Fundamentele Informatica 3
Antwoorden op geselecteerde opgaven uit Hoofdstuk 7

John Martin: Introduction to Languages and the Theory of Computation
(fourth edition)

Jetty Kleijn, Rudy van Vliet

Voorjaar 2012

N.B.: in these solutions, we use the *old* notation for configurations of a Turing machine, i.e., the notation from the *third* edition of the book of Martin: (q, xy) instead of xqy.

7.1 See Figure 7.6 (erroneously called Figure 7.5): a TM accepting $\{ss \mid s \in \{a,b\}^*\}$. We give the configuration sequence of the computation for input $aaba$:
$(q_0, \underline{\Delta}aaba) \vdash (q_1, \Delta\underline{a}aba) \vdash (q_2, \Delta A\underline{a}ba) \vdash (q_2, \Delta Aa\underline{b}a) \vdash$
$(q_2, \Delta Aab\underline{a}) \vdash (q_2, \Delta Aaba\underline{\Delta}) \vdash (q_3, \Delta Aab\underline{a}) \vdash (q_4, \Delta Aa\underline{b}A) \vdash$
$(q_4, \Delta A\underline{a}bA) \vdash (q_4, \Delta\underline{A}abA) \vdash (q_1, \Delta A\underline{a}bA) \vdash (q_2, \Delta AA\underline{b}A) \vdash$
$(q_2, \Delta AAb\underline{A}) \vdash (q_3, \Delta AA\underline{b}A) \vdash (q_4, \Delta A\underline{A}BA) \vdash (q_1, \Delta AA\underline{B}A) \vdash$
— $aaba$ is of even length —
$(q_5, \Delta A\underline{A}BA) \vdash (q_5, \Delta\underline{A}aBA) \vdash (q_5, \underline{\Delta}aaBA) \vdash (q_6, \Delta\underline{a}aBA) \vdash$
$(q_8, \Delta A\underline{a}BA) \vdash (q_8, \Delta Aa\underline{B}A) \vdash$ CRASH
or, in other words, $\vdash (h_r, \Delta Aa\underline{B}A)$.
$aaba$ is not of the form ss and not accepted by the Turing machine.

7.2
We first trace the operation of the TM on the example input $x = ab$:
$(q_0, \underline{\Delta}ab) \vdash (q_1, \Delta\underline{a}b) \vdash (q_1, \Delta a\underline{b}) \vdash (q_1, \Delta ab\underline{\Delta}) \vdash$
$(q_2, \Delta a\underline{b}) \vdash (q_5, \Delta a\Delta\underline{\Delta}) \vdash (q_6, \Delta a\Delta b\underline{\Delta}) \vdash (q_7, \Delta a\Delta\underline{b}b) \vdash$
$(q_7, \Delta a\underline{\Delta}bb) \vdash (q_2, \Delta\underline{a}\Delta bb) \vdash (q_3, \Delta\Delta\underline{\Delta}bb) \vdash (q_4, \Delta\Delta a\underline{b}b) \vdash$
$(q_4, \Delta\Delta ab\underline{b}) \vdash (q_4, \Delta\Delta abb\underline{\Delta}) \vdash (q_7, \Delta\Delta ab\underline{b}a) \vdash (q_7, \Delta\Delta a\underline{b}ba) \vdash$
$(q_7, \Delta\Delta\underline{a}bba) \vdash (q_7, \Delta\underline{\Delta}abba) \vdash (q_2, \underline{\Delta}\Delta abba) \vdash (h_a, \Delta\underline{\Delta}abba)$

We now consider arbitrary input x. After going to cell 1 and entering state q_1 the Turing machine moves to the right of (what still remains of) the input string (state q_2); it remembers and erases the last symbol (q_3 for a; q_5 for b) and moves it one cell to the right after which it proceeds (in q_4, resp. q_5) to the first empty cell to the right where it writes a, resp. b. Then it moves back to the left (in q_7) until it encounters an empty cell to the left of which the last symbol of the rest of the input is waiting. The process repeats from q_2 until no input is left, after which the Turing machine goes from q_2 to the accepting state h_a.
We conclude that $(q_0, \underline{\Delta}x) \vdash^* (h_a, \Delta\underline{\Delta}xx^r)$ for all $x \in \{a, b\}^*$: the machine adds the mirror image to a word (and the input is shifted one cell to the right).

