CS 4510 Automata and Complexity	3/28/2022
Lecture 14	
Lecturer: Frederic Faulkner	*Scribe(s):*

1 More definitions

We said last class that a TM accepts a string by entering q_{accept}, and it rejects a string by entering q_{reject}. So what happens if it never does either? It turns out that TM's have a third option; for a given input they either accept, reject, or run forever. Unfortunately you might never be sure that a machine is actually running forever, since even if you run it for 1000 years it might still finish tomorrow! This leads to a useful distinction between Turing machines.

Definition 1 *We say that a TM recognizes L if it accepts x if and only if $x \in L$.*

Definition 2 *We say that a TM decides L if it recognizes L and always rejects x if $x \notin L$.*

So a decider is in some sense nicer than a mere recognizer, because a decider always halts eventually, but a recognizer may run forever on inputs not in its language.

2 Conversion between models

2.1 All you need to know

Suppose you have a Turing machine currently computing some important function for you. It has been running for several days now, and you are sure it will give you the answer soon. However, you have an urgent need to use the TM for something else right now; rather than lose all of your progress, it would be nice if you could just write down everything about the current state of the machine. Then, later, you could pick up where you left off rather than starting over. What would you need to write down?

Well, the current state of the machine seems relevant, and the current contents of the tape. Anything else? Recall that a TM is roughly (a DFA + a tape + a pointer). You need to keep track of which character you were looking at.

It turns that this is enough! If you have just these three pieces of information (tape contents, current state, head location), the machine can pick up where it left off as though nothing had happened. They serve as a "snapshot" of all of the essential information about the progress of the machine. We call these three pieces of information the *configuration* of the machine. The conventional way of writing a TM configuration is something like $01q_1001$, representing that the machine is in state q_1, the tape currently reads 01001, and the head is reading the second 0.

2.2 Multiple tapes

A k-tape Turing machine has, as its name suggests, k tapes and k heads. At each step it decides which transition based on its current state and the k letters it is currently reading. It performs k write operations (one per head) simultaneously, and then moves each head left or right (each head can move a different direction). In all other ways it is a normal Turing machine. Does increasing the number of tapes increase the power of the Turing machine?

Theorem 3 *Any k-tape Turing machine has an equivalent 1-tape Turing machine.*

Proof. Let M be some k-tape Turing machine. We only have a single tape to keep track of k tapes, so we can use a special marker, say #, to divide up our tape into k sections, one for each tape of M. In addition to storing the tape, we must mark where the head for each tape is pointing. So for example if $k = 2$ and our tape read $00\dot{1}\#1\dot{0}00$, that would represent that M's two tapes contained 001 and 1000, and the two heads were located above the 1 on the first tape and the leftmost 0 on the second tape.

At each step, we must first scan the entire tape to see each of the characters that M is looking at. We can then move down the line updating each tape section one at a time as appropriate. Note: if we run out of space for one tape, we must pause the simulation of M to move all of the characters to the right to make more space.

2.3 Non-determinism

For DFA's we saw that adding non-determinism, while handy, doesn't increase the power of that model. On the other hand, although we didn't mention it in class, PDA's are sensitive to non-determinism, in that PDA's (which are default non-deterministic) are more powerful than deterministic pushdown automata. So, when you add non-determinism to a TM, does it get more powerful (like a PDA) or not (like a DFA)?

Theorem 4 *Every non-deterministic TM has an equivalent deterministic TM.*

First, let's clarify what we mean by a non-deterministic TM, or NTM. An NTM is defined by the same 7-tuple $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$ as a deterministic TM except that δ has a co-domain of $P(Q \times \Gamma \times \{L, R\})$ instead of just $Q \times \Gamma \times \{L, R\}$. Thus an NTM can have multiple options for which state to go to, which character to write, and which direction to move the head. An NTM N accepts a string x if any sequence of choices causes N to reach an accept state on input x.

For the proof, it may be helpful to think of an NTM's computation as a tree, where the top node is the starting configuration, and the children of a node are all the configurations reachable from that node in one step of the computation. Then the NTM accepts if there is any path from the root to an accepting leaf.

We can use a deterministic TM to run a BFS on this tree looking for an accepting computation. In other words, we will consider all possible configurations after running the original NTM for one step, then for two steps, then for three steps, etc.

Proof. Let N be an NTM, and we will construct an equivalent deterministic Turing machine D. D will have 3 tapes.

1st tape: contains the input to N. This tape never changes. We are going to keep restarting the computation of N, so we need to remember where to start.

2nd tape: the simulation tape. This tape will act like N tape as we simulate it for some number of steps.

3rd tape: the "plan" tape. Each time we restart N we want it to make different choices than the last we ran it so that we can try every possible sequence of choices. So this tape will contain a string like "3121". This corresponds to simulating N for 4 time steps, where at the first step we make choice "3", at the second time step we make choice "1", at the third time step make choice "2", and at the fourth time step we make choice 1.

Some number sequences will not be valid because, say, there might not be 3 available choices to pick from. That is okay, just restart again and try the next sequence.

Our TM simulates N by doing the following:

1. Write a 0 on tape 3.
2. Clear tape 2 if necessary and copy tape 1 to tape 2.
3. Simulate N on tape 2. At each step, use tape 3 to decide which choice to make. As you simulate N, if it accepts, we accept. If it rejects, go to step 4. If we reach the end of tape 3, also go to step 4.
4. Increase tape 3 to the lexicographically next sequence. (So 0 goes to 1, 132 goes to 133, etc.) Go back to step 2.

As the sequence on tape 3 grows, we will simulate N for longer amounts of time.

Note: Why do we have to restrict the number of steps that N runs for? Why can't we just start simulating N and keep going until we accept or reject?