1 and continue running the enumerator."

Now we first argue that M is a halting DTM, i.e., that on any input w it halts after finitely many steps. Clearly, as L is an infinite language, E keeps printing infinitely many words. However, there are only finitely many words of length at most $|w|$ and hence eventually the enumerator E must print a word of length larger than $|w|$, implying that M will reject (unless it accepted before). The fact that the words are printed in lexicographical order guarantees that once E prints a word w' which is longer than w (and hence comes in lexicographical order after w), it is not possible that w is printed afterward and the machine M can reject the input. Thus, M is a halting DTM with $L(M) = L$.

To prove b), let L be a computable language that is accepted by a halting DTM M. We construct a lexicographical enumerator E generating L as follows:

1 Let $w := \varepsilon$.
2 Run M on w and if M accepts, print w.
3 Let $w := w'$ where w' is the next word in the lexicographical order after w. Go to step 2.

Clearly, the enumerator prints only words in L, and does so in lexicographical order. Moreover, because M is halting and does not loop on any word, each accepted word will be eventually printed.

2. Next, we explain how to generalize the constructions given above to computably-enumerable languages. Again, we have to show two directions: a) if L is a language generated by an enumerator, L is a computably-enumerable language, and b) if L is a computably-enumerable language, it is generated by some enumerator.

   To prove a), assume L is generated by an enumerator E. We construct a multi-tape DTM M (not necessarily halting) for L in order to prove that L is a computably-enumerable language:

   "On input w:

   1 Run the enumerator E (on two tapes) until it prints a word w'.
   2 If $w = w'$, **accept**.
   3 Return to step 1 and continue running the enumerator."

   Now, if w is in L, then E will print it eventually, which means that M will accept it. If w is not in L, then E will never print it, which means M will not accept it (but also not reject it). Hence, we have $L(M) = L$ as required.

   To prove b), let L be a computably-enumerable language with some DTM M (not necessarily halting) such that $L = L(M)$. We construct an enumerator E generating L as follows:

   1 Let $n := 1$.
   2 Run M on all w with $|w| \leq n$ for n steps. Print all w for which an accepting configuration is reached within these n steps.
   3 Let $n := n + 1$.
   4 Go to step 2.

   Clearly, the enumerator prints only words in L as required. Moreover, let $w \in L$. We need to show that w is eventually printed by E. As w is in $L = L(M)$, there is some m so that M accepts w with a run with m steps. Then, w is printed by E when the variable n has value $\max\{|w|, m\}$. Altogether, E generates exactly L.