

CS 159: Homework 1  
Professor John Savage  
**Assigned:** February 1, 2008,  
**Due:** February 8, 2008

## Question 1

In class (Lecture 2, slide 10) we stated that DFSMs and NFSMs recognize the same sets of languages. Given a NFSM with states  $q_i \in Q$ , it is possible to construct a DFSM with states  $q'_i \in 2^Q$ . A full explanation of this can be found in Chapter 3 of *Models of Computation*.

Any language recognized by an FSM can be represented as a “regular expression”. Given an alphabet,  $\mathcal{A}$ , a regular expression  $R$  is either a symbol in  $\mathcal{A}$ , the concatenation of two regular expressions (denoted  $R = R_1R_2$ ), the union of two regular expressions (denoted  $R = R_1 + R_2$ ), or the closure of a regular expression (denoted  $R = R_1^*$ ). The meaning of these operations should be clear from the following examples, in which  $\mathcal{A} = \{0, 1\}$ .

- $R = 0 + 10$  : The language consisting of the binary strings “0”, or “10”.
  - $R = (0 + 1)^*10$  : The language consisting of all binary strings ending in “10”.
  - $R = 1(00 + 11)^*1 + 000$  : The language consisting of “000” or a binary string that begins and ends with a 1, and otherwise contains only pairs of 0’s and 1’s.
1. Write an NFSM that recognizes each of the above examples. Please describe your FSM using a state transition diagram and be sure to shade in any accept states. When you are done, briefly describe a systematic approach for converting a regular expressions to NFSMs.
  2. Regular expressions are used in many programming languages, but they are often described using concatenation, union and closure, along with additional operations. One common additional operation is complementation where  $R^c$  denotes the set of strings not described by  $R$ . Argue that complementation is convenient, but not necessary. Try to keep your argument simple by first noting the equivalence between FSMs and regular expressions.

## Question 2

In class, we noted that a single tape Turing machine is no more powerful than a multitape Turing machine. Describe how a two-tape Turing machine,  $M$ , can be simulated with a single-tape Turing machine,  $M'$ , over a larger alphabet. Now describe how  $M'$  can be converted to a single-tape Turing machine,  $M''$ , that operates over the alphabet  $\{0, 1\}$ .

Both of your constructions should be clear, but not overly detailed. You should definitely not provide a state transition diagram. In both cases, you should also use big-O notation to bound the time and space overhead associated with your constructions. In other words, given the time,  $T$ , and space,  $S$ , required by  $M$ , bound the time and space required by  $M'$  and  $M''$ .

### Question 3

In class we mentioned the halting problem. Given a description  $\lfloor M \rfloor$  of a Turing machine  $M$  and an input to the Turing machine  $x$ , we say  $\langle \lfloor M \rfloor, x \rangle$  is in the language  $HALT$  if  $M$  halts on input  $x$ .  $HALT$  is undecidable, meaning no Turing machine exists that accepts only the strings in  $HALT$ .

A Turing machine is said to “require  $k$  tape cells” if, when started on a blank tape, its head eventually reaches the  $k^{th}$  cell. The busy beaver problem is defined as follows:

- Given an integer  $n$ , output an integer  $k$ , in binary, such that there is no Turing machine with  $n$  or fewer states that both halts when started on a blank tape and requires at least  $k$  tape cells.

Use the fact that  $HALT$  is undecidable to prove that no TM can solve the busy beaver problem for all  $n$ .

### Question 4 (OPTIONAL)

In the first lecture, we reviewed a number of models of computation. One was the circuit in which logic gates form the nodes of an acyclic graph and wires interconnecting gates form the edges. In this model, the “size” is the number of gates.

A second model was VLSI circuits. As before, a circuit is specified by an acyclic graph, but now both gates and wires occupy area. In this model, it is no longer sufficient to specify the edges and vertices of a graph, the graph must also be embedded in a plane, that is, a layout must be given. To do this, one can consider a grid of equal sized cells. Each cell can contain only one edge (i.e. wire) or one vertex (i.e. gate). In the VLSI model, the size of a circuit is the number of cells that it occupies. Unlike the traditional circuit model, the wires (which tend to span many cells) may account for most of the circuit area.

Suppose you are given an arbitrary circuit comprised of  $C$  gates. Can you give an upper bound to the number of cells required in the VLSI model to embed this circuit in the plane? Your bound does not need to be tight, but it should be polynomial in  $C$ . For simplicity, you can assume that the inputs and outputs to a gate can come from any direction.