Solutions to Homework 1

Turing Machines

From First Principles

Problem 1. For a string $\mathbf{x} \in \{0,1\}^*$, let $\mathsf{int}(\mathbf{x}) \in \mathbb{N}$ denote the integer whose binary representation is $\mathbf{x}$; so if $\mathbf{x} = x_n x_{n-1} \cdots x_1 x_0$, then

$$\mathsf{int}(\mathbf{x}) = x_0 + 2x_1 + 4x_2 + \cdots + 2^{n-1}x_{n-1} + 2^n x_n.$$

Let $f_{\mathsf{add}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be the "addition" function which maps the pair of strings $(\mathbf{x}, \mathbf{y})$ to the string $\mathbf{z}$ which satisfies $\mathsf{int}(\mathbf{z}) = \mathsf{int}(\mathbf{x}) + \mathsf{int}(\mathbf{y})$. Write down the transition function for a Turing Machine which computes f_{add}. Your TM should use the standard alphabet $\{0, 1, \triangleright, \square\}$; it should have two input tapes ($\mathbf{x}$ will be written on one, $\mathbf{y}$ on the other), in addition to the usual work and output tapes; it should have state set $\{\mathsf{begin}, \mathsf{running}, \mathsf{halt}\}$. Finally, the tapes of your TMs should have their starting cells on the right and continue infinitely off to the left (so the opposite of how we have been drawing them in class). This is for convenience since it will mean x_0 and y_0 are aligned and they are the first you encounter as you move left from the starting position.

Solution. The transition function is below. We leave off the case of begin state; in this case the TM does no writing, moves all tape heads left and changes the state to $\mathsf{running}$. Additionally, we leave off the case in the $\mathsf{running}$ state when the symbols read are $\big(\square, \square, 0/\square\big)$; in this case the TM does not writing or moving and enters the halt state. For all other cases, the TM will always remain in the $\mathsf{running}$ state, and will move both input tape heads and the output head left one. Therefore, in our transition function below we omit these instructions and simply include the writes. If $\mathsf{state} = \mathsf{running}$, do the following:

- $\big(0, 0, 0/\square\big) \Rightarrow (0, 0)$;
- $\big(1, 0, 0/\square\big) \Rightarrow (0, 1)$;
- $\big(0, 1, 0/\square\big) \Rightarrow (0, 1)$;
- $\big(1, 1, 0/\square\big) \Rightarrow (1, 0)$;
- $\big(0, 0, 1\big) \Rightarrow (0, 1)$;
- $\big(1, 0, 1\big) \Rightarrow (1, 0)$;
- $\big(0, 1, 1\big) \Rightarrow (1, 0)$;
- $\big(1, 1, 1\big) \Rightarrow (1, 1)$.

Problem 2. Describe a TM which computes the function $f : \{0,1\}^* \to \{0,1\}$ defined by $f(\mathbf{x}) = 1$ if the integer $\mathsf{int}(\mathbf{x})$ is divisible by 3; $f(\mathbf{x}) = 0$ otherwise. Your TM should use the standard alphabet and have the standard three tapes (input/work/output). As in Problem 1, you may assume the tapes of your TM start on the right and continue off infinitely to the left.

Solution. Let $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ be the binary string representing the integer $\mathsf{int}(\mathbf{x}) = x_0 + 2x_1 + 4x_2 + \cdots + 2^n x_n$. The key behind our algorithm is that $\mathsf{int}(\mathbf{x})$ is divisible by 3 if and only if the alternating sum $x_0 - x_1 + x_2 - x_3 + \cdots + (-1)^n x_n$ is divisible by 3. This follows immediately from the fact that $2 \equiv -1 \pmod 3$:

$$\mathsf{int}(\mathbf{x}) = \sum_{i=0}^{n} 2^i x_i \equiv \sum_{i=0}^{n} (-1)^i x_i \pmod 3.$$

The right hand side is precisely the alternating sum above.

Our Turing Machine, M, uses the standard alphabet $\Gamma = \{\rhd, \Box, 0, 1\}$ and uses one work tapes, which consists of just three cells (the first of which contains $\rhd$). We will give M the non-standard ability to read and write both work-tape cells in a single step. The first work tape cell will keep track of whether the current input bit x_i of $\mathbf{x}$ has i even or odd (0 for even; 1 for odd), the second work tape cell will keep the running total mod 3 (0 for 0, 1 for 1, $\Box$ for 2). The set of states is $\{\mathsf{begin}, \mathsf{running}, \mathsf{halt}\}$. As usual, begin does no writing, moves all tape heads left and changes state to $\mathsf{running}$. While in $\mathsf{running}$, if $(\Box, \Box)$ is read from the work tape cells, the tape heads remain in place, the work tape is set to $(0, 0)$, and state $\mathsf{running}$ remains (this occurs only on the first $\mathsf{running}$ step, just after the begin state). If $\Box$ is ever read from the input tape, then b is written to the output tape where $b = 1$ if the second work tape cell is 0, $b = 0$ otherwise, and the state is set to halt. All other options, keep the state $\mathsf{running}$ and move the input tape to the left, thus we only describe the values written to the work tape.

