The Church-Turing Thesis

Mahesh Viswanathan

The default Turing machine model we will consider in the first few weeks of this class is the *deterministic, one-tape* Turing machine model, and we will often refer to this as the "the Turing machine" model. This model is shown in Figure 1. It has a finite set of (control) states, a semi-infinite tape that has a leftmost

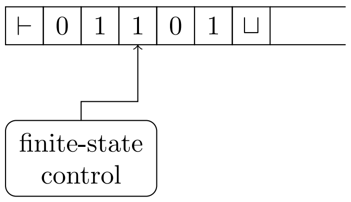

Figure 1: Single-tape Turing machine

cell containing a left end marker $\vdash$ (never written over) and extends to infinitely many cells to the right. This machine has a head that can move left or right one cell in each step, and in each step it reads the current symbol it is currently scanning and overwrites it before moving on. Initally, the tape contains the input string, which is finite, on cells numbered 1 onwards (cell 0 contains $\vdash$). The rest of the tape has a special symbol called the *blank* symbol $\sqcup$. Machine, initially, starts in state s with its input cell scanning the leftmost cell. In each step it changes its state based on its current state and symbol under its head, and it overwrites this cell, and moves its head left/right before taking the next step. The Turing machine is assumed to have two special *halting* states — *accept* state t and *reject* state r. Like the 2-way DFA, the Turing machine may never halt, if it does not reach either of these two halting states. The formal definition is as follows.

Definition 1. A *deterministic, one-tape Turing machine* is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,
- $s \in states$ is the *start state*,
- $t \in Q$ is the unique *accepting state*,
- $r \in Q$ $(r \neq t)$ is the unique *rejecting state*,
- $\delta : (Q \setminus \{t, r\}) \times \Gamma \to Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\}$ is the transition function that describes the next state, symbol to be written, and direction to move the tape head, given the current state and symbol read; it is assumed that no transitions are enabled from either t or r.

We assume that the left endmarker $\vdash$ is never over-written, i.e., for any $p \in Q \setminus \{t, r\}$, $\delta(p, \vdash) = (q, \vdash, \mathsf{R})$ for some $q \in Q$.

To describe computations of a Turing machine, we need to define the notion of a *configuration* which is a snapshot of all the relevant information about a Turing machine at a given step. For a Turing machine, this means the contents of the tape, the current state, and the current input head position.

At any given point, the contains of the machine's tape will be of the form $y\sqcup^\omega$, where $y \in \Gamma^*$ and $\sqcup^\omega$ is the infinite sequence $\sqcup\sqcup\sqcup\cdots$. In other words, all but finitely many symbols on the tape are $\sqcup$; this can be argued inductively as it is definitely true when the machine starts, and any step of the machine may increase the number of non-blank cells by at most 1.

A *configuration* α of TM M is an element of $Q \times \{y\sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}$. So a configuration $\alpha = (p, z, n)$ represents the fact that the TM is in state p, its tape contains z, and its head is scanning cell n. The *start configuration* of M on input $x \in \Sigma^*$ is $(s, \vdash x\sqcup^\omega, 0)$.

Just like 2-way DFAs, computations/runs are sequences of configurations related by the *next configuration relation*, $\xrightarrow[M]{1}$, which we define next. Before doing so, we introduce a convenient notation. For a tape $z = y\sqcup^\omega$ ($y \in \Gamma^*$), $s^n_b(z)$ is the string obtained from z by substituting b for the nth cell contents (z_n) in z. The relation $\xrightarrow[M]{1}$, is defined as

$$\delta(p, z_i) = (q, b, \mathsf{L}) \Rightarrow (p, z, i) \xrightarrow[M]{1} (q, s^i_b(z), i-1),$$
$$\delta(p, z_i) = (q, b, \mathsf{R}) \Rightarrow (p, z, i) \xrightarrow[M]{1} (q, s^i_b(z), i+1).$$

As for 2-way DFAs, we can define the n-fold composition of $\xrightarrow[M]{1}$, inductively, as follows.

$$\alpha \xrightarrow[M]{0} \alpha \text{ for every configuration } \alpha$$
$$\text{If } \alpha \xrightarrow[M]{n} \beta \text{ and } \beta \xrightarrow[M]{1} \gamma, \text{ then } \alpha \xrightarrow[M]{n+1} \gamma$$

Finally, the *reflexive, transitive closure* of $\xrightarrow[M]{1}$ is defined as, $\alpha \xrightarrow[M]{*} \beta$ if $\alpha \xrightarrow[M]{n} \beta$ for some $n \in \mathbb{N}$.

Definition 2. The machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ is said to *accept* input $x \in \Sigma^*$ if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (t, y, n)$ for some y and n. M is said to *reject* x if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (r, y, n)$ for some y and n. M *halts* on x if it either accepts or rejects x; a TM may not halt on some inputs.

