1 RAM

A word-RAM consists of:

- A fixed set of instructions $P_1, \ldots, P_q$. Allowed instructions are:
  - Modular arithmetic and integer division on registers; the standard model for a RAM machine does not have negative numbers, but you can verify that allowing signed integers doesn't chance the power of the model.
  - Bitwise operations on registers.
  - Loading a constant into a register.
  - Transfering values between a register and a word in main memory (as addressed by another register).
  - Conditional GOTO.
  - MALLOC – increase the size S of main memory by 1. This also increments the word size if the current word size is too small to address all of main memory.
  - HALT.
- The problem counter $l \in \{1, \ldots, q\}$, telling us where we are in the program. Except for GOTO statements, the counter increments after each instruction is executed.
- The space usage S, which starts at the size of the input.
- The word size w; the initial word size is $\lceil \max(n+1, k+1) \rceil$, where n is the length of the input and k is the maximum constant appearing in the program or input.
- A constant number of registers $R[0], \ldots, R[r-1]$, each of which is a w-bit word.
- Main memory $M[0], \ldots, M[S-1]$, each of which is a w-bit word.

The tuple (l, S, w, R, M) gives the *configuration* of the word-RAM.

We say that a word-RAM *solves* a computational problem $f : \Sigma^* \to 2^{\mathbb{N}^*}$ if the machine which starts with input x halts with some output in $f(x)$ as the set of characters in $M[0], \ldots, M[R[0]-1]$.

As we saw with Turing Machines, there are many adjustments we can make to a model which make it easier to reason about without significantly changing its power. Here's one for word-RAMs:

Exercise. *We define a 2-D word-RAM model: Suppose that main memory, instead of being represented by a one-dimensional array of size S, is represented by a two-dimensional array of size $S \times S$. Consequently, saves or loads from main memory require addressing it using two registers; as before, MALLOC increases S by 1 (and consequently the size of memory by $2S+1$).*

Prove that a computational problem can be solved by a 2-D word-RAM model in polynomial time if and only if it can be solved by regular word-RAM model in polynomial time. In this problem, provide a ***formal description*** *of how to convert between the two models.*

2 Turing Machines

Formally, a Turing machines is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$, where

- Q is a *finite* set of states.
  - This means you can use your states to do *finite* counting (e.g. modulo 2), but not counting to arbitrarily large numbers.
  - Sometimes, instead of having a single q_{halt} state, people will talk about having a q_{reject} and q_{accept} state.
- Σ is the input alphabet.
- Γ is the tape alpabet, with $\Sigma \subset \Gamma$ and $\sqcup \in \Gamma - \Sigma$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
  - As we saw in class, we can simulate a multiple tape Turing machine with a single tape ; in this case, if we have k tapes, the transition function is $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$.
  - We'll often also allow an S for staying still; this does not change the power of the TM.
- $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$ are the start state, accept state, and reject state. respectively

A Turing machine configuration is a string $uqv \in \Gamma^* Q \Gamma^*$ that encodes (i) the state q of M, (ii) the tape contents uv ignoring trailing blanks, and (iii) the location of the head within the tape. The starting configuration is $q_0 x$.

We say that a Turing Machine *solves* a computational problem $f : \Sigma^* \to 2^{(\Gamma/\sqcup)^*}$ if the Turing machine which starts with input x halts with some output in $f(x)$ as the set of characters before the first $\sqcup$.

The **extended Church-Turing Thesis** says that every reasonable model of computation can be simulated on a Turing machine with only a polynomial slowdown.

Exercise. *Give the state diagram for a Turing machine which converts unary to binary (that is, given the input 1^x, it should return the binary representation of x).*

Often, when we talk about computational problems, we talk about decision problems, where the output is in $\{0, 1\}$; this is the same idea as deciding whether the given string is in some language $L \in \Sigma^*$.

However, in many cases the problem of finding a solution is not necessarily more "difficult" than the problem of determining whether there exists one. You're asked to prove a few of these on your homework; we'll look at one more of them here.

Exercise. *Show that there is a polynomial-time algorithm for* 3-SATISFIABILITY *(given a boolean formula in 3-CNF form, determine whether there exists a satisfying assignment) if and only if the computational problem of finding a satisfying assignment (or returning nothing if none exists) is in P.*

*For this problem, you can use **high-level** algorithm descriptions.*