7.9 The TM in Figure 7.37 when given an input string 1^i repeatedly divides i by 2 until exactly one 1 remains after which it accepts. Hence the input is accepted if and only if i is a power of 2. Thus the TM accepts the language $\{1^{2^n} \mid n \geq 0\} = \{1, 11, 1111, 11111111, 1^{16}, \ldots\}$.

7.12 Let T be a TM accepting a language L. We modify T in such a way that the resulting TM T' also accepts L and never stops unsuccessfully (the reject state h_r is not needed, neither explicitly nor implicitly).
Recall that T stops unsuccessfully if and only if either it scans a symbol in a state for which combination it does not have an instruction or the head falls off the left side of the tape.

First we consider the problem of the head falling off the tape. We use symbols with a subscript L as an indication of their occurrence in the leftmost cell. We thus have a new symbol Δ_L and new tape symbols a_L for every $a \in \Gamma$ where Γ is the tape alphabet of T. We let T' begin its work on any input x by first changing the Δ in the leftmost cell into Δ_L. Thus from here (configuration $(q_0, \underline{\Delta}_L x)$) it behaves as T unless it sees a subscript L. For these cases, instructions $\delta'(p, a_L) = (q, b_L, D)$ are provided whenever $\delta(p, a) = (q, b, D)$ and D is either R or S (the head moves to the right or stays); for instructions $\delta(p, a) = (q, b, L)$ (when the head would move to the left and fall off), we do nothing. Note that T' (constructed sofar would crash by lack of instructions rather than falling off the tape. The set of accepted words has not been changed.

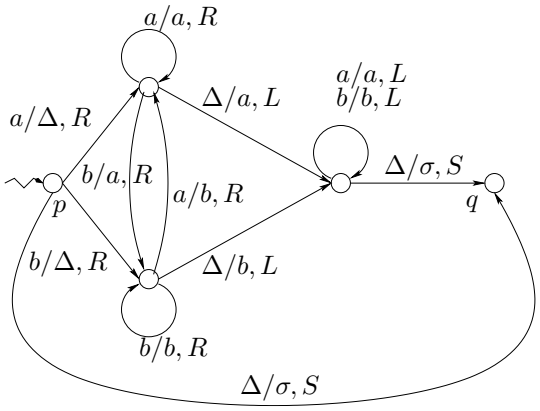
Finally, we add a new state `sink` together with instructions $\delta'(\texttt{sink}, a) = (\texttt{sink}, a, S)$ for all symbols $a \in \Gamma \cup \Gamma_L \cup \{\Delta\}$. Thus once the TM is in `sink` it will remain (loop) there forever scanning the same cell with the same symbol. For every combination (p, a) of a state p and a symbol a for which

there is no instruction (thus in particular for $p = h_r$ which is now considered an ordinary state), we set $\delta'(p, a) = (\texttt{sink}, a, S)$. Hence whenever T would crash because of lack of an instruction (or fall off the tape), the modification now takes care that it moves to `sink` (and only then). Note that adding these instructions does not make the TM non-deterministic nor affects the set of words accepted.

Alternative solution: instead of considering h_r as an ordinary state, we could replace every transition of the form $\delta(p, a) = (h_r, b, D)$ by $\delta(p, a) = (\texttt{sink}, b, D)$.

