Theory of Computer Science

G. Röger — University of Basel

Spring Term 2020 — Computer Science

Exercise Sheet 9 — Solutions

Exercise 9.1 (Turing machines; 2 marks)

We defined the tape of a Turing machine to be infinite in both directions. An alternative definition uses a tape that is only infinite in one direction. Formally, this can be achieved by changing the definition of a *step* and leaving all other definitions the same: in the definition on slide C7.16, we change the third case from

$$\langle \varepsilon, q, b_1 \dots b_n \rangle \vdash_M \langle \varepsilon, q', \square c b_2 \dots b_n \rangle \qquad \text{if } \langle q', c, L \rangle \in \delta(q, b_1), n \geq 1$$

to

$$\langle \varepsilon, q, b_1 \dots b_n \rangle \vdash_M \langle \varepsilon, q', c b_2 \dots b_n \rangle \qquad \text{if } \langle q', c, L \rangle \in \delta(q, b_1), n \geq 1.$$

Turing machines with this step model behave in the way as our Turing machines except if they try to move the head to the left of the first position. In this case the head just remains on the first position.

Using a doubly infinite tape does not make our Turing machines more expressive than machines that use a tape that is only infinite in one direction. Explain how a given Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, E \rangle$ with a two-sided infinite tape can be transformed to a machine M' with single-sided infinite tape, such that M' accepts the same language as M.

Note: A proof sketch is sufficient here, but you should explain what kind of additional symbols and states M' requires and what the main idea of the transformation is.

Solution:

There are different possible transformations and we explain two of them here. In the first case, the whole tape content is shifted right by one position whenever the head would move over the left end of the tape. In the other variant, the tape is "cut" to the left of the input into two singly infinite tapes that are then interleaved, i.e., the new tape uses positions form the left tape and positions from the right tape (represented from right to left) alternatingly.

Variant 1:

The main idea of the transformation is to move all tape content one step to right whenever the read head reaches the first position. The technical problem with this is that we cannot use the $\square$ to detect the end of the tape content. (The machine M could write a $\square$ in the middle of the word. We thus use two new symbols $\boxed{S}$ and $\boxed{E}$ for the start and end of the current tape content. At the beginning of M', we replace the first symbol by $\boxed{S}$ and move the head one position to the right. We then repeatedly replace the symbol that is currently under the head with the symbol that was last replaced. To make this possible, we need a new state q_x for every tape symbol $x \in \Sigma$. When we read a y in state q_x we replace it by an x and transition to state q_y (this way, the next symbol will be replaced by a y). We proceed this way until we reach the first $\square$ and append an $\boxed{E}$. Afterwards, M' moves to the left until it reaches the start of the input (one position right of $\boxed{S}$). With the last movement, we transition to the initial state of M. Instead of the original configuration $\langle \varepsilon, q_0, w \rangle$ we are now in the configuration $\langle \boxed{S}, q_0, w\boxed{E} \rangle$.

The machine M' now uses the same states as M but we extend the transition function so every state has two additional outgoing edges for $\boxed{S}$ and $\boxed{E}$. If a $\boxed{E}$ is read, it is replaced by a $\square$ and we write another $\boxed{E}$ one position to the right. Afterwards, M' transitions back to the state where it was before. Since we cannot "save" the state, we have to introduce a new state for this movement for every state of M. After this movement M' is in the same state as before and now reads the newly inserted $\square$; exactly as M would in this situation. We were able to change the configuration from $\langle \boxed{S}\alpha, q, \beta\boxed{E} \rangle$ to the configuration $\langle \boxed{S}\alpha, q, \beta\square\boxed{E} \rangle$.

