CSCC63 Assignment 1

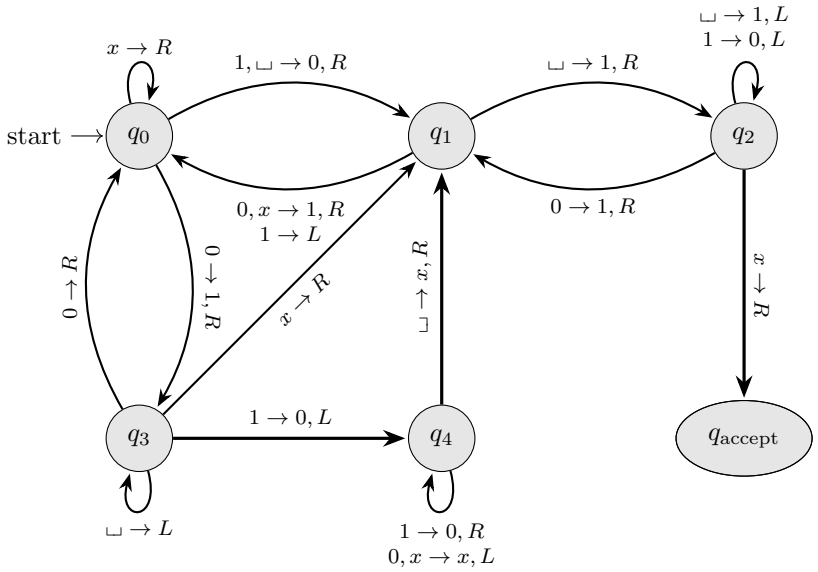
Turing Machines and Reductions

Due 11:59pm, June 07

Warning: *For this assignment you may work either alone or in pairs. Your electronic submission of a PDF to Crowdmark affirms that this assignment is your own work and that of your partner, and no one else's, and is also in accordance with the University of Toronto Code of Behaviour on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism in CSCC63. Note that using Google or any other online resource is considered plagiarism.*

1. (20 marks)

   (a) (15 marks) Let M_2 be the following TM. Suppose that M_2 is given the empty string as an input, so that the first configuration given this input is q_0. Write the first fifteen configurations that M_2 will take when you run it (this includes the initial configuration).



   (b) (5 marks) What is the is the largest number of characters by which any two neighbouring configurations C_i and C_{i+1} can differ for this TM? Why?

2. (10 marks) Consider the language L_2:

$$L_2 = \{\langle M \rangle \mid \exists n \in \mathbb{N}, 0^n 1^n \notin L(M)\}.$$

   Show that L_2 is not recognizable.

3. (10 marks) Consider the language L_3:

$$L_3 = \{\langle M \rangle \mid \exists n \in \mathbb{N}, 0^n 1^n \in L(M)\}.$$

   Show that L_3 is not co-recognizable.

4. (20 marks) Consider the language L_4:

$$L_4 = \{\langle M, k \rangle \mid \exists w \in \Sigma^*, \langle M, w, k \rangle \in \text{ACC}\}.$$

   Is L_4 recognizable, co-recognizable, neither, or decidable? Prove your answer.

5. (20 marks) Consider the language L_5:

$$L_5 = \{\langle M \rangle \mid \overline{L(M)} \subseteq \{0^n 1^n \mid n \in \mathbb{N}\}\}.$$

Is L_5 recognizable, co-recognizable, neither, or decidable? Prove your answer.

6. (10 marks) Suppose that you come across a new type of TM – a *plane TM*. A plane TM P is a lot like a regular TM in that it has a set Q of internal states, an input and tape alphabet Σ and Γ, a start state q_0, and designated *accept* and *reject* states q_A and q_R.

However, the tape of a plane TM P is a 4-way infinite two-dimensional plane: there is a cell for every x, y in $\mathbb{Z}^2$. Moreover, each of these cells contains a string in Γ^* instead of a single character. When we run P on n input $w \in \Sigma^*$, we set the string in cell $(0, 0)$ to w and every other cell to ε.

Consider any step of P in which we are in a state q_a and cell (i, j) of the tape.

- If P is in the accept state q_A it halts and accepts. If it is in the reject state q_R it halts and rejects.
- Otherwise, it reads the *final* character of the strings in the cells $(i-1, j-1), (i, j-1), (i+1, j-1), (i-1, j), (i, j), (i+1, j), (i-1, j+1, i, j+1)$, and $(i+1, j+1)$ (i.e., in the 3×3 square centered at (i, j)). The transition function δ of P can depend on all of these cells, as well as the state q_a.

