CSCI 341–Fall 2023: Lecture Notes
Set 13: Modified Turing Machines

Edward Talmage

October 25, 2023

We have repeatedly extended our Finite State Automata to get stronger machines. Can we do something similar to get a stronger version of a Turing Machine?

1 2 Tapes

Suppose we have two tapes instead of one. The input starts at the beginning of tape 1, second tape and rest of first are initially blank. Each tape has an independent Read/Write head, so the transition function is now

$$\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R\}^2$$

That is, we take a state and a symbol from each tape, write new symbols to both tapes, and move both Read/Write heads independently.

Claim 1. *A 2-tape TM is at least as strong as a TM.*

This is easy to see, as we can just choose to never do anything with the second tape. That is, update each $\delta(q, x) = (q', x', D)$ to be $\delta(q, x, \sqcup) = (q', x', \sqcup, D, L)$.

Claim 2. *A TM is at least as strong as a 2-tape TM.*

Proof. We prove this by constructing a TM that can simulate a 2-tape TM. Set up the tape as $\#w\#\sqcup$, where w is the input. The # symbols will tell us where the beginning of each simulated tape is. That is, everything between the #'s is the first tape, everything after the second # is the second tape. Track the locations of the two Read/Write heads by having special marked versions of each symbol, initially mark the first entry after each # and update the marks in every transition.

To simulate a transition of the 2-tape machine, walk across the tape left-to-right. Store the contents of the two marked cells and compute the 2-tape transition. Go back and update both heads, and the values of the cells to which they point, accordingly.

> **Exercise:** Why can we store the contents of the cells at the two tape heads? We cannot store their indices.

If the first tape needs to be longer, which we can detect by the first Read/Write head mark moving right onto a #, we can walk to the right end and shift the entire second tape right one cell. □

2 Many Tapes

Suppose we have a TM with k tapes, where k is any positive integer greater than 1. Note that k must be finite, so we can only do finite work in a single step. A similar argument applies as that for 2 tapes, using multiple # signs to divide the tape into as many pieces as needed. We can still store the contents of all current cells in the DFA state since it is a finite tuple of characters from a finite set. Since the machine is still controlled by a DFA, we can simulate the transition function, as well.

3 Doubly-Infinite Tape

Now, the tape is infinite in both directions. The tape head starts on the first character of the input, but there is no other positional reference.

Simulating a singly-infinite tape is slightly harder than it was with two tapes, since we have to simulate the behavior of a TM that bounces off the left end of its tape. (Never mind how bad an idea it may seem to build a TM that ever does that, we want a complete simulation.) We can add a special symbol one space left of the first symbol of the input, then step back right without changing anything anytime we see that symbol.

Exercise: How would you simulate a doubly-infinite tape TM with a singly-infinite tape TM?

Fold the tape in half! That is, increase Γ so that each cell stores two values from the doubly-infinite TM's tape. Now each "index" i (there are no indices in TMs, but we can picture them) stores the values at index i and index $-i$. (Technically, we have 0 and -0, but we can just have our singly-infinite TM 1-indexed and skip over 0.) If Q is the state set of the doubly-infinite TM, our new state set is $Q \times \{+, -\}$, tracking whether we are in a positive- or negative-index cell. That tells the transition function which value to use, and we reverse the direction the doubly-infinite TM moves when we are at a negative index.

4 Non-Determinism

While DFAs and NFAs were equivalent, deterministic and non-deterministic PDAs were not (though we did not spend time on that point). What happens if we add non-determinism to a Turing Machine?

First, note that it is trivial to implement a non-deterministic TM given a deterministic one, as we can return the input machine.

Next, how would we simulate a non-deterministic TM using a deterministic one? A non-deterministic TM has transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ and accepts if any path reaches q_{accept}. Our idea is to organize all reachable configurations of the NTM on a given input w in a tree, rooted at the start configuration $q_0 w$, and search the tree to see if any branch reaches q_{accept}. We will have to be careful not to follow an infinite branch, as that would mean we never get around to exploring the rest of the tree, so we will do a Breadth-First Search. Thus, if a branch halts in finite time, that is at finite depth, we will reach it in finite time.

