QUANTUM ALGORITHMS
HOMEWORK 1 SELECTED SOLUTIONS

PROF. MATTHEW MOORE

AP 1. Write a Turing machine that takes as input a binary number a (a string in "0" and "1") and outputs $a-1$ (in binary). Be careful to specify the alphabet and all instructions.

Solution: Let's say that the most significant bit of a is the leftmost bit. We will implement binary subtraction for *strictly* positive numbers.

Let $A = \{0,1\}$ be the external alphabet, $-$ the blank symbol, and $S = A \cup \{-\}$ the full alphabet. We will make use of 3 states, $Q = \{q_0, q_1, q_2\}$, with starting state q_0. The transition δ function is defined by

$$\begin{aligned} &\delta(q_0,0) = (q_0,0,1), && \delta(q_1,0) = (q_1,1,-1),\\ &\delta(q_0,1) = (q_0,1,1), && \delta(q_1,1) = (q_2,0,0),\\ &\delta(q_0,-) = (q_1,-,-1). && \end{aligned}$$

AP 2. Let $\mathcal{T} = \big\{\mathcal{M} \mid \mathcal{M} \text{ is a Turing machine}\big\}$.

Find a function $f : \mathcal{T} \to \mathbb{N}$ such that if $\mathcal{N}$ and $\mathcal{M}$ are Turing machines and $f(\mathcal{N}) = f(\mathcal{M})$ then $\mathcal{N} = \mathcal{M}$ (that is, $\mathcal{N}$ and $\mathcal{M}$ have the same alphabet, same transition function, etc.).

Solution: A Turing machine $\mathcal{M}$ is determined by its external alphabet A, the full alphabet S, the blank symbol $- \in S \setminus A$, the set of states Q, the initial state, and the transition function δ. Let

$$A = \{a_1, \dots, a_\alpha\}, \qquad S = \{s_1, \dots, s_\sigma\}, \qquad \text{blank} = s_b \in S \setminus A \text{ with } 1 \le b \le \sigma.$$

Enumerate the possible directions of movement $\{-1, 0, 1\}$ as $\{D_1, D_2, D_3\}$, and let's suppose that δ is defined on d-many values,

$$\delta(q_{v_j}, s_{w_j}) = (q_{x_j}, s_{y_j}, D_{z_j}) \qquad \text{for} \qquad 1 \le j \le d.$$

Furthermore, we will sort the list of δ's values lexicographically so that δ is uniquely determined by this list of d values. Let $p_1, p_2, \dots$ be the increasing sequence of distinct prime numbers. Define

$$f(\mathcal{M}) = p_1^\alpha \cdot p_2^\sigma \cdot p_3^b \cdot \prod_{j=1}^{d} \Big(p_{5j-1}^{v_j} \cdot p_{5j}^{w_j} \cdot p_{5j+1}^{x_j} \cdot p_{5j+2}^{y_j} \cdot p_{5j+3}^{z_j} \Big).$$

Since we have fixed the order of the primes and the order in which values of δ are considered, given a number of this form we will be able to (after factoring it into primes) fully reconstruct the machine $\mathcal{M}$. This implies that if $f(\mathcal{N}) = f(\mathcal{M})$ then $\mathcal{N} = \mathcal{M}$.

Date: January 30, 2020.

1.3. ("The halting problem is undecidable"). Prove that there is no algorithm that determines, for given Turing machine and input string, whether the machine terminates at that input or not.

Solution:

Proof. Let us suppose towards a contradiction that the halting problem is decidable. This means that there is a Turing machine which computers the function

$$h(x, y) = \begin{cases} 1 & \text{if } x \text{ halts on input } y, \\ 0 & \text{otherwise,} \end{cases}$$

where x is a description (encoding) of a Turing machine and y is an input string. If $h(x, y)$ is computable, then so is the function $f(x) = h(x, x)$. Let TURING be the algorithm

```
def TURING($x$):
  if $f(x) = 1$:
    enter an infinite loop, never halting
  else:
    halt
```

Since $f(x)$ is computable, certainly TURING is as well. Let $\mathcal{T}$ be the Turing machine which computer Turing, and let us consider whether $\mathcal{T}(\mathcal{T})$ halts.

If $\mathcal{T}(\mathcal{T})$ halts, then $f(\mathcal{T}) = 0$, so $h(\mathcal{T}, \mathcal{T}) = 0$. This implies that $\mathcal{T}(\mathcal{T})$ does not halt, a contradiction. Therefore it must be that $\mathcal{T}(\mathcal{T})$ does not halt. This implies that $f(\mathcal{T}) = 1$ and $h(\mathcal{T}, \mathcal{T}) = 1$. This is only possible if $\mathcal{T}(\mathcal{T})$ halts, another contradiction. Thus we have arrived at a contradiction in all cases. The only assumption at this point is that the halting problem was computable, so it must be that the halting problem is not computable. □

1.5. Let $T(n)$ be the maximum number of steps performed by a Turing machine with $\leq n$ states and $\leq n$ symbols before it terminates starting with the empty tape. Prove that the function $T(n)$ grows faster than any computable total function $b(n)$, i.e., $\lim_{n\to\infty} T(n)/b(n) = \infty$.

Solution:

Proof. Towards a contradiction, suppose that $\lim T(n)/b(n) = \lambda < \infty$. This implies that there is some N such that for all $n \geq N$ we have

$$\lambda - 1 \leq \frac{T(n)}{b(n)} \leq \lambda + 1,$$

so for all $n \geq N$ we have $(\lambda - 1)b(n) \leq T(n) \leq (\lambda + 1)b(n)$. A Turing machine with n states and symbols either halts in fewer than $T(n)$ steps or does not halt. Since $b(n)$ is computable, the above inequality gives us a procedure for solving the halting problem.

(1) Given a Turing machine $\mathcal{T}$ on n states with m symbols, we can either pad the number of symbols with extraneous ones or pad the number of states with extraneous ones to produce an equivalent machine with an equal number of states and symbols. Call the resulting machine $\mathcal{T}'$ and suppose that it has m symbols and states.

(2) Evaluate $b(m)$ (using the Turing machine that computes it) and multiply the value by $\lambda + 1$. Call the result γ.

(3) Run $\mathcal{T}'$ for γ steps. If $\mathcal{T}'$ has not halted in this time, then since $T(m) \leq \gamma$, it will never halt. □