If we instead read a $\boxed{S}$, we have to move all tape content one position to the right. The symbol $\boxed{S}$ remains in the first position, we replace the second symbol with a $\Box$, the third symbol with the second symbol, and so on. this movement works exactly as the movement at the start of M', with the only exception that we now also move $\Box$ and repeat until we moved $\boxed{E}$. Afterwards, we move the head back left until we reach $\boxed{S}$ and move one position to the right from there. In this last step, we go back to the state that triggered the movement. As described above, we need this structure once for every state of M, so we know which state to return to after the movement. With this, we are able to change the configuration from $\langle \varepsilon, q, \boxed{S}\beta\boxed{E}\rangle$ to $\langle \boxed{S}, q, \Box\beta\boxed{E}\rangle$.
With these extensions, M' behaves exactly as M, only that it it sometimes interrupted, to move the tape content and insert blanks.
Variant 2:

For the explanation of this idea, we rename the infinitely many tape positions in a way that p_0 is the first position of the input and $p_1, p_2, \dots$ are the positions to the right. Left of p_0 are the positions (from right to left) $p_{-1}, p_{-2}, \dots$.
The idea is to simulate the machine with TM with singly infinite tape that represents the original tape content in the order $p_0, p_{-1}, p_1, p_{-2}, p_2, \dots$. The main idea is that the head moves two position to the left if it is on a positive index and should move to the left; and moves two position to the right it it is on a non-negative position and should move to the right; etc. For negative indices, we also need to reverse the direction. Of course, the indices do not really exist, so we have to include them in the program of the TM.
To do so, we use six additional states $q_+, q_-, q_+^R, q_-^R, q_+^L$ and q_-^L for every state q of the original TM. The subscript denotes whether we are currently moving on the positive or negative indices. States with a superscript of R or L are used as intermediate states to show that an additional step in the respective direction is necessary. The additional step then leads to the corresponding state without superscript.
We have not covered the transition between positive and negative positions yet. To achieve that correctly, we first shift the whole tape content one position to the right and mark the left end of the tape with a new symbol # (using additional states similar to variant 1).
If we now are in a state q_+^L (that is, we moved from a non-negative position to the right but only moved one step so far), we now move an additional step the left and in state q_+ if we do not read #. Otherwise, we started the move in position 0 and now want to end up on position -1 which is two positions to the right of the current cell. The TM thus moves the head there using an additional state q_0^R and then transitions to state q_-. The change from negative positions to the positive ones works similarly but an original movement to the right still first triggers two movements to the left and only then, we test whether the head has to be moved back one step the right to the non-negative positions.

Exercise 9.2 (Multiplication is Turing-computable; 2 marks)

Describe the main idea of a proof showing that multiplication of binary numbers is Turing-computable, i.e., describe a Turing machine that computes mul^{code} for the function $mul : \mathbb{N}_0^2 \to_{\text{p}} \mathbb{N}_0$ with $mul(n_1, n_2) = n_1 \cdot n_2$.
Note: You may use the fact that addition is Turing computable and a high-level description of the Turing machine is sufficient.

Solution:

The main idea to initialize a number with 0 and then add the number n_2 a total of n_1 times. To count how often we still have to add n_2, we can reduce n_1 by 1 after every iteration and stop when it reaches 0. The individual components are all known from the lecture (addition of two numbers, subtraction of 1) or easy to implement (test for 0, initialize part of the tape to 0).
The problem is that the computation of a Turing machine consumes its input and may write into other areas outside of its input. For example, after the addition, the two added number are not longer on the tape. We thus have to separate several areas on the tape. Every machine for a suboperation is restricted to "its" part of the tape. If such an part is not large enough its size can be increased by moving all other parts (see exercise 9.1). To execute the different operations one

after the other, our machine only has to write the correct inputs to the correct part other tape and transition to the initial state of the specialized machine (e.g., the one for addition). Instead of going to an accepting state, this machine is modified so it transitions to the next state of our machine instead. in this case, for example, n_1 should be reduced by 1 after each addition.

Exercise 9.3 (Composition of computable functions is computable; 2 marks)

Let $f : \Sigma^* \to \Sigma^*$ and $g : \Sigma^* \to \Sigma^*$ be Turing-computable partial functions for an alphabet Σ. Show that the *composition* $(f \circ g) : \Sigma^* \to \Sigma^*$ is also Turing-computable.
In general the composition of two functions is defined as $(f \circ g)(x) = f(g(x))$. Specifically, the value $(f \circ g)(x)$ is undefined if $g(x)$ is undefined.