The language of M is given by $\mathbf{L}(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.

The execution of a TM M on an input x can have one of three possibilities: it may accept x, it may reject x, or it may not halt on x. The language accepted/recognized by M only consists of inputs that M accepts. In other words, inputs on which M either rejects or does not halt do not belong to $\mathbf{L}(M)$. One important class of Turing machines are those that are *total.* A TM M is *total* if it halts on every input. That is, on any input, a total TM either accepts or rejects.

Definition 3. A language/decision problem L is

- *recursively enumerable* (RE) if $L = \mathbf{L}(M)$ for some TM M;
- *recursive* (REC) if $L = \mathbf{L}(M)$ for some *total* TM M; and
- *co-r.e.* (co-RE) if $\overline{L}$ is RE.

A number of different computational models are equivalent to the single tape Turing machine model that we have introduced. We give details of some of these variants and how they can be simulated on the Turing machine model.

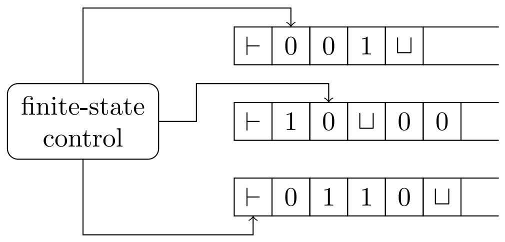

Figure 2: Multi-tape Turing machine

1 Multi-tape Turing Machine

A *Multi-tape Turing machine* is like the Turing machine model. However, the machine has multiple tapes on which it can write information and read from. Each tape has its own head that moves independently. Such a machine is schematically shown in Figure 2. Intuitively, this machine works as follows.

- Input is written out on Tape 1
- The leftmost cell of all tapes has a left endmarker $\vdash$ that is never overwritten.
- Initially all heads scanning cell 0 (leftmost cell), and tapes 2 to k are blank (except for the left endmarker).
- In one step, the machine read symbols under each of the k-heads, and depending on the current control state, writes new symbols on each of the tapes, moves each tape head independently in possibly in different directions, and changes its control state.

Such a machine can be formally defined as follows.
Definition 4. A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, $\{\vdash, \sqcup\} \subseteq \Gamma \setminus \Sigma$
- $s \in Q$ is the initial state
- $t \in Q$ is the accept state
- $r \in Q$ is the reject state; the states t, and r are assumed to be distinct states
- $\delta : (Q \setminus \{t, r\}) \times \Gamma^k \to Q \times (\Gamma \times \{\mathsf{L}, \mathsf{R}\})^k$ is the transition function; thus, given the current state, and the symbols read by each of the k-heads, the transition function determines what the next state will be, what should be written on each of the tapes, and in which direction to move each tape head.

As always we assume that none of the heads move left, if they are reading $\vdash$; we skip the formal restriction on δ to ensure this.

As for (single-tape) Turing machines, in order to define computations, acceptance, and the language recognized, we need to identify what the configuration of such a machine is. The configuration of such a k-tape Turing machine is a tuple that identifies the current state, the contents of each of k-tapes, and the positions of each of the tape heads. Thus, a configuraion is $\alpha \in Q \times (\{y \sqcup^\omega \mid y \in \Gamma^*\})^k \times \mathbb{N}^k$. For such a machine, the initial configuration on input x is $(s, \vdash x \sqcup^\omega, \vdash \sqcup^\omega, \ldots, \vdash \sqcup^\omega, 0, 0, \ldots, 0)$. The formal definition of a single step based on the transition function is skipped, but is similar to the definition of a single step of a Turing machine. The machine *accepts* input w if it reaches state t, and it *rejects* input w if it reaches state

r. If the machine neither accepts nor rejects on a an input then the machine is said to not halt. Finally, the language recognized/accepted by M is given as $\mathbf{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } x\}$

The ability of a machine to write and read from additional tapes does not increase its computational power. In other words, any decision problem solved on a k-tape Turing machine can also be solved on a single tape Turing machine.

Theorem 5. *For any k-tape Turing Machine M, there is a single tape TM* single(M) *such that* $\mathbf{L}(\text{single}(M)) = \mathbf{L}(M)$.

Proof. Recall that in order to simulate the k-tape machine, the single tape machine needs to keep track of the configuration of M. That means keeping track of the state of M, storing the contents of M's tapes, and keeping track of the the position of each head. How do we do this? This requires us to address the following challenges.

- How do we store k-tapes in one?
- How do we simulate the movement of k independent heads?

We address these challenges in order.

