Week 5 Notes (Turing Machines)

March 6, 2024

Contents

1 Lecture Portion **1**
- 1.1 Turing Machines . . . 1
  - 1.1.1 Turing's Model . . . 1
  - 1.1.2 Task . . . 2
  - 1.1.3 Register Machines . . . 2
- 1.2 Metalogic of Turing Machines . . . 2
  - 1.2.1 Finite Sets . . . 2
  - 1.2.2 Natural Numbers . . . 2
  - 1.2.3 Larger Infinities . . . 2
  - 1.2.4 Uncomputable Functions . . . 3

2 Tasks **3**
- 2.1 . . . 3
  - 2.1.1 [Task Description] . . . 3

1 Lecture Portion

1.1 Turing Machines

1.1.1 Turing's Model

It's actually equivalent to lambda calculus but it was published later. It was a simplified model based on the punch tape / punch card machines of the time.

A Turing maching has two components:

1. A finite state machine.
2. An unlimited tape (this is different from being infinite, it's more like an all-you-can-eat).

There exists one Turing Machine per possible program.

- Each cell can store the symbol S_1 or S_0.
- Only one location on the tape can be active at any given time.
- Every state transition depends on the active tape state and the FSM state.
- Every transition can:
  - Write to the tape at the active location.
  - Move the active location one to the left or right.
  - (Both? actually does it have to be both? Figure this out.) *

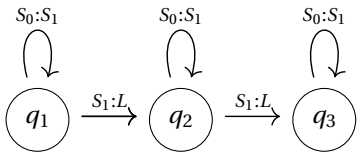
*I actually looked this up and there's a distinction between 4-tuple and 5-tuple Turing machines. The 4-tuple model seems to be the one used in the workshop slides and this model is one where you either perform a movement or a write, while the 5-tuple model is one where you perform both. They're pretty clearly mathematically equivalent though, since you can just use an intermediate state in the 4-tuple model to know to perform two actions.

– Change the state of the machine.

There are many ways of describing an FSM. You can use a table (1), a flowchart (2) or simply a list of quadruples (3).

	S_0	S_1
q_1	$S_1 q_1$	$L q_2$
q_2	$S_1 q_2$	$L q_3$
q_3	$S_1 q_3$	

(1)



(2)

$q_1 S_0 S_1 q_1, q_1 S_1 L q_2$
$q_2 S_0 S_1 q_2, q_2 S_1 L q_3$
$q_3 S_0 S_1 q_3$

(3)

If no action is given then we halt (this is relevant later when we discuss the halting problem.

Starting off: state = q_1 location = 1 tape has a finite number of ones on it (presumably finitely far apart?)

This can be modelled as logic (write out).

The rules are strictly deterministic, one state always implies at most one other state, so paralellisation is harder.

1.1.2 Task

Task: Use doubling program from slides (this isn't binary doubling, it's a tally doubling).

We start with 3.

$\overset{1}{1}11$

1.1.3 Register Machines

Every register can have any positive integer instead of just 1 or 0. You can add or remove one instead of setting to 0 or 1. You can prove this is equivalent to a turing machine by writing a virtual machine. A good intuition for this is the fact that you can say that numbers are sets of ones separated by single zeroes.

1.2 Metalogic of Turing Machines

1.2.1 Finite Sets

We can use ||S|| to mean the size of set S. Two sets have the same size if you can pair their elements 1:1. [fill in with example]

1.2.2 Natural Numbers

Let's define $\|N\| = \aleph_0$. We can pair this with evens, primes, fractions, and even programs. This is clear if you just think of code as a series of

1.2.3 Larger Infinities

Cantor showed in 1891 that the set of all functions has count > (cardinality?) aleph 0.

Begin by supposing that we have a way of pairing every function with a natural number. We can write all functions as a list of lists.

Say every function has an output of 1 or 0. (Not needed though, we can just use != instead of = !). (maybe i'm wrong, so go with binary after all, it's in the notes as such.)

We can make a new function with the negative along the long diagonal of the infinite matrix. Now we have a new function that matches none of the functions on the list.

Therefore by contradiction the functions aren't countable.

1.2.4 Uncomputable Functions

2 Tasks

2.1 Carry out the TM doubling program in full

Appendix

[Filename]