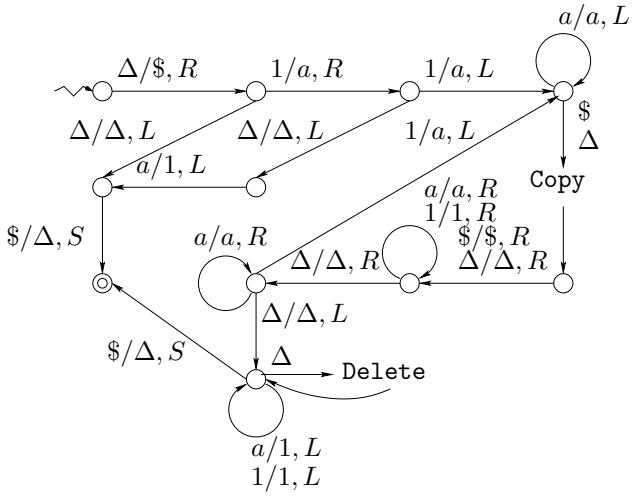
7.13 As mentioned in Example 7.21, during (or after) a computation of a TM there is no (general, effective) procedure to determine the position of the rightmost cell containing a non-blank symbol. Simply walking to the right does not give insight in whether the last non-blank has been seen or whether there might still be another one. As we discuss now also more subtle methods will not work.
Assume we have a module TM T_0 which when started on a tape will move the head to the rightmost position on the tape containg a non-blank (if the tape is completely blank it moves the head to cell 0) and then stop.
First consider a TM T_1 which halts (for some input) in the accepting configuration $(h_a, \underline{\Delta}1)$. Then T_0 begins its work and after some finite time it halts with its head on the 1 in cell 1 with the tape otherwise empty. We now make a TM T_2 which works as follows. It erases its input, moves back to the beginning of the tape and writes 1 in cell 1, then it invokes T_0 but with the following modification: it marks cell 2 with the endmarker # and whenever # is read it is treated as Δ and the marker is shifted one cell to the right. In this way there will always be exactly one # on the tape directly after the rightmost cell ever visited by T_0. Once (the thus) modified version of T_0 has finished its work we stop in the configuration $(h_a, \underline{\Delta}1\Delta^n\#)$ with cell $n + 1$ the rightmost cell ever visited by T_0.
Now we ask T_0 to find the rightmost non-blank cell left by T_2. Since it is deterministic it will proceed as for T_1 beginning from $(h_a, \underline{\Delta}1)$ and will never visit the cell with #. Thus T_0 does not work correctly for T_2.

7.14 We construct a (module) Turing machine *Insert*(σ) with input alphabet Σ which places the symbol $\sigma \in \Sigma \cup \{\Delta\}$ at the current position of the tape head and shifts the tape contents which follow one position to the right. More precisely: $(p, y\underline{z}) \vdash^* (q, y\underline{\sigma}z)$ where p is the initial state of *Insert*(σ) and q its terminating state (h_a if you like); z does not contain Δ's.
The TM achieves this by writing a Δ at the current position if that is not blank (if it is, we know that $z = \Lambda$ and the TM can write σ rightaway and stop); the TM moves to the right, all the time replacing the next symbol by the previous one. When it sees Δ it writes the final symbol, moves back to the left until it sees the blank cell left at the original beginning of z where it writes σ and stops.
Below follows a detailed transition diagram for the case that the input alphabet of the Turing machine consists of a, b only (and so $\sigma \in \{a, b\} \cup \{\Delta\}$). Generalizing this to arbitrary alphabets should be easy.



7.17
c We construct a TM that computes the square of any given positive integer (in unary), thus we aim at $(q_0, \underline{\Delta}1^n) \vdash^* (h_a, \underline{\Delta}1^{n^2})$.
Basically given an input string of n 1's we have to produce another $n-1$ strings of n 1's. Therefore we will use the modules *Copy* and *Delete*, the latter to remove the redundant Δ's inbetween the copied strings.
In order to distinguish the first Δ in cell 0 from other occurrences of Δ we change it till the end of the computation in \$. This implies that we use a slightly extended version of the module *Copy* as described in Example 7.18 (Figure 7.19). The modified version used here treats \$ as a Δ, that is $(p, \underline{\Delta}x) \vdash^* (q, \underline{\Delta}x\Delta x)$ and also $(p, \underline{\$}x) \vdash^* (q, \underline{\$}x\Delta x)$ with p the initial state of *Copy* and q its final state. Also this modified version of *Copy* does not visit

any other cells than those used in the final configuration. (Give a transition diagram for this *Copy*.)
The module *Delete* is given in Example 7.20 (Figure 7.22, with some errors); $(p, y\underline{a}z) \vdash^* (q, y\underline{z})$ where z doesn't contain blanks and is followed by a blank, a may be a blank, and p and q are the initial and final state. This Turing machine only visits the cells occupied by $az\Delta$, the last part of its input.
Our TM works as follows:
If the TM is given 0 as input, it can stop immediately (after changing \$ back into Δ), because $0^2 = 0$.
If the input is not empty, the first 1 is marked (as a) and the TM looks for a second 1. If there is none it changes the a back into 1 and the \$ into Δ and it stops, because $1^2 = 1$.
If the TM discovers a second 1, then it proceeds as follows:
the 1 is marked (again as a), the head moves back to \$ and *Copy* copies the current string. The head moves to the right to the beginning of the fresh copy and looks for an unmarked 1. If there is one, it is marked, the head moves back to the beginning of this block and it is copied. This procedure is repeated until there is no 1 left in the last copied block, after which the head moves to the left, on the way deleting the Δ's and changing the a's into 1's until \$ is reached. This is changed into Δ and the machine stops.
The transition diagram is given below.



