Turing Machines

Scribe: Connor Mooney

Date: 21 May 2019
Turing Machines are a fundamental model of computation postulated to be able to compute all computable problems.

1 Review

Definition. *An **alphabet** is just a set of "characters," or "symbols."*

Definition. *A **word** is just a string of characters from the alphabet.*

Definition. *A **language** $\mathscr{L}$ is a subset of the set of all words*

Example 1.1. *All $\{0,1\}^*$ strings with an even number of 0s form a language.*

Example 1.2. *All sentences in $[a-z]^*$ including "the" form a language.*

Example 1.3. *all a, b, c, n such that $a^n + b^n = c^n, n > 2$ form a language.*

2 Push Down Automata

Push Down Automata are a type of automata that also contain something called the "stack," which behaves like the stack we know from programming.
Consider the language $\mathscr{L} = ww^R$, or the palindromes in $\{0,1\}^*$.

- Read a character from input
- Push that character onto the "stack"
- Repeat OR go to the next step
- Read a character and pop off the stack
- Compare the next character to the one popped off the stack
- Repeat previous two steps
- If the stack is empty when we run out of input, we end in ACCEPT state

This specific instance of a Push-Down Automata model is **Nondeterministic**, which means that it somehow always takes the path to an "ACCEPT" state if there is one.

3 Turing Machines

Turing Machines are even more robust.

Definition. *A **Turing Machine** has access to:*

- *Input,*
- *state,*
- *an infinite "tape," which can be read or written to.*

And thus has the ability to perform these operations:

- *Read input OR tape*
- *Write to tape*
- *Move left OR right*

Example 3.1. *Add $x + y$ in unary, i.e. $7 + 3 = 10$ in unary is*

$$1111111 + 111 = 1111111111.$$

We model the input as

$$1111111\#111$$

and the code(notated with the state on the left before the colon and the state moved to as after the semicolon,)

$$0 : 1 \to write(1), right; 0$$

$$0 : \# \to nothing; 0$$

Example 3.2. *Multiplication in Unary*

We can take the input for multiplying 1111 and 111 to be $1111\#111$ Using this, we can get our process to be this:

- Copy input to tape
- Read the first 1 in the tape, and move right until the next blank
- Copy the 1, and then change the first 1 to a 0
- Back up to the 0
- Repeat copying the 1 and pasting and then converting to a 0 until we hit the next string of digits.

- Change that 1 to a 0
- Change all 0s before back to a 1
- Goto step 3
- End at the string "0#1"

Example 3.3. *Convert binary to unary*

$$\underline{001101} \rightarrow ?$$

Code:
$s0$: $0 \rightarrow$ nothing; $s0$,
$s0$: $1 \rightarrow$ write 1; $s1$,
$s1$: $0 \rightarrow$ <u>double</u> output string; $s1$,
$s1$: $1 \rightarrow$ <u>double</u> output string, write 1; $s1$
double x :
  write $x\#x$
  add, as we previously reviewed

Example 3.4. *Dividing binary strings (integer division, not floating.)*

$$1001101//110011$$

Procedure

- Convert to unary
- Subtract the a-b (overwrite ones with zeros)
- Stop when b¿a

Q: Does a multitape Turing Machine have more power than a single tape Turing Machine? A: We can interweave the contents of both tapes onto one tape, and then Record the position of each multitape's position on the tape, like so

$$A_{-1}B_{-1}A_0B_0A_1B_1A_2B_2\cdots\#P_A\#P_B.$$

Because a multitape is simulatable in a single tape Turing Machine, it is not more powerful

4 Non-determinism

$\mathscr{L}$ = <u>composite</u> numbers in binary We can determine membership either deterministically or non-deterministically

4.1 Deterministic Implementation

Simulate 4 Tapes: x, D, $S_1 = x\#D$, and $S_2 : x \mod D$ And we can use the code:
Increment D
  if $x \mod D \equiv 0$, ACCEPT
  Else D++
If D=x, Reject This is $p\mathcal{O}(divide)$, where p is the smallest prime factor of x. This is an exponential time algorithm in binary length of input.

4.2 Non-deterministic Implementation

Write D OR x,
  Accept if $D|x$ and $D < x$.
This is $\mathcal{O}(divide) = \mathcal{O}(n)$

4.3 the P-NP problem

The question is, then: Are non-deterministic Turing Machines necessarily faster than their deterministic counterparts, when