Alex Hubers
CSC:341 Spring 2024

Lecture Notes #7A

#1 Whoops

I want to introduce a convention that I did not introduce last week but is ultimately pretty darn useful.

In Arora and Barak, we have this special symbol $\triangleright$ with the property the *input tape* and all *work-tapes* start with $\triangleright$ printed in the tape's first cell. Why? Recall a sentence like "... and regress the head back to the start of the tape" from last week. Imagine that you are on the configuration below with the head above x_3.

$$\begin{array}{|c|c|c|c|}\hline x_1 & x_2 & x_3 & \cdots \\ \hline\end{array}$$

Now, ask yourself, "and how do I know that I made it back to the start?" You might have

$$\delta(q_{regress}, x_i) = (..., ..., L)$$

But once you reach x_1 you will try to move left forever. If we want to know where the start is, we have to mark it as such.

$$\begin{array}{|c|c|c|c|c|}\hline \triangleright & x_1 & x_2 & x_3 & \cdots \\ \hline\end{array}$$

And now:

$$\delta(q_{regress}, \square) = (q_{foobar}, ..., ...)$$

You may feel free to adopt this convention whenever it is helpful.

#2 Motivation (NDTMs)

1. We can't really discuss the class NP without talking about nondeterministic Turing machines.
2. All forms of computation that we know of are equivalently expressive to Turing machines, *and that includes non-deterministic Turing machines!*. So we have another result saying that parallelism is no more expressive than determinism.

The rest of the motivation for nondeterminism—you know, that it models parallel computing, and so forth—I have given already in our lectures on NFAs. (Sipser §3.2)

#3 Formal Definition

Definition 1 (Nondeterministic Turing Machine (NDTM)). *A nondeterministic Turing machine, or NDTM, is a Turing machine with all components equal except for δ, which has type*

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$$

We write

$$(q, x, i) \Rightarrow (k, y, j)$$

if $((k, y, j) \in \delta(q, x, i)$, and

$$(q, x, i) \Rightarrow^* (k, y, j)$$

if there exists a sequence of yields resulting in (k, y, j) from (q, x, i).

$$(q, x, i) \Rightarrow ... \Rightarrow (k, y, j)$$

(Call such a sequence a trace of configurations.) Finally, we say that an NDTM N accepts $x \in \Sigma^*$ if there is at least one trace in which x yields an accept state. We say N rejects x if all traces yield a reject state.

N.b. that this is the same trick we used to turn DFAs into NFAs.

There is an alternative definition of NDTMs that you may find more helpful.

Definition 2 (NDTMs (Arora & Barak version)). *An NDTM is a Turing Machine with at least* 2 *transition functions* δ_0 *and* δ_1 *(not necessarily unequal).*

The definitions of configurations, traces, acceptance, and rejection are the same as above (but appropriately reworded).

I spare you the trivialities, but you should be able to see (with some thought) how to convert back and forth between having multiple transition functions versus having one transition function that returns sets as values. Feel free to use either definition as needs be.

#4 Example

Here are two examples using Arora & Barak definition.

Example 1. Let M be a TM deciding some language L with transition function δ and let N be an NDTM where $\delta_0 = d_1 = \delta$. Obviously, N also decides L, and the behavior of M and N are identical.

Example 2. Let N be a 2-tape Computing NDTM with the following transition functions:

δ_0	δ_1
$(q_s, \rhd, \rhd) \to (q_{scan}, \rhd, \rhd, R, R)$	$(q_s, \rhd, \rhd) \to (q_{scan}, \rhd, \rhd, R, R)$
$(q_{scan}, 0, \square) \to (q_{scan}, 0, \square, R, R)$	$(q_{scan}, 0, \square) \to (q_{scan}, 1, \square, R, R)$
$(q_{scan}, 1, \square) \to (q_{scan}, 0, \square, R, R)$	$(q_{scan}, 1, \square) \to (q_{scan}, 1, \square, R, R)$
$(q_{scan}, \square, \square) \to (q_H, \square, \square, S, S)$	$(q_{scan}, \square, \square) \to (q_H, \square, \square, S, S)$

In class we can think about what this looks like as a tree. We can also think about what this would look like using Sipser's NDTM definition.

#5 Equivalence

Claim 1. *Every NDTM has an equivalent deterministic Turing Machine.*

See Sipser Theorem 3.16 for a proof. I am not sure it is particularly helpful for you to understand this one in too great a depth. More important (later) is that you understand the relationship between verifiers (to be defined later in the course) and NDTMs.