Solutions to Supplementary Problems

S8.1 Extend the notion of a Turing machine by providing the possibility of a *two-way infinite tape*. Show that nevertheless such machines recognise exactly the same languages as the standard machines whose tape is only one-way infinite.

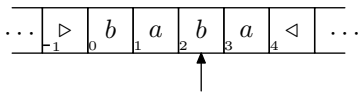
Solution. We shall use the same Turing machine notation as on the lecture slides and the Finnish lecture notes. This is essentially the same as in Sipser's book except that, in the basic version:

- the left end of the tape is marked with a special tape start symbol $\triangleright$ that cannot be overridden and the cursor cannot move past it to the left, and
- the end of the tape is marked with a special tape end symbol $\triangleleft$ (instead of $\sqcup$) that can be overridden and, in such a case, moves automatically to the right.

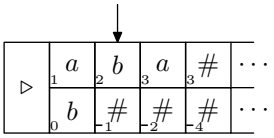
In the Sipser book notation, one has to specially mark the left end of the tape in order to be able to find it later (see Example 3.11); this would add some tedious and uninteresting technical details to the present solution.

A Turing machine with a two-way infinite tape works otherwise in the same way as a standard machine except that the position of the tape start symbol ($\triangleright$) is not fixed, and it can move in a same way as the end symbol ($\triangleleft$). The tape positions are indexed by the set $\mathbb{Z}$ of integers, where 0 denotes the initial position of $\triangleright$.

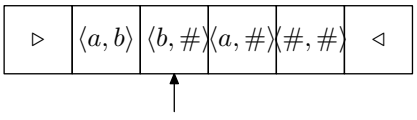
We can simulate such a Turing machine by a two-track one-way tape machine as follows. Conceptually, we divide the tape into two parts: upper and lower. The upper part holds the two-way tape cells $i > 0$ and the lower part cells $i \leq 0$. For example, a two-way tape



is expressed as a one-way tape



As presented in Lemma 8.1 of Lecture 8, the two tracks can be reduced to one by replacing the tape alphabet Γ by a new alphabet $\Gamma' = (\Gamma \cup \{\#\}) \times (\Gamma \cup \{\#\})$, where $\#$ is a new tape symbol that does not occur in the original Turing machine. Each symbol of Γ' thus denotes two symbols of Γ. So, the above example is expressed as



We still need a way to decide which track, i.e. tape-half is being used at each step of the simulation. The easiest way to do this is to define a mirror image state q' for each state q. When the machine is in state q, it examines only the upper track symbol when deciding what move to take next (this corresponds to the simulated tape head being on the right half of the tape). Similarily, in state q' it examines only the lower track symbol (simulated tape head is on the left half). Since the cells on the lower track are presented in a reversed order, all tape head move instructions must be similarly reversed.

Formalisation of the solution

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ be a Turing machine with two-way infinite tape. We form a standard Turing machine as follows:

$$\begin{aligned} M' &= (Q', \Sigma, \Gamma', \delta', q_0, q_{\text{acc}}, q_{\text{rej}}) \\ Q' &= Q \cup \{q' \mid q \in Q\} \\ \Gamma' &= (\Gamma \cup \{\#\}) \times (\Gamma \cup \{\#\}) \end{aligned}$$

The transition function is

$$\begin{aligned} \delta' = {} & \{(q_1, \langle a, \gamma\rangle) \mapsto (q_2, \langle b, \gamma\rangle, \Delta) \mid (q_1, a) \mapsto (q_2, b, \Delta) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q_1, \triangleleft) \mapsto (q_2, \langle b, \#\rangle, \Delta) \mid (q_1, \triangleleft) \mapsto (q_2, b, \Delta) \in \delta\} \cup \\ & \{(q_1, \triangleleft) \mapsto (q_2, \langle \#, \#\rangle, L) \mid (q_1, \triangleleft) \mapsto (q_2, \triangleleft, L) \in \delta\} \cup \\ & \{(q_1, \langle \#, \gamma\rangle) \mapsto (q_2, \langle \#, \gamma\rangle, L) \mid (q_1, \triangleleft) \mapsto (q_2, \triangleleft, L) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q_1, \langle \#, \gamma\rangle) \mapsto (q_2, \langle b, \gamma\rangle, \Delta) \mid (q_1, \triangleleft) \mapsto (q_2, b, \Delta) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q_1', \langle \gamma, a\rangle) \mapsto (q_2', \langle \gamma, b\rangle, \overleftarrow{\Delta}) \mid (q_1, a) \mapsto (q_2, b, \Delta) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q_1', \triangleleft) \mapsto (q_2', \langle \#, b\rangle, \overleftarrow{\Delta}) \mid (q_1, \triangleright) \mapsto (q_2, b, \Delta) \in \delta\} \cup \\ & \{(q_1', \triangleleft) \mapsto (q_2', \langle \#, \#\rangle, L) \mid (q_1, \triangleright) \mapsto (q_2, \triangleright, R) \in \delta\} \cup \\ & \{(q_1', \langle \gamma, \#\rangle) \mapsto (q_2', \langle \gamma, b\rangle, \overleftarrow{\Delta}) \mid (q_1, \triangleright) \mapsto (q_2, b, \Delta) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q_1', \langle \gamma, \#\rangle) \mapsto (q_2', \langle \gamma, \#\rangle, L) \mid (q_1, \triangleright) \mapsto (q_2, \triangleright, R) \in \delta, \gamma \in (\Gamma \cup \{\#\})\} \cup \\ & \{(q, \triangleright) \mapsto (q', \triangleright, R), (q', \triangleright) \mapsto (q, \triangleright, R) \mid q \in Q\} \end{aligned}$$