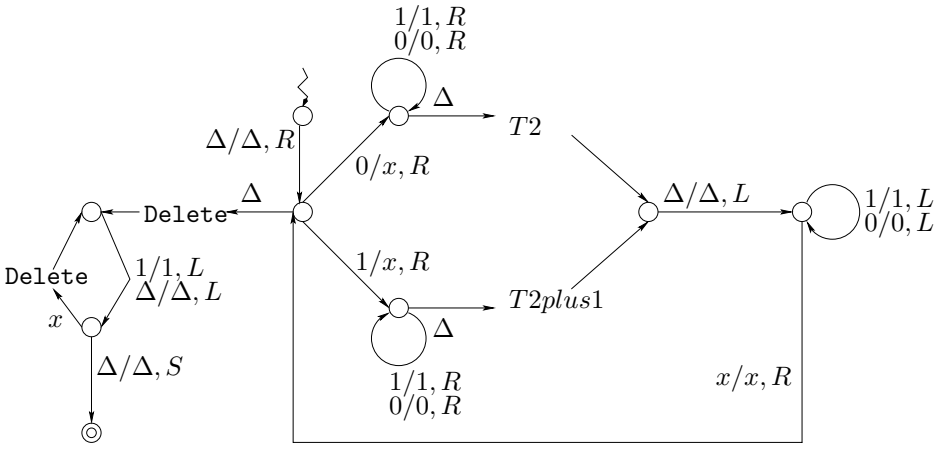
d Modify (and take care of a proper final configuration) the TM in Figure 7.37 (see Exercise 7.9).

7.18 Note that in Figure 7.38, the big arrow labelled $\Delta/\Delta, R$ has two

arrowheads; it should however point only to the left.
$f(x) = b^{n_b(x)}a^{n_a(x)}$, i.e., $f(x)$ has the same symbols as x, but with all the b's at the beginning.

7.19 We have two Turing machines T_1 and T_2, computing the functions f_1 and f_2, respectively. Our composite Turing machine first computes $f_1(x)$ and $f_2(x)$ using T_1 and T_2 given some input x. After that, it adds the results:
Copy the input x and insert a marker \$: $(q_0, \underline{\Delta}x) \vdash^* (q_{01}, \Delta x\$\underline{\Delta}x)$;
Compute f_1 using T_1: $(q_{01}, \Delta x\$\underline{\Delta}x) \vdash^* (h_1, \Delta x\$\underline{\Delta}f_1(x))$;
Interchange x and $f_1(x)$: $(h, \Delta x\$\underline{\Delta}f_1(x)) \vdash^* (q_{02}, \Delta f_1(x)\$\underline{\Delta}x)$;
Compute f_2 using T_2: $(q_{02}, \Delta f_1(x)\$\underline{\Delta}x) \vdash^* (h_2, \Delta f_1(x)\$\underline{\Delta}f_2(x))$;
Delete the marker, and go to the beginning of the tape:
$(h_2, \Delta f_1(x)\$\underline{\Delta}f_2(x)) \vdash^* (q_+, \underline{\Delta}f_1(x)\Delta f_2(x))$;
Add $f_1(x)$ and $f_2(x)$: $(q_+, \underline{\Delta}f_1(x)\Delta f_2(x)) \vdash^* (h_a, \underline{\Delta}f_1(x) + f_2(x))$.
Thus, apart from T_1 and T_2 we need TM modules to copy (see Example 7.18), to insert (Exercise 7.14), to swap two strings (design your own TM module for this task), to delete (Example 7.20) and to add two natural numbers. Note that if the two numbers are in unary representation, then they can be added by simply deleting the blank symbol separating them.