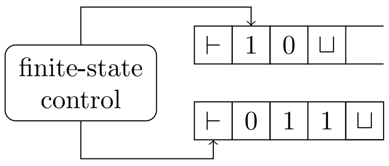

Figure 3: Multi-tape TM M

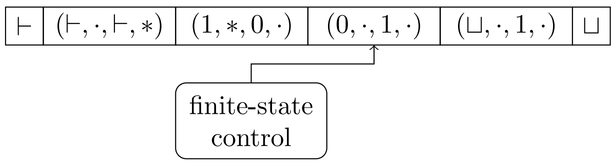

Figure 4: 1-tape equivalent single M

The idea behind storing multiple tapes on a single tape is to store the contents of the ith cell of each tape in the $i + 1$th cell of the single tape machine; we need to right shift by one cell because the 0th cell of the single tape contains the left end marker, and the first cell contains an encoding of the endmarkers of each of the k-tapes. This is achieved by enlarging the tape alphabet of the single tape machine to be the cartesian product of the tape alphabet of the k-tape machine. This is illustrated in Figures 3 and 4. The other piece of information that the single tape machine needs to keep track of are the head positions of the k heads. This it will do by "marking" the cell in which a particular tapes head is positioned. This is indicated by the presence of "*" as shown in Figure 4.

Having outlined how the single tape TM stores the k-tape contents and the k head positions, it is clear that the single tape TM can store the configuration of the k-tape machine by storing the state in its control state. We now need to address how the single tape TM can simulate a single step of the k-tape machine. To simulate a single step of the k-tape machine, we need to find what is being read by each of the k tapes. This can be accomplished by scanning the entire tape from left to right, and storing the contents of the cells that are marked (to indicate the presence of a tape head) in the state as we encounter them. After this scan the 1-tape TM knows what the next step of the k-tape machine will be. But in order to finish its simulation, we need to update the tape to reflect the changed contents of each of the k tapes, and determine the new head positions. This will be accomplished by a couple of scans that update the tape contents, and move the markers to indicate the new head positions on the tape.

Putting these ideas together we can see that the high-level algorithm for the 1-tape TM is as follows. On input x

1. First the machine will rewrite input x to be in "new" format that stores the contents of all k-tapes in each cell.
2. To simulate one step

- the machine will read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
- Next, it will read from left-to-right, changing symbols, and moving those heads that need to be moved right.
- Then, it will scan back from right-to-left moving the heads that need to be moved left.

The above high-level description can be translated into a concrete set of transitions in a straight-forward (but painful) manner. To illustrate how this can be accomplished, we illustrate this process by giving a precise formal definition of this construction.

Formally, let $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$. To define the machine single(M) it is useful to identify modes that single(M) could be in while simulating M. The modes are

$$\text{mode} = \{\text{init}, \text{back-init}, \text{read}, \text{back-read}, \text{fix-right}, \text{fix-left}\}$$

where

- init means that the machine is rewriting input in new format
- back-init means the machine is just going back all the way to the leftmost cell after "initializing" the tape
- read means the machine is scanning from left to right to read all symbols being read by k-tape machine
- back-read means the machine is going back to leftmost cell after the "read" phase
- fix-right means the machine is scanning from left to right and is going to make all tape changes and move those heads that need to be moved right
- fix-left means the machine is scanning right to left and moving all heads that need to be moved left

Now single(M) $= (Q', \Sigma', \Gamma', \vdash, \sqcup, \delta', s', t', r')$ where

- Recall, based on the high-level description, single(M) needs to remember a few things in its state. It needs to keep track of the current "mode"; M's state; during the read phase the symbols being scanned by each head of M; at the end of the read phase, the new symbols to be written and direction to move the heads. Thus,
$$Q' = \{s', t', r'\} \cup (\text{modes} \times Q \times (\Gamma \times \{\mathsf{L}, \mathsf{R}, *\})^k)$$
where s', t', r' are new initial, accept and reject states, respectively. "$*$" is new special symbol that we will use to when placing new head positions, and can be ignored for now. Intuitively, when the mode is "read" the directions don't mean anything, and symbols in Γ will be the symbols that M is scanning. During the "fix" phases the directions are the directions each head needs to be moved, and the symbols are the new symbols to be written.
- $\Sigma' = \Sigma$; the input alphabet does not change
- On the tape, we write the contents of one cell of each of the k-tapes and whether the head scans that position or not. Thus, $\Gamma' = \{\vdash, \sqcup\} \cup (\Gamma \times \{\cdot, *\})^k$, where $\vdash$ will be the left-end-marker, and $\sqcup$ is the blank symbol of the machine.
- The initial state, accept state and reject states are the new states s', t', and r'.

We will now formally define the transition function δ'. We will describe δ' for various cases below; for a case not covered below, we will assume that the machine single(M) goes to the reject state r' and moves the head right.

