Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems Group (NSG)

HS 2015

Prof. L. Vanbever & A. El-Hassany,
based on Prof. R. Wattenhoffer's material

Discrete Event Systems

Solution to Exercise Sheet 5

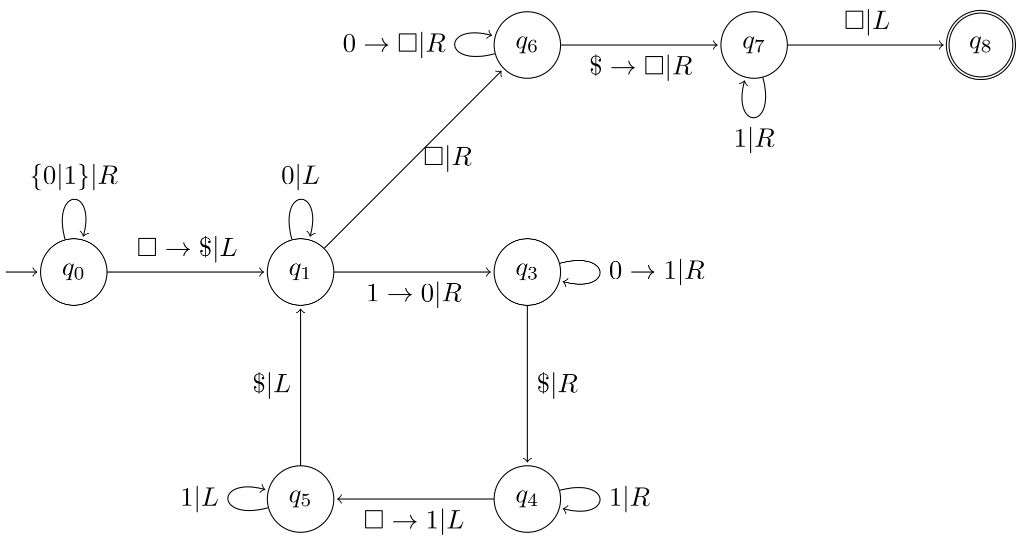
1 Designing Turing Machines

The proposed Turing machine decrements the value of a until $a = 0$. In each step, it adds a '1' to the output:

1. Move the TM head to the right of a and place a \$ sign. We will use this marker to return to the LSB of a.
2. Look at the LSB of a. If it is '1', we change it to 0 (transition between q_1 and q_3) and move to the right. Then, we continue moving to the right until we hit a $\square$, which is changed to a '1' (transition q_4 to q_5). Finally, we move back to the LSB of a.
3. If the LSB of a is '0', we search for the first '1' in a from the right (loop on q_1 and transition from q_1 to q_3).

3.1 If we find a '1', we change it to '0'. While moving back to the \$ symbol, we change all '0' to '1' (self-loop on q_3). Then, we proceed as in point 2 after passing the \$ symbol.

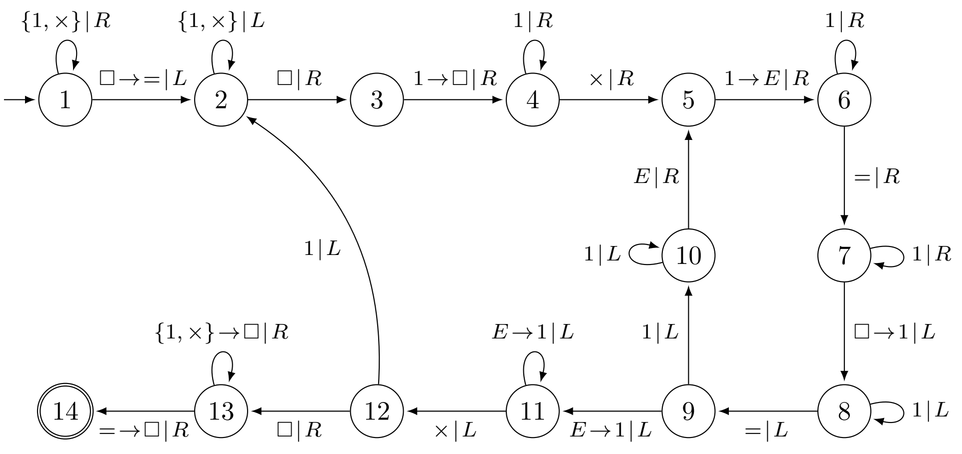
3.2 If we don't find a '1' in a at all (transition q_1 to q_6), we start the cleanup procedure: Remove all 0 on the right of the \$ symbol, and finally remove the \$ symbol itself and move to the right of u.