7.21 Let us assume that we are given two Turing machines $T2$ and $T2plus1$: $T2$ computes the function $f_2(1^k) = 1^{2k}$, for all $k \geq 0$ (in other words: $f_2(k) = 2k$), and
$T2plus1$ computes the function $f_{2+1}(1^k) = 1^{2k+1}$, for all $k \geq 0$ (in other words: $f_{2+1}(k) = 2k + 1$). We can now construct a TM which when given an arbitrary binary string (which may start with leading 0's) computes in unary the number it represents:
Going from left to right through the input string we compute after the blank following the input (starting from $0 = \Delta$) in unary 2 times the current value for each 0 and 2 times the current value +1 for each 1 we encounter. Once the whole input has been processed (and marked) in this way, we delete the Δ between marked input and result and then moving to the left we delete all x's and stop with the head on the blank in cell 0.
See the transition diagram.



Exercise: trace the computation of the above Turing machine in case the input consists of only 0's.

7.22 We assume that we already have a Turing machine T_{+1} which when given any natural number n in binary representation $\mathtt{bin}(n)$ (which is either 0 or starts with a 1) computes the binary representation of $n+1$:
$(q_0, \underline{\Delta}0) \vdash^* (h_a, \underline{\Delta}1)$ and $(q_0, \underline{\Delta}\mathtt{bin}(n)) \vdash^* (h_a, \underline{\Delta}\mathtt{bin}(n+1))$ for all $n \geq 1$.
A Turing machine that converts unary into binary could now work as follows:
Input 1^k;
if $k = 0$ (empty tape), write 0 (in cell 1), move to cell 0 and stop;
otherwise (there is at least one 1), move to the end of the input string, change the last 1 into a marker x, move two cells to the right, write a 1 and go back to the right end of the remaining input;
repeat, until no input 1 remains, for each rightmost 1 in the remaining input: replace 1 by x, move to the right until the first Δ, apply T_{+1}, go back to the right end of the remaining input;
if all input has been thus dealt with, change the tape contents from $\Delta x^k \Delta \mathtt{bin}(k)$ into $\Delta\mathtt{bin}(k)$ and stop with the head on cell 0.
Draw the transition diagram.

7.24 In Example 7.5, a TM has been given for the language $L = \{ss \mid s \in \{a,b\}^*\}$. We now sketch a TM that accepts L, this time without using any additional tape symbols ($\Gamma = \Sigma$):
Given is an input word $x \in \{a,b\}^*$.
If $x = \Lambda$ (the tape is empty) we can accept immediately. Thus from here we assume that $x = c_1 \ldots c_k$ with $c_i \in \{a,b\}$ and $k \geq 1$.
First it is checked that x is of even length:
insert an extra blank in cell 1: $(p, \underline{\Delta}x) \vdash^* (q, \Delta\underline{\Delta}x)$;
move c_1 one cell to the left and c_k one cell to the right; repeat this procedure for $c_2 \ldots c_{k-1}$ until either it turns out that k is odd (no symbol found to move to the right) in which case x is rejected; or we end with tape contents $\Delta c_1 \ldots c_{k/2} \Delta\Delta c_{(k/2)+1} \ldots c_k$.
Next compare the two halves of x symbol by symbol more or less as before, but rather than using uppercase letters as in Example 9.3 we replace symbols that have been dealt with by Δ. Before each pass we have to check however whether or not the symbols to be compared are the last symbols. This to avoid that, in the next pass, the head falls off when we move to the left in search for a nonblank symbol.

7.26 *NB* and *PB* are the TMs from Example 7.17, which move the tape head to the next (or previous, respectively) blank on the tape.
G is the NTM described in Example 7.30 with $(q_0, \underline{\Delta}) \vdash^* (h_a, \underline{\Delta}w)$ where w