Initial State In the first step, single(M) will move to the "initialization phase", which will rewrite the tape in the correct format for the future. Thus, from initial state s' you go to a state whose "mode" is init. Since we are going to insert a symbol containing left end-marker of each of the k-tapes, we need to "shift" all symbols of the input one-space to right, which can be accomplished by remembering the next symbol to be written in the state. So $\delta'(s', \vdash) = (s', \vdash, \mathsf{R})$ and $\delta'(s', a) = (\langle \text{init}, s, a, \mathsf{L}, \sqcup, \mathsf{L}, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, (\vdash, *, \vdash, *, \ldots, \vdash, *), \mathsf{R})$ for $a \neq \vdash$; the symbol L doesn't mean anything (and so can be changed to whatever you please).

Initialization In the initialization phase, we just read a symbol and write it in the "new format", which means writing blank symbols for all the other tape cells, and moving right. When we scan the entire input to go back left, i.e., change mode to back-init. There is one caveat to this. We are shifting symbols of the input one position to the right; so we actually write what we remembered in our state, and remember what we read in the state.

$$\delta'(\langle \text{init}, s, a, \mathsf{L}, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, b) = (\langle \text{init}, s, b, \mathsf{L}, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, (a, \cdot, \sqcup, \cdot, \ldots, \sqcup, \cdot), \mathsf{R})$$
$$\delta'(\langle \text{init}, s, a, \mathsf{L}, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, \sqcup) = (\langle \text{back-init}, s, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, (a, \cdot, \sqcup, \cdot, \ldots, \sqcup, \cdot), \mathsf{L})$$

Ending Initialization After we have rewritten the tape, we move the head all the way back, and move to the next phase which is "reading".

$$\delta'(\langle \text{back-init}, s, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, b) = (\langle \text{back-init}, s, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, b, \mathsf{L})$$
$$\delta'(\langle \text{back-init}, s, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, \vdash) = (\langle \text{read}, s, \sqcup, \mathsf{L}, \ldots, \sqcup, \mathsf{L}\rangle, \vdash, \mathsf{R})$$

where $b \neq \vdash$

Reading Here we scan the tape to the right, and whenever we encounter a position, where there is a tape head (i.e., a $*$ in the appropriate position), we will remember that symbol in the state. When we reach the right end (i.e., read a $\sqcup$), we know all the information to determine the next step of M. We will remember the new symbols to right and directions of the head in the state, and move to the next phase back-read where we just go back all the way to the left end. These two cases are formally given as

- Suppose the current state is $P = \langle \text{read}, q, a_1, d_1, \ldots, a_i, d_i, \ldots, a_k, d_k\rangle$, and we read a symbol $X = (b_1, h_1, \ldots, b_i, h_i, \ldots, b_k, h_k)$, where if $h_i = *$ that means that the ith tape head is read this position, and if $h_i = \cdot$ then the ith tape head is not reading this position. Thus,

$$\delta'(P, X) = (\langle q, a'_1, d_1, \ldots, a'_i, d_i, \ldots, a'_k, d_k\rangle, X, \mathsf{R})$$

  where $a'_i = a_i$ if $h_i = \cdot$ and $a'_i = b_i$ if $h_i = *$.

- Suppose the current state is $P = \langle \text{read}, q, a_1, d_1, \ldots, a_i, d_i, \ldots, a_k, d_k\rangle$, and we read $\sqcup$. This means we have finished scanning and all the symbols a_i are the symbols that are being read by M, and its state is q. So suppose M's transition function δ says

$$\delta(q, a_1, a_2, \ldots, a_k) = (q', b_1, d'_1, \ldots b_k, d'_k)$$

  That is, it says "replace symbol a_i on tape i by b_i and move its head in direction d'_i". Then

$$\delta'(P, \sqcup) = (\langle \text{back-read}, q', b_1, d'_1, \ldots b_k, d'_k\rangle, \sqcup, \mathsf{L})$$

So the new state of M, symbols to be written and direction of heads in stored in the state and we go back.

Ending Reading After reading and determining the next step, we move the head all the way back, and move to the next phase which is fixing the tape to reflect the new situation.

$$\delta'(\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, b) = (\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, b, \mathsf{L})$$
$$\delta'(\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, \vdash) = (\langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k\rangle, \vdash, \mathsf{R})$$

where $b \neq \vdash$.

Right Scan of Fixing In this phase we move all the way to the right. Along the way, we change the symbols to new symbols, wherever the old heads were, and move all heads that need to be moved right. To move a head right, we will use "$*$" in the state to remember that the old head position of the tape is the current cell, and it needs to be moved to the next cell. Finally, there is the boundary case of moving the head on some tape to right of the rightmost non-blank symbol on any tape. We will capture these cases formally.