- $\big(b, (0, T)\big) \Rightarrow \big(1, T + b \pmod 3\big)$;
- $\big(b, (1, T)\big) \Rightarrow \big(0, T - b \pmod 3\big)$;

Bipartite Graphs

Definition (Graph and Adjacency Matrix). A *graph* G is a pair of finite sets $G = (V, E)$. We call V the set of *vertices* and $E \subset V \times V$ the set of *edges*. The *adjacency matrix* of G, denoted T_G is a $|V| \times |V|$ matrix with 0/1 entries where for any $u, v \in V$, the $(u, v)-$th entry is 1 if and only if $(u, v) \in E$. In this way, the graph $G = (V, E)$ can be represented by a string $T_G \in \{0, 1\}^{|V|^2}$.

Definition (Bipartite Graph). We say that a graph $G = (V, E)$ is *bipartite* if V can be partitioned into two sets A and B such that all edges in E connect a vertex in A with a vertex in B. Formally, G is bipartite if there exist $A, B \subsetneq V$ such that $A \cup B = V$ and $A \cap B = \emptyset$ such that for every $(u, v) \in E$ either $u \in A, v \in B$ or $u \in B, v \in A$. Let $f_{\mathsf{bipartite}} : \{0, 1\}^* \to \{0, 1\}$ be such that $f_{\mathsf{bipartite}}(\mathbf{x}) = 1$ if $\mathbf{x}$ is the adjacency matrix of a bipartite graph; $f_{\mathsf{bipartite}}(\mathbf{x}) = 0$ otherwise.

Problem 3. Describe part of a TM which computes $f_{\mathsf{bipartite}}$. For simplicity, let us assume your TM is always given input $\mathbf{x} \in \{0, 1\}^{n^2}$ for some integer n (so in particular, the input can be parsed as the adjacency matrix of a graph). Thus, your TM is not responsible for rejecting inputs which are not of this form (this would be trivial but tedious). Also, assume your TM has two 2-dimensional work tapes called the "neighbor tape" and the "partition tape". Assume that both 2d tapes have been populated as dictionaries with one key for every vertex in the graph;

the values in the neighborhood tape corresponding to the vertex v is a list of all vertices $w \in V$ which are *neighbors* of v (*i.e*, $\{w \in V : (v,w) \in E\}$). Write down a transition function which greedily iterates through the vertices of the graph and attempts to put each vertex into either A or B so that all edges traverse from A to B. Your TM can then output 1 if this process terminates successfully; it can output 0 if not.

Solution. Our TM M uses two additional (one-dimensional) work tapes called the "label A" and "label B" tapes. To enhance clarity, we describe the transition function for M using pseudocode; hopefully it is clear how to transfer each step to the precise Turing Machine language.

1. Write v_1 to the "label A" tape.
2. While the "label A" and "label B" tapes are not both empty, do the following:
   - assume the "label A" tape is not empty; if the "label A" tape is empty, then the "label B" tape is not empty, so do this step precisely except with the roles of A and B reversed;
   - read a vertex v_i from the "label A" tape;
   - go to the row in the partition tape corresponding to v_i;
   - if the value of this row is B, output 0 and halt; if the value is A, erase v_i from the "label A" tape and continue;
   - otherwise, the value is empty so write A as the value of v_i;
   - go to the row in the adjacency tape corresponding to v_i; write the entire value of this row to the "label B" tape and continue;
3. Output 1 and halt.

Non-Deterministic Turing Machines

Definition (Non-Deterministic Turing Machine). A *non-deterministic Turing Machine* is the same as a regular Turing Machine except that has two different transition functions δ_0 and δ_1. In particular, just like a regular Turing Machine, a non-deterministic Turing Machine has three tapes (input, work, output) and a register which keeps track of state. During each time step, the non-deterministic Turing Machine reads from the tape heads, checks the state and makes its decision on the new state, what to write, and where to move the heads according to one of its transition functions.