can be any string over Σ.
Copy is the TM from Example 7.18 (Figure 7.19) with $(q_0, \underline{\Delta}x) \vdash^* (h_a, \underline{\Delta}x\Delta x)$ for all words $x \in \{0,1\}^*$.
Equal is the TM from Example 7.24 with, for all $x, y \in \{0,1\}^*$, $(q_0, \underline{\Delta}x\Delta y) \vdash^* (h_a, v\underline{c}w)$ for some v, c, w if and only if $x = y$.
Delete is the TM from Example 7.20 (Figure 7.22, with some errors) with $(q_0, y\underline{a}z) \vdash^* (h_a, y\underline{z})$ where z doesn't contain blanks and a may be a blank.
The NTM in the exercise, when given an input word $u \in \{0,1\}^*$, first goes to the right end of the input (using *NB*), then applies the NTM G starting from the first blank right from the input. Thus (nondeterministically) the tape contents are changed into $\Delta u\Delta w\Delta \ldots$ with $w \in \{0,1\}^*$ an arbitrary word; the head is on the Δ inbetween u and w.
Next w is copied (using *Copy*) and the Δ inbetween the copies is deleted (using *NB* and *Delete*).
The head moves to cell 0 (using *PB* twice) and with *Equal* it is tested whether $u = ww$.
Consequently, the language accepted by this NTM is $\{ww \mid w \in \{0,1\}^*\}$.

7.27 Hints to construct an NTM for $\{1^n \mid n = k^2 \text{ for some } k \geq 0\}$:
Use the NTM G described in Example 7.30, the TM *Square* from exercise 7.17(c), and the TM *Equal* from Example 7.24.
As in Exercise 7.26, generate an arbitrary string from the language given, compare it with the input and accept if and only if they are equal. Thus: walk to the right end of the input word $x \in \{1\}^*$, generate with G an arbitrary string w, apply *Square* which crashes when it encounters a 0. If the NTM has not crashed, then $w = 1^k$ for some $k \geq 0$ and we now have on the tape $\Delta x\Delta 1^{k^2}$. Finally, x and 1^{k^2} are compared.

7.28 Let $L \subseteq \Sigma^*$ be a language accepted by a Turing machine T.
In Example 7.30, an NTM is described which accepts the set of all prefixes of elements of L. This NTM begins by moving past the input x, calls the NTM G which generates an arbitrary word w from Σ^* and concatenates this to x by deleting the Δ inbetween x and w. It then moves back to cell 0 and calls T, which will lead to acceptance if and only if $xw \in L$, thus if and only if x is a prefix of a word from L.
a As in Example 7.30, but now the word w generated by G has to be moved to the left of x: we need a module $(q_0, \Delta x\underline{\Delta}w) \vdash^* (h_a, \underline{\Delta}wx)$. Then call T.
b First as in Example 7.30, concatenate x with an arbitrary w leading to xw; then, as in **a**, concatenate an arbitrary v to the left of xw which yields vxw. Then call T.

7.32 We are asked to describe a TM that enumerates the palindromes over $\{0,1\}$ in canonical order: $x \prec y$ iff $|x| < |y|$ or $|x| = |y|$ and x comes alphabetically before y.
We will use the fact that a palindrome v preceeds a palindrome w if and only if the first half of v (including its middle symbol if it is of odd length) preceeds the first half of w.
First we describe a module TM which when given a palindrome as input produces the next (in the canonical order) palindrome as output:
$(q_0, \underline{\Delta} x_n) \vdash^* (h_a, \Delta x_n \underline{\Delta} x_{n+1})$.

— use the module *Copy* to copy the input: $(q_0, \underline{\Delta} x_n) \vdash^* (q_1, \underline{\Delta} x_n \Delta x_n)$

— examine the copy:

if x_n contains no 0's, then change the copy to $0^{|x_n|+1}$, which is a palindrome; otherwise find the middle of the copy and change from there — going from right to left through the string — all 1's into 0's, until a 0 is found which is changed into 1; then modify the second half accordingly (palindrome).
Thus,
if $x_n = y0y^r$ for some string y, the copy of x_n has been changed into $y1y^r$;
if $x_n = y01^j0y^r$ for some string y and $j \geq 0$, the copy of x_n has been changed into $y10^j1y^r$ (check that this is indeed x_{n+1}!);

— go back to the Δ just before the thus modified copy and stop.

Clearly we can now make a TM enumerating all palindromes in canonical order by changing h_a into q_0. Hence when started on the empty palindrome, the TM will never stop and only be busy creating on the tape a list of all palindromes in canonical order.

Note that we could also have given a module TM which replaces the input x_n itself, rather than a copy, by x_{n+1}, which then would have led to a TM which shows all palindromes one after the other (in time), rather than behind one another.
