Discrete Event Systems
Exercise Sheet 4

1 Designing Turing Machines

Alice is very happy because she was accepted for an internship at Tintel, one of the world's leading processor manufacturers. Unfortunately, she has only attended the famous DES lecture during her studies at ETH and knows nothing about electronic circuits. Therefore, she wants to solve her first assignment using a Turing Machine – please assist her:

Alice is asked to implement a *binary to unary converter*. This converter takes a number a in binary (alphabet $\{0,1\}$) and converts it to a unary number u (alphabet $\{1\}$). Initially, the TM head points to the MSB of a. At the end, the head should point to the right-most digit of u.

Provide a plain text description of your TM as well as a finite state machine controlling the tape head. Use the following notation for transitions:

'$\alpha \to \beta \mid \gamma$' read α from the tape at the current position, then write a β and finally move left if $\gamma = L$ or move right if $\gamma = R$.

'$\alpha \mid \gamma$' abbreviation for transitions of the form $\alpha \to \alpha \mid \gamma$ (these transitions do not modify the content of the tape).

Hint: The number n in unary representation consists of n ones. Also, you might want to extend the alphabet Γ to put temporary symbols on the tape.

2 An Unsolvable Problem

It's the first day of your internship at the software firm Bug Inc., and your boss calls you to his office in order to explain your task for the next three months. He says that many clients complain that the programs of Bug Inc. often contain faulty loops that never terminate. In order to prevent such errors in future, you are asked to implement a program that may check whether a given program will halt on all possible inputs or not.

a) Try to find a proof that convinces your boss that this is not possible for general programs.

Hint: The proof works by contradiction. Assume a procedure `halt(P:Program):boolean` that takes a program `P` and decides whether `P` halts on all possible inputs or not. Now construct a program `X` that terminates if `halt(X)` is false and loops endlessly if `halt(X)` is true, which yields the desired contradiction.

b) Your boss still disagrees and proposes the following method: `halt(Y)` simply simulates the execution of program `Y`. If the program terminates it returns true, and if it loops it returns false. Where is the problem of this approach?

c) Your boss is finally convinced but argues that your proof is a very special case that hardly reflects reality. Are there assumptions under which it is always possible to check whether a program halts or not?