where $a, b \in \Gamma$ (note: Γ does not include the symbols $\triangleright$ and $\triangleleft$), $\overleftarrow{L} = R$ and $\overleftarrow{R} = L$.

S8.2 Design deterministic Turing machines NEXT and DUP that perform the following tasks:

1. NEXT replaces the string on the tape with the string that is next in the shortlex order[1] (lexicographical order except that shorter strings precede longer ones).
2. DUP copies the tape contents to the end of the tape (e.g., tape contents abb is replaced with $abbabb$).

Solution. (a) The shortlex order $<_{\text{L}}$ on the set Σ^* of all Σ-strings is defined on top of an order $<$ on alphabet Σ. Usually the ordering $<$ is a natural alphabetic or numerical order, for instance

- if $\Sigma = \{a, \ldots, z\}$, then usually $a < b < c < \cdots < z$, and
- if $\Sigma = \{0, 1\}$, then $0 < 1$.

The shortlex order $<_{\text{L}}$ is defined as follows. Let $x, y \in \Sigma^*$, $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_m$. Now $x <_{\text{L}} y$ if either

(i) $n < m$, or

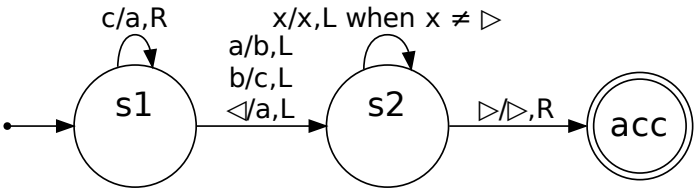
(ii) $n = m$ and there is an $i \leq n$ such that $x_i < y_i$ and $x_j = y_j$ for all $j < i$.

[1] `http://en.wikipedia.org/wiki/Shortlex_order`

As an example, the strings over alphabet $\{0,1\}$ are ordered as follows:

$$\varepsilon <_{\mathrm{L}} 0 <_{\mathrm{L}} 1 <_{\mathrm{L}} 00 <_{\mathrm{L}} 01 <_{\mathrm{L}} 10 <_{\mathrm{L}} 11 <_{\mathrm{L}} 000 <_{\mathrm{L}} 001 <_{\mathrm{L}} 010 <_{\mathrm{L}} 011 <_{\mathrm{L}} \ldots$$

In the following, we illustrate the solution with a Turing machine over the alphabet $\Sigma = \{a, b, c\}$. For the sake of simplicity, we also first assume that the string to be incremented is written in reverse order on the tape. The machine is (again in the notation of the lecture slides and the Finnish lecture notes, as in Problem S8.1):



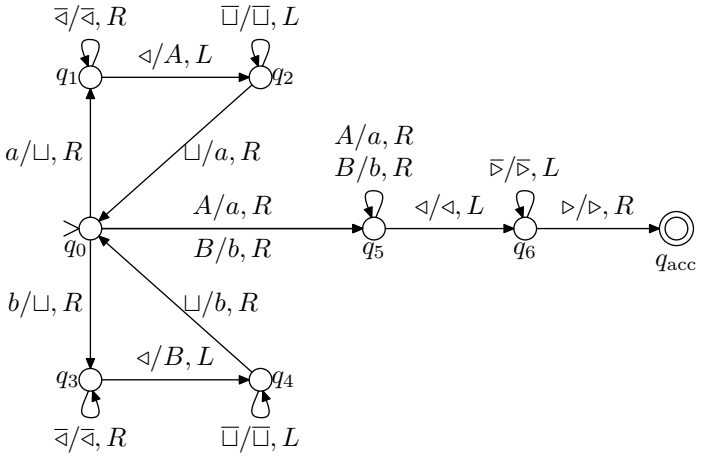
The computation of the machine (in Sipser book notation) on string $bacc$ (reversed to $ccab$ on the tape) is:
$\triangleright s_1\, ccab\triangleleft$
$\triangleright a\, s_1\, cab\triangleleft$
$\triangleright aa\, s_1\, ab\triangleleft$
$\triangleright a\, s_2\, abb\triangleleft$
$\triangleright s_2\, aabb\triangleleft$
$s_2 \triangleright aabb\triangleleft$
$\triangleright q_{\mathrm{acc}}\, aabb\triangleleft$
In order to obtain a solution when the input string is not reversed, one has to take into account the fact that the string length increases by one when computing the successor of a string that consists only of c's. For instance, the successor of ccc is $aaaa$. One solution is to first check whether the string consists of only c's. If so, substitute it with a string of equally many plus one a's. If not, go to the end of the string and apply a machine that performs the same actions as the states s_1 and s_2 above, but from right to left.

