# CSC 363, Winter 2010 — Short Assignment #2

*Due at **start** of tutorial time on January 27. Late assignments will **not** be accepted, as the tutorial instructors will immediately go over the solution. I prefer that you hand the assignment in on paper, but if you need to, you can email it (as a plain text file or PDF attachment, no Microsoft Word files please) to `radford@cdf.utoronto.ca`. (Please use this email address only for assignment submission.)*

*This assignment is to be done by each student individually. You are encouraged to discuss the course material in general with other students, but you should not discuss this assignment (verbally, in writing, by email, or in any other way) with people other than the course instructor and tutors, except to clarify the meaning of the question. Handing in work that is not your own is a serious academic offense.*

There is a Turing Machine that decides the language

$$A_{DFA01} = \{\langle B, w\rangle \mid B \text{ is a DFA with alphabet } \{0,1\} \text{ that accepts input string } w\}$$

A proof of this is briefly outlined on page 166–167 of the text (Theorem 4.1), for the more general case where the DFA could have any alphabet. For this assignment, you are to provide more details of the proof that $A_{DFA01}$ is Turing decidable, by giving an implementation-level description of a TM that decides $A_{DFA01}$. You should not give a formal description (such as a detailed state diagram). (See page 157 for more on the distinction between formal and implementation-level descriptions.)

The definition of $A_{DFA01}$ given above is not precise. A precise definition requires that we specify a way of encoding a DFA with alphabet $\{0,1\}$ so that it can be input to the TM. We can choose whatever input alphabet we wish to facilitate such an encoding. But note that whatever input alphabet we choose is *fixed*, the same for any DFA that the TM will simulate.

You should use the following encoding, which uses the input alphabet $\{0, 1, A, R, \$\}$. We order the states of the DFA to be encoded, and number them as $0, 1, 2, \ldots, k-1$, with 0 being the start state. The encoding of the DFA is $\$\delta_0\delta_1 \cdots \delta_{k-1}\$$, where $\delta_i$ is a string specifying whether or not $i$ is an accept state and the transitions from state $i$ when the input symbol is 0 or 1. We encode $\delta_i$ as $Xt_0 0 t_1$, where $X$ is either the symbol $A$, if $i$ is an accept state, or the symbol $R$, if $i$ is not an accept state, and $t_0$ and $t_1$ are the numbers of the states that follow state $i$ if the input symbol is a 0 or a 1. We encode the number $j$ as $j$ consecutive 1 symbols (so zero is encoded as the empty string).

For example, consider the DFA with $k = 3$ states, numbered $0, 1, 2$, in which 0 is the start state, 2 is the only accept state, and

$$\begin{array}{lll} \delta(0,0) = 0 & \delta(1,0) = 0 & \delta(2,0) = 2 \\ \delta(0,1) = 1 & \delta(1,1) = 2 & \delta(2,1) = 2 \end{array}$$

The encoding of this DFA is `$R01R011A11011$`. The encoding $\langle B, w\rangle$ for a DFA plus input string is just the encoding of the DFA followed by the input string.

Here, I will give a high-level description of a TM for deciding $A_{DFA01}$ with this encoding. You should produce an implementation-level description. For convenience, I will use a 3-tape TM, which as we've seen could be converted to a one-tape TM. The input string will be on tape 1 when the TM starts.

The TM starts by scanning the input, checking that it starts with a valid encoding of a DFA. This is mostly simple, except for checking that all the transitions specify valid state numbers (ie, if there are $k$ states, the transitions should be to states numbered 0 to $k-1$). To check that the state numbers are valid, the TM copies each state number (a string of some number of 1s) to tape 2, starting at the beginning of that tape, but *without* clearing tape 2 to blanks to start. The result is that when the end of the description of the DFA is reached (as marked by the second \$), tape 2 will contain the largest state number encountered. The TM can then check whether this number is too large for the the number of states in the description, and reject if it is too large.

Once the validity of the DFA encoding has been checked, the TM scans the input string after the DFA encoding, and copies it to tape 2.

The TM then simulates the transitions of the DFA. The number of the current state of the DFA is stored on tape 3, which is initialized to all blanks, which corresponds to state zero. To simulate a transition, the TM scans tape 3 from the beginning, and the encoding of the DFA (still on tape 1) from the beginning, moving past 1 symbols on tape 3 and past encodings of transitions from states on tape 1, until the right part of the encoding of the DFA is reached. Depending on whether the current input symbol (on tape 2) is 0 or 1, the appropriate new state number is then copied to tape 3, and the head for tape 2 is moved right to look at the next symbol. When the end of the input string on tape 2 is reached, the TM accepts or rejects depending on whether the current state is an accept state or reject state.

In producing an implementation-level description of such a TM, you can of course choose any tape alphabet you wish (but it must be *fixed*, the same for any input), and any set of states. You needn't specify exactly what the set of states is — you can just mention operations the TM does that will require various states in order to remember things (but keep in mind that you can remember only a fixed number of things in the state of the control unit). Your implementation-level description should be at a level of detail similar to what was given to you for short assignment #1, not at the level of detail you handed in as the solution for that assignment.