Definition (Non-Deterministic Decision). For a function $f : \{0,1\}^* \to \{0,1\}$ and a time function $T : \mathbb{N} \to \mathbb{N}$, we say that a non-deterministic Turing Machine M *computes f in time T* if for all $\mathbf{x} \in \{0,1\}^*$, there exists a sequence of bits $b_1, b_2, \ldots, b_{T(|\mathbf{x}|)}$ such that:

$$f(\mathbf{x}) = 1 \iff \mathsf{M}^{\mathbf{b}}(\mathbf{x}) = 1 \text{ in at most } T(|x|) \text{ steps},$$

where $\mathbf{b} = (b_1, b_2, \ldots, b_{T(|x|)})$ and the shorthand "$\mathsf{M}^{\mathbf{b}}(\mathbf{x}) = 1$" indicates that if M is initialized with $\mathbf{x}$ on its input tape and run so that in time step i, M uses the transition function δ_{b_i}, then M eventually halts with 1 on its output tape.

Problem 4. Let $f : \{0,1\}^* \to \{0,1\}$ be a function and $T : \mathbb{N} \to \mathbb{N}$ a time function. Prove that $f \in \mathsf{NTIME}\big(T(n)\big)$ if and only if there exists a non-deterministic Turing Machine M which computes f in time T.

Solution. Recall from class that $f \in \mathsf{NTIME}(T)$ iff there exists an $\mathcal{O}(T)-$time TM M such that for all $\mathbf{x} \in \{0,1\}^*$:

$$f(x) = 1 \Leftrightarrow \exists \mathbf{w} \in \{0,1\}^* \text{ s.t.} \mathsf{M}(\mathbf{x}, \mathbf{w}) = 1.$$

Let δ be the transition function for M. Then since M has two inputs, during each step of computation bits are read from both of M's input tapes and M's work tape and an action is taken according to δ. The non-deterministic TM computing f has only one input tape (because f takes only the one input $\mathbf{x}$), and the two transition functions δ_0 and δ_1 are both derived from δ as follows: δ_b performs the same computation that δ would have performed if it read b from the second input tape. Thus,

$$f(x) = 1 \Leftrightarrow \exists \mathbf{w} \in \{0,1\}^* \text{ s.t.} \mathsf{M}(\mathbf{x}, \mathbf{w}) = 1 \Leftrightarrow \exists \mathbf{b} \in \{0,1\}^* \text{s.t.} \mathsf{M}^{\mathbf{b}}(\mathbf{x}) = 1.$$

As M is an $\mathcal{O}(T)$-time TM, the non-deterministic computation halts in $\mathcal{O}(T)$ steps.

A Non-Deterministic Time Hierarchy Theorem

Problem 5. Prove the two claims below to complete the proof of the theorem via the technique of "lazy diagonalization".

Theorem. $\mathsf{NTIME}(n) \subsetneq \mathsf{NTIME}(n^2)$.

Notation. If M is a Turing Machine we let $\mathsf{str}(\mathsf{M}) \in \{0,1\}^*$ be the string representation of M. For a string $\alpha \in \{0,1\}^*$, M_α denotes the Turing Machine described by α, and $\mathsf{int}(\alpha) \in \mathbb{N}$ denotes the integer whose binary representation is α.

Proof. Define a function $\phi : \mathbb{N} \to \mathbb{N}$ recursively: $\phi(1) = 2$ and $\phi(i+1) = 2^{\phi(i)^{1.2}}$ for $i \geq 1$. Let us now define the function $f : \{0,1\}^* \to \{0,1\}$ as follows: $f(\mathsf{str}) = 0$ unless str can be parsed as a pair $(\alpha, \mathbf{x})$ such that $\mathbf{x} = 1^n$ (*i.e.*, the string of n consecutive 1s) for some $n \in \mathbb{N}$ such that $\phi(i) < n \leq \phi(i+1)$, where $i = \mathsf{int}(\alpha)$, and furthermore such that one of the following two points holds:

1. $n < \phi(i+1)$ and there exists $\mathbf{w} \in \{0,1\}^{n^{1.5}}$ such that $\mathsf{M}_\alpha(1^{n+1}, \mathbf{w}) = 1$ in at most $n^{1.5}$ steps;
2. $n = \phi(i+1)$ and for all $\mathbf{w} \in \{0,1\}^{(\phi(i)+1)^{1.1}}$, $\mathsf{M}_\alpha(1^{\phi(i)+1}, \mathbf{w}) \neq 1$ in $(\phi(i)+1)^{1.1}$ steps.

Claim. $f \in \mathsf{NTIME}(n^2)$.