2 Turing again

a) (i) First we write a '=' (as a separator) to the right of the last 1 of b (state 1). Then we remove the leftmost 1 of a (states 2–4) and write for every 1 of b a 1 to the right of '=' (states 5–10) by replacing every already used 1 with 'E'. When there are only 'E's to the left of '=' (until '$\times$'), we replace them with '1's (states 11–12) and start again. This is repeated until there are no '1's to the left of '$\times$'. When this is the case, we have calculated the correct number to the right of '=' and we only need to remove all superfluous symbols (states 13–14).

(ii) A TM operated by the following DFA computes the desired output:



b) Obviously, every program that can run on a TM of type M_1 can also be run on a TM of type M_2, just ignore that additional tape exists. The other directions is a bit harder: The (limited) TM M_1 implements a mapping of a cell index x in M_2 to a cell index $f(x)$ in M_1 as follows:

$$f(x) = \begin{cases} 2x & x \geq 0 \\ 2|x| - 1 & x < 0 \end{cases} .$$

To implement this, the TM M_1 works in two modi A and B, defined as follows:

Modus A:

$$\begin{aligned} M_2 \to R \quad &\Rightarrow \quad M_1 \to 2R \\ M_2 \to L \quad &\Rightarrow \quad M_1 \to 2L \end{aligned}$$

Modus B:

$$\begin{aligned} M_2 \to R \quad &\Rightarrow \quad M_1 \to 2L \\ M_2 \to L \quad &\Rightarrow \quad M_1 \to 2R \end{aligned}$$

M_1 starts in modus A. If M_1 reaches the left end of the tape and cannot execute going to the left twice ($2L$), then M_1 goes one step to the right and switches to the other modus (in some sense, this is a similar construction as mapping the set of integers to the set of natural numbers).

3 An Unsolvable Problem

a) It is surprisingly easy to prove that your boss is demanding too much. Assume a function `halt(P: Program): boolean` which takes a program `P` as a parameter and returns a boolean value denoting whether `P` terminates or not.

Now consider the following program `X` which calls the `halt()` function with itself as an argument just to do the contrary:

```
function X() {
  if (halt(X))
    while(true);
  else
   return;
}
```

Obviously, if `halt(X)` is true `X` will loop forever, and vice versa.

b) If the simulation stops we can definitively decide that the program does not contain an endless loop. However, while the simulation is still running, we do not know whether it will finish in the next two seconds or run forever. Put differently: There is no upper bound on the execution time of the simulation after which we can be sure that the program contains an endless loop.

c) As we have seen, it is not possible to predict whether a general program terminates or not. However, under certain constraints we can solve the halting problem all the same. For example, consider a restricted language with only one form of a loop (no recursion etc.):

```
for (init; end; inc) {...}
```

where `init`, `end` and `inc` are constants in $\mathbb{Z}$. The loop starts with the value `init` and adds `inc` to `init` in every round until this sum exceeds `end` if `end` > 0 or until it falls below `end` if `end` < 0. Obviously, there is a simple way to decide whether a program written in this language terminates: For every loop, we check whether $\mathrm{sgn}($`inc`$) = \mathrm{sgn}($`end`$)$, where $\mathrm{sgn}(\cdot)$ is the algebraic sign. If not, the program contains an endless loop (unless `init` itself already fulfills the termination criterion which is also easy to verify).

4 How hard is it to add two numbers?

In our last course, slide 35 asked the following question:

Consider the language $L = \{x = y + z \mid x, y, z$ are binary bit-strings satisfying the equation$\}$. Proof that L is not context-free.

As usual, the proof works by contraction. That is, we first assume that L is context-free, meaning that the pumping lemma must hold for L. Let's p be the pumping length, and let's consider as string s: $1^p0^p = 1^p0^p + 0$. Clearly, s belongs to L and $|s| \geq p$. The pumping lemma guarantees that s can be divided into five pieces $s = uvxyz$ such that the following three conditions hold:

- $|vy| \geq 1$
- $|vxy| \leq p$
- $uv^ixy^iz \in L$ for all $i \geq 0$

Now, consider all the possible breakdowns of s:

a) vxy occurs to the left of "="

b) vxy occurs to the right of "="

c) vxy contains "="

In case **a)**, it is easy to see that s will not belong to the language anymore when pumped. Indeed, there will be more 1s on the left hand side of the "=", rendering the addition bogus. The same reasoning applies for **b)**. Again, because $|vxy| \leq p$, the pumping will start adding 1s, this time, on the right hand side of the equation. Again, rendering it incorrect.

This brings us to **c)**. Clearly, if v or y contains "=", then s won't belong to the language anymore as only one "=" can occur. This means that x must contain the "=" and $v = 0^k$ and $y = 1^j$, where k or j (or both) are non-zero. With such assignments, it is easy to see that the entire equation cannot be mathematically correct as we are pumping 0s on the left hand side and 1s on the right hand side.

We have shown that there is no way to divide s into $uvxyz$ so that the pumping lemma holds. This concludes the proof that L cannot be context-free.