Simulating Things

Peter Dixon

March 1, 2021

1 Recap

A Turing machine is like a DFA with memory. It has

1. an input alphabet Σ (that doesn't contain $\sqcup$ or $\vdash$)
2. a tape alphabet Γ that contains Σ and two special characters $\sqcup$ and $\vdash$
3. states Q, including 3 special states Start s, Accept t, and Reject r
4. a transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

The memory is a "tape" or "cells". Each cell can store one symbol in Γ. The tape has a left end marked with $\vdash$, and infinite space to the right. When the Turing machine is given a string $x \in \Sigma^*$ as input, the tape will start out with $\vdash$ in the left end, then the symbols in x, then an infinite number of $\sqcup$. The transition function checks the current state and tape cell, then writes something in the current cell, changes state, and moves left or right on the tape.

2 Equivalent Models of Computation

The main idea we're going to cover today is converting between models of computation. This is how we compare the power levels of two models of computation. If a Turing machine can simulate a DFA, then anything you could do with a DFA can also be done with a Turing machine.

2.1 Example 1

We'll (formally) describe how to convert a DFA into a Turing machine.

Take a DFA $A = \langle Q, \Sigma, \delta, s, F \rangle$. Construct a TM $M = \langle Q', \Sigma, \Gamma, s, t, r, \delta' \rangle$ where

- $Q' = Q \cup \{t, r\}$
- $\Gamma = \Sigma \cup \{\sqcup, \vdash\}$

- $\delta'(q,x) = \begin{cases} \delta(q,x), x, R & x \in \Sigma \\ s, \vdash, R & x = \vdash \\ t, \sqcup, R & x = \sqcup, q \in F \\ r, \sqcup, R & x = \sqcup, q \notin F \end{cases}$

This shows that Turing machines are at least as powerful as DFAs. (If you really want to be complete, you could formally prove that this TM accepts a string w if and only if the DFA accepts w.)

What's the real goal of simulation? It's two things:

1. If we want to show that Turing machines CAN do something, it's useful to have a very powerful model that's still simulatable by a Turing machine. (We only have to do the hard work of simulating the powerful model once.)

2. If we want to show that Turing machines CAN'T do something, it's useful to have a very restricted model that's still strong enough to simulate a Turing machine. (Again, we only have to do the hard work of simulating once.)

Analogy: once you can compile Java to assembly, you don't have to program assembly any more.

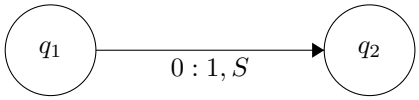
2.2 Example 2

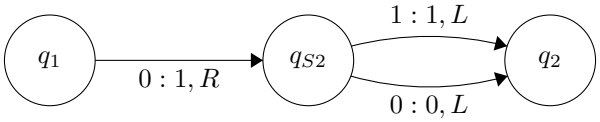
Now we're going to show that you don't change anything by making a Turing machine with a Stay option. (It can go left, right, or stay in the current cell.) We'll call it a STM.

Obviously an STM can simulate a regular TM. We just need to show that a TM can simulate a STM. We have to somehow replace every S transition without changing what the TM accepts.

How can we replace a S transition with L and R transitions? Any time we would stay, we go right and then left.

What does that look like?





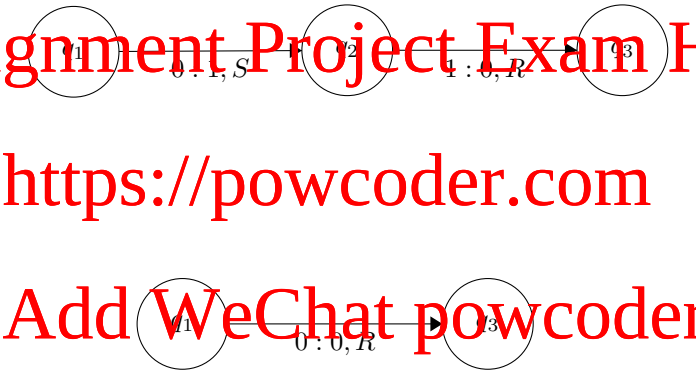
Last, how do we write it formally?

Take a STM $M = \langle Q, \Sigma, \Gamma, \delta, s, t, r \rangle$. We'll build a TM $M' = \langle Q', \Sigma, \Gamma, \delta', s, t, r \rangle$ where:

- $Q' = Q \cup \{s_1, \ldots s_{|Q|}\}$
- $\delta'(q,x) = \begin{cases} \delta(q,x) & \delta(q,x) \text{ is not a S transition} \\ (s_t, y, R) & \delta(q,x) = (t,y,S) \\ (t,x,L) & q = s_t \end{cases}$

The first line says "make a new state for each state in M". I'll leave the rest for you all to figure out.

There's actually another solution: If the Turing machine stays in the same cell, then the next symbol it reads is whatever it just wrote. So we can figure out what the Turing machine will do next, and combine the stay transition with that.



How would you write this formally?