Proof. We describe an $\mathcal{O}(n^2)-$time TM M such that $f(\alpha, \mathbf{x}) = 1$ iff $\exists \mathbf{w} \in \{0,1\}^*$ s.t. $\mathsf{M}(\alpha, \mathbf{x}, \mathbf{w}) = 1$. This proves $f \in \mathsf{NTIME}(n^2)$. M takes inputs $(\alpha, \mathbf{x})$ and $\mathbf{w} \in \{0,1\}^{|\mathbf{x}|^{1.5}}$ and computes as follows:

1. M checks that $i = \mathsf{int}(\alpha)$ and $\mathbf{x} = 1^n$ for some n such that $\phi(i) + 1 \leq n \leq \phi(i+1)$;
2. if $n \neq \phi(i+1)$ then M runs $\mathsf{M}_\alpha(1^{n+1}, \mathbf{w})$ for $n^{1.5}$ steps; if during this time M_α halts with output 1, M halts with output 1, otherwise M halts with output 0;
3. if $n = \phi(i+1)$ then M enumerates over all possible $\mathbf{w} \in \{0,1\}^{(\phi(i)+1)^{1.1}}$; for each $\mathbf{w}$, M runs $\mathsf{M}_\alpha(1^{f(i)+1}, \mathbf{w})$ for $\big(f(i)+1\big)^{1.1}$ steps; if M_α halts and outputs 1 for any $\mathbf{w}$, M halts with output 0; otherwise M halts with output 1.

Note that M computes f by construction, we bound the running time of M. The check in step 1 can be done efficiently using the observation:

$$\phi(i) < n \leq \phi(i+1) \iff \phi(i-1) < \log^{5/6} n \leq \phi(i).$$

Therefore, to check whether $\phi(i) < n \leq \phi(i+1)$ we can iteratively apply the function $g(n) = \log^{5/6} n$, $i-1$ times and check whether the result is between $2 = \phi(1)$ and $2^{2^{1.2}} \approx 5$. This can certainly be done in n^2 time. Step 2 requires running M_α for $n^{1.5}$ steps, and so takes less than n^2 time (it takes $n^{1.5}$ time times a logarithmic factor for bookkeeping). Finally, step 3 requires checking all $2^{(\phi(i)+1)^{1.1}}$ potential witnesses $\mathbf{w}$ and for each, running the Turing Machine M_α for $\big(\phi(i)+1\big)^{1.5}$ steps. Thus the total time for step 3 is roughly $2^{(\phi(i)+1)^{1.1}}\big(\phi(i)+1\big)^{1.5} < 2^{(\phi(i)+1)^{1.2}} = \phi(i+1) = n$. It follows that the time required to compute M is $\mathcal{O}(n^2)$, and so $f \in \mathsf{NTIME}(n^2)$. ☐

Claim. $f \notin \mathsf{NTIME}(n)$.

Proof. Suppose M is an $\mathcal{O}(n)-$time TM such that

$$f(\alpha, \mathbf{x}) = 1 \Leftrightarrow \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}(\alpha, \mathbf{x}, \mathbf{w}) = 1. \tag{1}$$

Let $\alpha = \mathsf{str}(\mathsf{M})$ be the string representation of M, let $i = \mathsf{int}(\alpha)$, and consider the family of unitary strings

$$\{1^n : \phi(i) < n \leq \phi(i+1)\}.$$

For $\phi(i) < n < \phi(i+1)$, we have

$$\begin{aligned} \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^n, \mathbf{w}) = 1 &\iff f(\alpha, 1^n) = 1 \\ &\iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^{n+1}, \mathbf{w}) = 1, \end{aligned}$$

where the first equivalence uses (1) and the second uses the definition of f. Note the difference between the first and the last statements is 1^n vs 1^{n+1}. Thus, by iterating this equivalence for all $\phi(i) < n < \phi(i+1)$, we get

$$\begin{aligned} \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^{\phi(i)+1}, \mathbf{w}) = 1 &\iff f(\alpha, 1^{\phi(i)+1}) = 1 \\ &\iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^{\phi(i)+2}, \mathbf{w}) = 1 \\ &\iff \cdots \\ &\iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^{\phi(i+1)}, \mathbf{w}) = 1 \\ &\iff f(\alpha, 1^{\phi(i+1)}) = 1. \end{aligned}$$

However, by definition of f,

$$f(\alpha, 1^{\phi(i+1)}) = 1 \iff \nexists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_\alpha(1^{\phi(i)+1}, \mathbf{w}) = 1.$$

Thus we have obtained our contradiction (and thus $f \notin \mathsf{NTIME}(n)$) since the first statement in the chain is equivalent to its opposite. ☐