(b) We illustrate the solution for the alphabet $\Sigma = \{a, b\}$. Extending the solution to other alphabets is straightforward and the details are omitted here.

The principle of the solution is as follows. The string is copied, symbol by symbol, to the end of the tape. To keep track of which symbol is being copied, a marker symbol $\sqcup$ is inserted in its place temporarily. Moreover, the copied string is first written with uppercase tape symbols $\{A, B\}$ in order to distinguish it from the original string. When all of the original symbol has been copied, the uppercase symbols are replaced with the corresponding lowercase ones, and the tape head is returned to the beginning of the tape.

A detailed state diagram of the machine is as follows:

DUP:



where the abbreviation of form $\overline{\triangleleft}/\overline{\triangleleft}, L$ denotes all the transitions that read (but don't overwrite) a symbol that is not $\triangleleft$ and move the cursor leftwards.

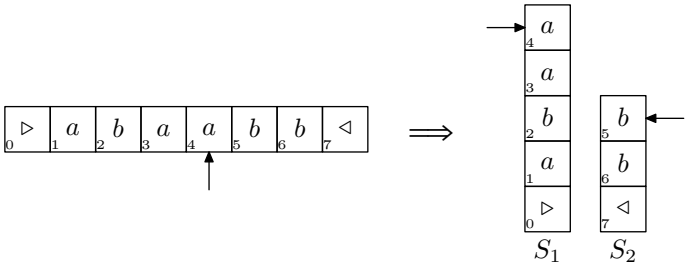
The computation sequence of the machine on input abb is:

$$
\begin{aligned}
\triangleright q_0\, abb\triangleleft \quad &\vdash \quad \triangleright\sqcup q_1\, bb\triangleleft \\
&\vdash^+ \quad \triangleright\sqcup bb\, q_1 \triangleleft \\
&\vdash \quad \triangleright\sqcup b\, q_2\, bA\triangleleft \\
&\vdash^+ \quad \triangleright q_2\, \sqcup bbA\triangleleft \\
&\vdash \quad \triangleright a\, q_0\, bbA \\
&\vdash \quad \triangleright a\sqcup q_3\, bA\triangleleft \\
&\vdash^+ \quad \triangleright a\sqcup bA\, q_3 \triangleleft \\
&\vdash \quad \triangleright a\sqcup b\, q_4\, AB\triangleleft \\
&\vdash^+ \quad \triangleright ab\, q_0\, bAB\triangleleft \\
&\vdash^+ \quad \triangleright abb\, q_0\, ABB\triangleleft \\
&\vdash \quad \triangleright abbA q_5\, BB\triangleleft \\
&\vdash^+ \quad \triangleright abbabb\, q_5 \triangleleft \\
&\vdash^+ \quad \triangleright q_{\text{acc}}\, abbabb\triangleleft
\end{aligned}
$$

where $\vdash$ means that one step is taken and $\vdash^+$ that multiple steps are taken.

S8.3 Show that pushdown automata with *two* stacks (rather than just one as permitted by the standard definition) would be capable of recognising exactly the same languages as Turing machines.

Solution. We first show that a two-stack pushdown automaton can simulate a Turing machine. The only difficulty is to find a way to simulate the Turing machine tape using two stacks. This can be done using a construction that is similar to the one presented in Problem S8.1: the first stack holds the part of tape that is left to the tape head (in reversed order), and the second stack holds the symbols that are to the right of the head.



The computation of the automaton consists of two stages:

1. Initialisation, where the automaton copies the input to stack S_1 one symbol at a time, and then moves it, again one-by-one, to stack S_2. (With the exception of the first symbol).
2. Simulation, where the automaton decides its next transition by examining the top symbol of S_1. If the machine moves its head to left, the top element of S_1 is moved into S_2. If it moves to the other direction, the top element of S_2 is moved to S_1.

A two-stack pushdown automaton formed using these principled simulates a given Turing machine.

Finally, we have to show that we can simulate any two-stack pushdown automaton with a Turing machine. This can be done trivially using a two-tape nondeterministic Turing machine where both stacks are stored on their own tapes.