Solution:

General Ideas:
If f and g are Turing-computable, then there exist DTMs M_f and M_g which compute $f(y)$ given input y, respectively $g(x)$ given input x.
If we start the DTM M_g on an input x where $g(x)$ is defined, then after termination the tape content and the reading head are exactly in the configuration needed for the input for M_f (according to the definition of functions computed by a DTM). Thus it is in this case sufficient to combine M_f and M_g to a new DTM by changing all transitions of M_g which lead to an end state to the start state of M_f. Thus the DTM first computes $g(x)$ given input x. This is then the input y for f, which means the DTM computes $f(y) = f(g(x)) = (f \circ g)(x)$.
Technical Details:
We assume without restriction that the sets of states of M_f and M_g are disjoint (if not, we can rename states as needed) and that the tape alphabets are identical otherwise we can unite the two tape alphabets and arbitrarily choose all new transitions which need to be defined now (for example when M_g reads a symbol only occurring in the tape alphabet of M_f). We can do this arbitrarily because these transitions will never be used.
One technical problem arises though if M_g terminates in a configuration that does not represent a valid computation (if the head is on a wrong position or illegal symbols are on the tape). In this case the machine should not return a valid result, since $(f \circ g)(x)$ is undefined, but we cannot guarantee this without further demands to M_f.
The easiest way to remove this problem is to modify M_g in a way to ensure that it cannot stop in an invalid configuration. For example we could check at the end of the computation of M_g if the tape content has a correct form. One difficulty we have to deal with is to recognize where the visited part of the tape starts and ends. The easiest way to solve this is to never use $\square$ during the calculation of M_g but instead write a new symbol $\hat{\square}$ which behaves exactly as $\square$. Thus we ensure that when checking the result of the computation $\square$ marks both ends of the visited tape. During the final check we then replace $\hat{\square}$ with $\square$.

Exercise 9.4 (Enumerable Functions; 2 marks)

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Specify total and computable functions $f : \mathbb{N}_0 \to \Sigma^*$ which recursively enumerate the following languages. Additionally specify the function values $f(0)$, $f(1)$, ..., $f(5)$. You may use all computable functions which were discussed in the lecture.

(a) $L_1 = \{\mathtt{b}^n\mathtt{a}^{2n} \mid n \in \mathbb{N}_0, n \text{ is even}\}$

Solution:

$$f_{L_1}(x) = \begin{cases} \mathtt{b}^x\mathtt{a}^{2x} & \text{if } x \text{ even} \\ \varepsilon & \text{else} \end{cases}$$

x	0	1	2	3	4	5
$f_{L_1}(x)$	ε	ε	`bbaaaa`	ε	`bbbbaaaaaaaa`	ε

(b) $L_2 = \{w \in \Sigma^* \mid \mathtt{a} \text{ occurs in } w \text{ exactly once}\}$

Solution:

$f_{L_2}(x) = \texttt{b}^{decode_1(x)}\texttt{ab}^{decode_2(x)}$

x	0	1	2	3	4	5
$decode_1(x)$	0	0	1	0	1	2
$decode_2(x)$	0	1	0	2	1	0
$f_{L_2}(x)$	`a`	`ab`	`ba`	`abb`	`bab`	`bba`

Exercise 9.5 (Decidability; 2 marks)
Which of the following statements are true? Justify your answers with one or two sentences each.

(a) If L is decidable, then L is also finite.

Solution:

The statement is false. For example, the language of all words starting with `a` is infinite but decidable.

(b) If L is regular then L is also decidable.

Solution:

The statement is true. For every regular language there is a DFA that can be simulated on an input in finite time to decide if the word is in the language or not.

(c) If L_1 and L_2 are decidable, then $L_1 \cap L_2$ is decidable.

Solution:

The statement is true. Since L_1 and L_2 are decidable, there are Turing machines that compute their characteristic functions. Since those machines always stop, we can compute them one after the other and return 1 if both of them return 1.

(d) If L is context-sensitive, then L is also semi-decidable.

Solution:

The statement is true. Every context-sensitive language is also a Type-0 language and we have shown that these are exactly the semi-decidable languages.