- Let the current state of single(M) be $P = \langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k \rangle$ and let the symbol being read be $X = (b_1, h_1, \ldots b_k, h_k)$. In such a situation, single(M) will move to state $P' = \langle \text{fix-right}, q, a_1, d'_1, \ldots, a_k, d'_k \rangle$, move R and write $X' = (b'_1, h'_1, \ldots b'_k, h'_k)$. P' and X' are determined as follows. If $h_i = *$ (that is, tape i's head was here) and $d_i = \mathsf{R}$ then $b'_i = a_i$ (write new symbol), $h'_i = \cdot$ (new head is not here), and $d'_i = *$ (remember to put head in next cell). If $h_i = *$ and $d_i = \mathsf{L}$ then $b'_i = a_i$ (write new symbol), $h'_i = h_i$ (defer head movement to next phase), and $d'_i = d_i$. If $h_i = \cdot$ and $d_i = *$ (i.e., we remember new head position is here) then $b'_i = b_i$ (don't change symbol), $h'_i = *$ (new head is here), and $d'_i = \mathsf{R}$ (this was the original direction). If $h_i = \cdot$ and $d_i \neq *$ then $b'_i = b_i$ and $d'_i = d_i$.
- Consider the case when the state is $P = \langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k \rangle$, and $\sqcup$ is on the tape. There are two possibilities. If $d_i \neq *$ for every i then we move to state $P' = \langle \text{fix-left}, q, a_1, d_1, \ldots q_k, d_k \rangle$, write $\sqcup$ and move L. On the other hand, suppose there is some i such that $d_i = *$. Then we move to state $P' = \langle \text{fix-right}, q, a_1, d'_1, \ldots a_k, d'_k \rangle$, move R and write $X' = (b'_1, h'_1, \ldots b'_k, h'_k)$, where X' and P' are given as follows. First $b'_i = \sqcup$ for all i. Next, if $d_i = *$ then $h_i = *$ and $d'_i = \mathsf{R}$. On the other hand if $d_i \neq *$ then $h'_i = \cdot$ and $d'_i = d_i$.

Left Scan of Fixing In this phase we move all the way to the left, and along the way we move all the head positions that needed to be moved left. These changes are similar to the case of moving heads to the right, except for the case when moving a head left from the leftmost position.

- Let the current state of single(M) be $P = \langle \text{fix-left}, q, a_1, d_1, \ldots, a_k, d_k \rangle$ and let the symbol being read be $X = (b_1, h_1, \ldots b_k, h_k)$. In such a situation, single(M) will move to state $P' = \langle \text{fix-left}, q, a_1, d'_1, \ldots, a_k, d'_k \rangle$, move L and write $X' = (b_1, h'_1, \ldots b_k, h'_k)$. P' and X' are determined as follows. If $h_i = *$ and $d_i = \mathsf{L}$ (that is this tape's head needs to be moved left) then $h'_i = \cdot$ (new head is not here), and $d'_i = *$ (remember to put head in next cell). If $h_i = *$ and $d_i = \mathsf{R}$ then $h'_i = h_i$ and $d'_i = d_i$ (don't do anything since we already handled the right moves). If $h_i = \cdot$ and $d_i = *$ (i.e., we remember new head position is here) then $h'_i = *$ (new head is here), and $d'_i = \mathsf{L}$ (this was the original direction). If $h_i = \cdot$ and $d_i \neq *$ then $h'_i = h_i$ and $d'_i = d_i$.
- Now we consider the case when we finish the left scan, i.e., the symbol being read is $\vdash$. Let the state be $P = \langle \text{fix-left}, q, a_1, d_1, \ldots, a_k, d_k \rangle$. Since we assume that no tape head moves to the left of the left endmarker, there should be no pending left moves, i.e., $d_i \neq *$ for all i. In this case, we fixed all the moves correctly, and so we move to state $P' = \langle \text{read}, q, a_1, d_1, \ldots, a_k, d_k \rangle$, write $\vdash$, and move R, and start simulating the next step.

Acceptance/Rejection If M accepts/rejects (i.e., its state is t or r) then single(M) will move to its accept/reject state.

$$\delta'(\langle \text{read}, t, a_1, d_1, \ldots, a_k, d_k \rangle, b) = (t', b, \mathsf{L})$$
$$\delta'(\langle \text{read}, r, a_1, d_1, \ldots, a_k, d_k \rangle, b) = (r', b, \mathsf{L})$$

$\square$

2 Nondeterministic Turing Machines

The Turing machines we have considered thus far (whether it being single tape or multi-tape) are *deterministic* machines — at each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt. In contrast, we could consider *nondeterministic* machines where in each step the future is not determined. That is, at each step, there are finitely many possibilities.