  When reading the final character of the empty string ε, we treat it as a blank character $\sqcup$.
- The transition function can tell P to either write any charcter in Γ to the end of the string in (i, j), or to erase the final character in that string. If we try to erase the final character in an empty string we just get the empty string again.

  After writing or erasing a charcter, P can change its state to a new state $q_{a'}$ and move its tape head to a new cell in the 3×3 square centered at (i, j) (so it can move to any of the cells it has just read from).

  As usual, if there is no transition available for P to take it just goes into the reject state.

To answer this question, show how, given a plane TM P, you would write a regular TM M such that on any input w:

- If P accepts w, then so does M.
- If P rejects w, then so does M.
- If P loops on w, then so does M.

Your answer should include the pseudocode for the TM M and a justification (not a full proof) that your code is correct. If you write a multitape TM, describe what the different tapes are used for and how the data on those tapes will be formatted.

7. (10 marks) **Note: for this question you'll need to have covered oracles and the Arithmetic Hierarchy. If the notes for this aren't up they'll be up soon.**

In class we've said that problems like the halting problem are not solvable using algorithms, where algorithms are programs that must halt and return the correct answer in a finite number of steps. But what happens if we allow TMs to loop?

First of all, how can a TM even return an answer if it loops?

Let's consider the following modification of TM: A **looping** TM M is like a regulr TM, but instead of the usual *accept* and *reject* states it has a write-only output tape, a *true* state q_T and a *false* state q_F. The states q_T and q_F are not halting states – they have outgoing arrows and act like every other state in M, except:

- Whenever M enters the q_T state it prints a T to the output state and moves the output head one step to the right.
- Whenever M enters the q_F state it prints a F to the output state.

If M reaches a point where there is no arrow to follow it goes into q_F as usual, but as we have said it doesn't halt.

> *That is, M periodically outputs its best current guess as to whether its input is accepted or not. Since it does not halt, it can change its answer.*

As before, M will take finite strings w as inputs. We'll say that M *converges to a true* on input w iff, when run on w, there is some time n_0 in which it prints a T to its output, and for every $n > n_0$ it does not print a F to its output. Similarly, M *converges to a false* on input w iff, when run on w, there is some time n_0 in which it prints a F to its output, and for every $n > n_0$ it does not print a T to its output. In either case we say it *converges*. If M does not converge on w then we say that it *diverges*.

We'll say that M accepts a string w if it converges to a true on w, and we'll set

$$L(M) = \{\langle w \rangle \mid M \text{ accepts } w\}.$$

> *We'll say that M recognies L if $L = L(M)$, and that M decides L if it recognies it and converges for all inputs.*

The important distinction of this model is that if M decides L then it will always converge to the right answer, but we'll in general never know when. We lose the ability to ever be sure that we know the answers to our questions. You can see why we don't use this model to describe computing – when we write programs we generally want to be certain when we've gotten to our answers. But there are situtions in which a guarantee of convergence is the best we can do, so it's worth looking at this model.

It turns out the looping TMs are strongly equivalent to regular OTMs that use oracles for the language HALT: If you give me a looping TM M then I can give you an OTM N^{HALT} such that for all inputs w:

- If M accepts w then so does N^{HALT}.
- If M converges to a false on w then N^{HALT} rejects.
- If M diverges on w then N^{HALT} loops.

The converse also holds. So the languages decidable by looping TMs are eactly the class Δ_2 and the languages recognizable by looping TMs are exactly Σ_2 (a similar statement can be made about co-recognizability and Π_2)

To answer this question, show how, given a looping TM M, you could build such an OTM N^{HALT}. Your answer should include the pseudocode for the OTM N^{HALT} and a justification (not a full proof) that your code is correct. You do not need to show me the states or the tape format for N^{HALT}.

8. **Bonus** (10 marks — your mark will be rounded to the nearest multiple of 2.5)

   Show the converse to the previous question: Given an OTM N^{HALT}, show me how you would write a looping TM M such that:

   - If N^{HALT} accepts w then so does M.
   - If N^{HALT} rejects w then M converges to a false on w.
   - If N^{HALT} loops on w then M diverges.

   Your answer should include the pseudocode for the looping TM M and a justification (not a full proof) that your code is correct. You do not need to show me the states or the tape format for M.