To track everything that is going on, we will build a 3-tape TM. As seen above, this is equivalent to a basic TM, so if we can build a 3-tape deterministic TM equivalent to an arbitrary NTM, then there is also a 1-tape deterministic TM equivalent to that NTM. The tapes will behave as follows:

1. Stores the input. Never edit the tape, just read it.
2. Store a copy of the input, and use for working space. We run the simulation of a single path in the NTM on this tape.
3. Keep track of where we are in the tree of configurations. Call this the address tape.

To understand this construction, we need to figure out how to track our BFS in the address tape. Let b be the size of the largest set output by the transition function δ. This must be finite, as a TM can only have finitely many next configurations on a given input. We will use $\{1, 2, \ldots, b, \sqcup\}$ as the alphabet for the address tape. Strings over this alphabet describe a path in the configuration tree, or the sequence of non-deterministic choices that led to the current configuration.

- The empty string (address tape is all $\sqcup$'s) is the root, corresponding to the start configuration where we have taken no steps.
- A string like 1432 indicates the first "process" spawned in stepping from the root, then the fourth process spawned from that configuration, then the third and second processes at each of the next two steps.
- In general, a k-digit string indicates that the TM has stepped k times, and the i^{th} digit indicates which non-deterministic choice we took in step i.
- Some (many) address strings may be impossible paths. For example, if there is only one possible step from the start configuration, then the string 21732 is not a valid configuration. This is not a problem, as we will just skip ahead to the next possibility.

To run our deterministic TM and simulate NTM N:

1. Initially, input is on tape 1, tapes 2,3 are empty.
2. Copy input to tape 2.
3. Use tape 2 to simulate N on input w. Before each step of N, look at tape 3 to see which branch to try next.
   - If tape 3's current location is empty or specifies a non-existent choice (tape 3's value is higher than the number of non-deterministic branches from the current configuration), we are in an illegal branch, so skip ahead to step 4.
   - If we reach q_{accept}, accept and terminate.
   - If we reach q_{reject}, the current branch is not useful, so goto step 4.
4. Put the next path to explore on tape 3. Goto step 2.
   - The next path to explore, in breadth-first order, will be the next integer in base b.
   - Overall, we will the address tape will count $1, 2, \ldots, b, 11, 12, 13, \ldots, 1b, 21, \ldots, 2b, 31, \ldots, bb, 111, \ldots$

If some branch of N accepts after n steps, then our TM will halt and accept after at most b^n iterations, when the path that led to accept appears on the address tape. If N never accepts w, then our simulation will run forever.

5 Enumerators

There is an alternate way to consider TMs, based on the idea that it would be nice if a TM would list all of the strings in its language, instead of just checking one input per run.

Definition 1. An *enumerator* E is a TM with a printer. The tape is initially empty. Every time the TM reaches a particular state (the *print state*), it prints the contents of the tape to the printer. $L(E) = \{w \mid E \text{ prints } w\}$.

Theorem 1. *A language is enumerable iff it is Turing-Recognizable.*

Proof. We must prove both directions of the equivalence.

($\Rightarrow$) Assume we have an enumerator E. We will build a TM to recognize $L(E)$.

1. On input w, run E (we can use second and third tapes to simulate it).
2. When E prints a string, compare it to w. Accept if equal.

E will eventually generate every string in $L(E)$, so we will eventually accept for every $w \in L(E)$.

($\Leftarrow$) Assume we have a TM M that recognizes language L. We will build an enumerator for L.

1. Let $s_1, s_2, s_3, \ldots$ be some fixed ordering of all strings in Σ^*.
2. For $i = 1, 2, 3, \ldots$ to infinity, repeat the following:
   (a) Simulate M for i steps on each input $s_1, \ldots, s_i$.
   (b) If any of these simulations reach q_{accept}, print the corresponding s_i.

By using a fixed order and only computing a limited number of steps for each input in each iteration, if M accepts s_i in a finite number of steps, then we will print it after a finite number of steps, so we have built an enumerator for L. □