Lectures and Reading. This week finishes the equivalence of regular expressions and finite automata, followed into Monday by the Myhill-Nerode technique for showing languages to be non-regular. We will then rejoin the Arora-Barak notes. To smooth the transition, I’ve given out the three CRC Handbook on Algorithms and Theory of Computation chapters that preceded mine with Allender and Loui already given out (which we will get to next month). My name was added to the last after I gave substantial edits and additions. Skip most of the chapter on grammars, reading only up to its section 2.1 on regular expressions, and in the “Basic Notions” chapter, also skim/skip “alternating” and “oracle” TMs. That chapter also has brief overviews of the NFA/DFA/regexp proofs from this week which may help to summarize the longer Rochester notes/slides.

The order of business after Myhill-Nerode will start with the equivalence of 1-tape and multitape TMs for computability and space complexity, though with a quadratic slippage in time complexity—this is “Claim 1.6” on page 20 of Arora-Barak, and is similarly skimmed by Jiang-Li-Ravikumar. My ulterior motive for spending time on the proof is to illustrate the idea of a master “Read-Evaluate-Write” loop. This idea carries into a topical understanding of my “RAM Simulator” handout which was also given out. It could become a proof of the old flowchart theorem that only one while loop is ever needed (see https://en.wikipedia.org/wiki/Structured_program_theorem) but that’s just a remark. The main point for me is that a universal machine is created out of elementary operations (most of which we’ve seen: copying, matching, simple arithmetic, testing bits at specific times) plus just one control structure. It carries a quartic—that is, $\tilde{O}(t^4)$ where the tilde $\sim$ means to ignore factors of $\log t$—slippage in time complexity if we take the worst case of what can happen when the handout’s machine is run. (For a technote, just FYI, there is a way to implement a caching scheme that gets the overhead down to $\tilde{O}(t^3)$—and to $\tilde{O}(t^2)$ if the RAM operations are timed at bit-cost rather than unit-cost. But this is beyond the purpose of saying that all the forms are equivalent for defining the class $\mathcal{P}$ for polynomial time, for which $O(t^4)$ etc. suffices.)

October will then begin with computability and undecidability from Arora-Barak chapter 1 and the JLRR chapter “Computability.”

(1) Convert the following NFA $N$ with $\lambda$-transitions into an equivalent DFA. The code for $N$ has $Q = \{1, 2, 3, 4\}$, $\Sigma = \{a, b\}$,

$$\delta = \{(1, \lambda, 2), (1, a, 3), (2, a, 2), (2, b, 4), (3, b, 2), (3, b, 4), (4, a, 4), (4, b, 1)\},$$

$s = 1$, and $F = \{2\}$. (18 pts.)

(2) Calculate a regular expression over $\Sigma = \{a, b\}$ for the language of strings that are not accepted by the following NFA: $Q = \{s, q, f\}$, $F = \{f\}$, and

$$\delta = \{(s, a, q), (s, b, f), (q, b, s), (q, a, f), (f, a, s), (f, \lambda, q)\}.$$

(Note that if the last instruction were on $b$ not $\lambda$ it would be a DFA.) You must use a strategy based on theorems in lectures and posted notes, not just inspection (that is, “hacking”). (24 pts., for 42 total on the set)