Definition 6. Formally, a *nondeterministic Turing machine (NTM)* is $N = (Q, \Sigma, \Gamma, \vdash, \sqcup, \Delta, s, t, r)$, where

- $Q, \Sigma, \Gamma, \vdash, \sqcup, s, t, r$ are as before for the 1-tape deterministic machine
- $\Delta : (Q \setminus \{t, r\}) \times \Gamma \to 2^{Q\times\Gamma\times\{\mathsf{L},\mathsf{R}\}}$

Once again we need to define configurations, and computations of an NTM. A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting/rejecting/halting configurations. A single step $\xrightarrow[N]{1}$ is defined similarly — $(p, z, i) \xrightarrow[N]{1} (q, s^i_b(z), i{+}d)$, if $(q, b, d) \in \Delta(q, z_i)$; here "$i{+}d$" is $i{+}1$ if $d = \mathsf{R}$ and $i{-}1$ if $d = \mathsf{L}$. x is *accepted* by M, if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[N]{*} (t, z, n)$ for some z, n. Finally, $\mathbf{L}(N) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$

Surprisingly, like for NFAs, nondeterminism does not provide any additional computational power.

Theorem 7. *For any nondeterministic Turing Machine N, there is a (deterministic) TM* $\det(N)$ *such that* $\mathbf{L}(\det(N)) = \mathbf{L}(N)$.

Proof. When we translated a NFA to DFA, the DFA kept track of all possible active threads of the NFA; this idea does not generalize very easily. Instead, $\det M$ will simulate M on the input by simulating M on each possible sequence of computation steps that M may try in each step.

In order understand this approach it is useful to consider nondeterministic computation on an input. Recall that any nondeterministic computation can be though of as a tree, where each path corresponds to one of the possible "active threads". This is shown in Figure 5. If $r = \max_{q,X} |\Delta(q, X)|$ then the runs of M

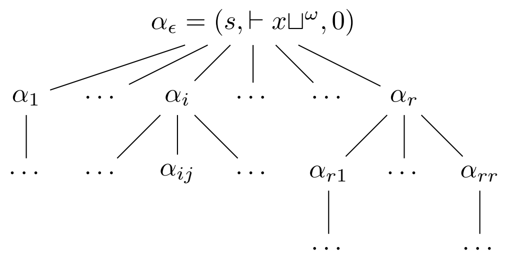

Figure 5: Computation of a nondeterministic machine. Each path in the tree corresponds to the sequence of steps on one of the active threads of the machine.

can be organized as an r-branching tree. $\alpha_{i_1 i_2 \cdots i_n}$ is the configuration of M after n-steps, where choice i_1 is taken in step 1, i_2 in step 2, and so on. Input x is accepted iff $\exists$ accepting configuration in tree.

The idea behind constructing the deterministic machine is that $\det(N)$ will search for an accepting configuration in computation tree by exploring the tree along each path. The configuration at any vertex can be obtained by simulating N on the appropriate sequence of nondeterministic choices. Since the tree may potentially be infinite, if x is not accepted, $\det(N)$ may not terminate.

We now give more details about the machine $\det(N)$. $\det(N)$ will use 3 tapes to simulate N (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called *input tape*, will always hold input x
- Tape 2, called *simulation tape*, will be used as N's tape when simulating N on a sequence of nondeterministic choices
- Tape 3, called *choice tape*, will store the current sequence of nondeterministic choices

det(N) will carry out the following steps.

1. Initially: Input tape contains x, simulation tape and choice tape are blank
2. Copy contents of input tape onto simulation tape
3. Simulate N using simulation tape as its (only) tape
   (a) At the next step of N, if state is q, simulation tape head reads X, and choice tape head reads i, then simulate the ith possibility in $\Delta(q, X)$; if i is not a valid choice, then goto step 4
   (b) After changing state, simulation tape contents, and head position on simulation tape, move choice tape's head to the right. If Tape 3 is now scanning $\sqcup$, then goto step 4
   (c) If N accepts then accept and halt, else goto step 3(a) to simulate the next step of N.
4. Write the lexicographically next choice sequence on choice tape, erase everything on simulation tape and goto step 2.

To summarize the machine det(N) works as follows.

- det(N) simulates N over and over again, for different sequences, and for different number of steps.
- If N accepts x then there is a sequence of choices that will lead to acceptance. det(N) will eventually have this sequence on choice tape, and then its simulation N will accept.
- If N does not accept x then no sequence of choices leads to acceptance. det(N) will therefore never halt!

$\square$

3 Random Access Machine

Random Access Machines are an idealized model of modern computers. They have a finite number of "registers", an infinite number of available memory locations, and store a sequence of instructions or "program" in memory.

- Initially, the program instructions are stored in a contiguous block of memory locations starting at location 1. All registers and memory locations, other than those storing the program, are set to 0.

We will assume that the program consists of the following instruction set.

- `add X, Y`: Add the contents of registers X and Y and store the result in X.
- `loadc X, I`: Place the constant I in register X.
- `load X, M`: Load the contents of memory location M into register X.
- `loadI X, M`: Load the contents of the location "pointed to" by the contents of M into register X.
- `store X, M`: store the contents of register X in memory location M.
- `jmp M`: The next instruction to be executed is in location M.
- `jmz X, M`: If register X is 0, then jump to instruction M.
- `halt`: Halt execution.

Once again such machines do not provide any additional computational power over the model of a 1-tape TM.

Theorem 8. *Anything computed on a RAM can be computed on a Turing machine.*

Proof. We outline a proof sketch. The RAM will be simulated by a multi-tape TM. In order to simulate the RAM, the TM stores contents of registers, memory etc., in different tapes as follows.

- Instruction Counter Tape: Stores the memory location where the next instruction is stored; initially it is 1.
- Memory Address Tape: Stores the address of memory location where a load/store operation is to be performed.
- Register Tape: Stores the contents of each of the registers.
    - Has register index followed by contents of each register as follows: $\#\langle \text{RegisterNumber}\rangle * \langle \text{RegisterValue}\rangle \# \cdots$. For example, if register 1 has 11, register 2 has 100, register 3 has 11011, etc, then tape contains $\#1 * 11\#10 * 100\#11 * 11011\# \cdots$
- Memory Tape: Like register tape, store $\#\langle \text{Address}\rangle * \langle \text{Contents}\rangle \#$
    - To store an instruction, have opcode, $\langle$arguments$\rangle$
- Work Tapes: Have 3 additional work tapes to simulate steps of the RAM

The TM will simulate the RAM as follows.

- TM starts with the program stored in memory, and the instruction location tape initalized to 1.
- Each step of the RAM is simulated using many steps.
    - Read the instruction counter tape
    - Search for the relevant instruction in memory
    - Store the opcode of instruction and register address (of argument) in the finite control. Store the memory addres (of argument) in memory address tape.
    - Retrieve the values from register tape and/or memory tape and store them in work tape
    - Perform the operation using work tapes
    - Update instruction counter tape, register tape, and memory tape.

To illustrate the above sequence of steps, consider the example of simulating an add instruction.

- Suppose instruction counter tape holds 101.
- TM searches memory tape for the pattern $\#101*$.
- Suppose the memory tape contains $\cdots \#101 * \langle \texttt{add}\rangle, 11, 110\# \cdots$
- TM stores "`add`", 11 and 110 in its finite control. In other words, it moves to a state $q_{\texttt{add}\ 11,110}$ whose job it is to add the contents of register 11 and 110 and put the result in 11.
- Search the register tape for the pattern $\#11*$. Suppose the register tape contains $\cdots \#11 * 10110\# \cdots$; in other words, the contents of register 11 is 10110. Copy 10110 to one of the work-tapes.
- Search the register tape for pattern $\#101*$, and copy the contents of register onto work tape 2.
- Compute the sum of the contents of the work tapes
- Search the register tape for $\#11*$ and replace the string 10110 by the answer computed on the work tape. This may involve shifting contents of the register tape to the right (or left).
- Add 1 to the instruction counter tape.

$\square$

4 Church-Turing Thesis

Over past several decades, various efforts to capture the power of mechanical computation have resulted in formal models that have the same expressive power.

- Non-Turing Machine models: random access machines, λ-calculus, type 0 grammars, first-order reasoning, π-calculus, ...
- Enhanced Turing Machine models: TM with 2-way infinite tape, multi-tape TM, nondeterministic TM, probabilistic Turing Machines, quantum Turing Machines ...
- Restricted Turing Machine models: queue machines, 2-stack machines, 2-counter machines, ...

This has led to what is called the Church-Turing thesis which states

"Anything solvable via a mechanical procedure can be solved on a Turing Machine."

The Church-Turing thesis is not a mathematical statement that can be proved or disproved! Our belief in it is based on the fact that many attempts to define computation yield the same expressive power as Turing machines. As a consequence, in the course, we will use an informal pseudo-code to argue that a problem/language can be solved on Turing machines.

5 Universal Simulation

A crucial observation about Turing machines is that there are Turing machines, called *Universal Turing machines*, that can simulate other Turing machines when given their encoding as inputs. A universal Turing machine U is such that given the necoding of Turing machine M and an input x, U can simulate M on x and accept if M accepts, and reject if M rejects.

Before describing how such a machine might work, we need to describe how an arbitrary Turing machine M can be encoded as a string. Since U is a fixed machine, it's tape alphabet is also fixed. However, it needs to be able to simulate TMs with arbitrary tape alphabet. This can be achieved by describing an encoding of any such machine using a fixed alphabet.

5.1 Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$. We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$. This will be built using encodings of symbols $a \in \Gamma$, and states $q \in Q$. A symbol $a \in \Gamma$ will be encoded as a binary string enc(a) of length $\log |\Gamma|$. We assume that $\vdash = 0^{\log|\Gamma|}$ and $\sqcup = 0^{\log|\Gamma|-1}1$. Similarly, each state $q \in Q$ will be encoded as a binary string enc(q) of length $\log |Q|$. We will assume that $s = 0^{\log|Q|}$, $t = 0^{\log|Q|-1}1$ and $r = 0^{\log|Q|-2}10$. Finally, directions L and R will be encoded as 0 and 1, respectively.

Turing machines will be encoded by encoding all its transitions as a string. A transition $\delta(p, a) = (q, b, d)$ is encoded as $[\text{enc}(p) \bullet \text{enc}(a) | \text{enc}(q) \bullet \text{enc}(b) \bullet \text{enc}(d)]$. Then code of the machine itself is just the encoding of all its transition concatenated together, i.e.,

$$[\underbrace{\qquad}_{\text{trans 1}}\underbrace{\qquad}_{\text{trans 2}}\cdots\cdots\underbrace{\qquad}_{\text{last trans}}]$$

An input string $x = a_1 a_2 \cdots a_n \in \Gamma^*$ will be encoded as $[\text{enc}(a_1) \bullet \text{enc}(a_2) \bullet \cdots \bullet \text{enc}(a_n)]$.

The above scheme outlines one possible way in which a Turing machine and its input string may be encoded. The precise choice of the alphabet and the exact encoding scheme is not important. It has been outlined to give a concrete example of one such scheme. Therefore, it is important not to place too much emphasis on its details. For what follows, we could use encoding scheme that is easy to interpret and is capable of encoding all Turing machines upto isomorphism.

We will now describe a universal Turing machine U that will recognize the following language.

$$\mathbf{L}(U) = \{\text{enc}(M)\#\text{enc}(x) \mid x \in \mathbf{L}(M)\}$$

In other words, given the encoding of a Turing machine M and an encoding of an input x, the machine U determines if x is accepted by M. We will often abuse notation and instead write the language of U as

$$\mathbf{L}(U) = \{M\#x \mid x \in \mathbf{L}(M)\}.$$

It is important not to get confused by such a notation.

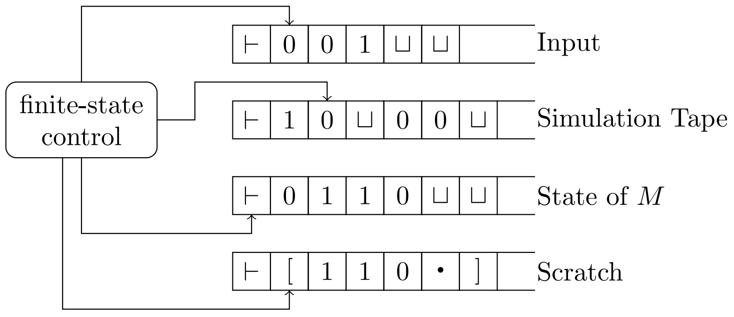

Figure 6: A universal Turing machine

The machine U is shown schematically in Figure 6. It has 4 tapes. The *input* tape stores the encoding $M\#x$. The *simulation* tape will be used as the tape of M when U "runs" M, and the state of M will be stored on the *state* tape. Thus the configuration of M during the simulation is stored on the simulation and state tapes. It is important to realize that the state of M cannot be stored in U. This is because U has a fixed number of states, whereas since M could be any machine (not know at the time of designing U) its states cannot be bounded. The *scratch* tape will be used by U to auxiliary information as it simulates M.

The machine U will work as follows.

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one "|" between [and], etc.

2. If code is ok, then copy x onto tape 2

3. Write $0 \cdots 0$, the start state of M, on the third tape, and scan the "first cell" of tape 2.

4. To simulate a move of M, search for a transition $[\text{enc}(p) \bullet \text{enc}(a) | \text{enc}(q) \bullet \text{enc}(b) \bullet \text{enc}(d)]$ on tape 1, where $\text{enc}(p)$ is on tape 3 (current state) and $\text{enc}(a)$ is read from tape 2 from the current "cell", i.e., between two successive $\bullet$ symbols from the current head position. Then, write $\text{enc}(q)$ on tape 3 (after erasing its current contents), write $\text{enc}(b)$ on tape 2 instead of $\text{enc}(a)$, and finally move the head on tape 2 to the appropriate "cell".

5. If state on tape 3 is $0 \cdots 01$ then accept; if state is $0 \cdots 010$ then reject.