The Final Exam is on Tuesday, May 10, in the “middle time” 11:45am–2:45pm in the lecture room, Hochstetter 114. The rules are closed book, open notes, no electronics. Ideally notes should be in one binder or folder, to minimize the space you take next to your neighbors. There is no need for a calculator. If you bring a backpack, we will ask you to stow it on the side of the room.

Reading and Coverage for the Last Week

Tuesday’s lecture will finish Chapter 5 and start Chapter 7 up through section 7.3. Note that lectures have already previewed the concepts of linear time (versus \( O(n^2) \) time), so the idea of polynomial time is not much of a stretch. We assume you’ve had \( O,o \)-notation from CSE250 and/or other courses, and note that Theorem 7.8 is basically a repeat of Theorem 3.13 since the quadratic running time was already observed in lecture. Hence the pickup will be quick until the issue of time for nondeterministic TMs in Definition 7.9, and then go on to define the classes \( P \) and \( NP \). I will relate the PATH problem to how we solved \( E_{DFA} \) in lecture, but the proof that every CFL belongs to \( P \) will be handwaved.

Into Thursday, I intend to jump over HAMPATH and CLIQUE and SUBSET-SUM in section 7.3 to emphasize SAT as the primary example of a problem in \( NP \). Thursday’s lecture will focus on the Cook-Levin Theorem in section 7.4. When you hit the word “tableau” stop—there is an “alternative proof” of it as Theorem 9.34 at the end of chapter 9, so we will jump to section 9.3.

Here are notes for this jump—shorter even than the text’s proof. The key is that the entire tableau diagram can be quickly converted into a roughly \( n^k \times n^k \) sized circuit of Boolean NAND gates \( g \). The circuit has input nodes \( w_1, \ldots, w_n \) assigned to the respective bits of \( w \) and extra input nodes \( y_1, \ldots, y_p \) that are left unassigned and represent possible nondeterministic choices by the NTM \( N \). NAND is a universal gate, and we can code up the TM’s states and alphabet symbols via binary strings, so Boolean logic can simulate the entire diagram with one output wire \( w_o \) that equals 1 if the last row (or some row before it) is an accepting configuration, \( w_o = 0 \) if not. Indeed, the circuit has a highly regular structure based on overlapping tiles of the \( 2 \times 3 \) “window” shown in the text’s diagram, so it’s a bit like repetitive stamped-out chip wafers. The slogan for this handwave is: “Software Can Be Efficiently Burned Into Hardware.”

Then every wire—not just the output wire or the variable input wires \( y_1, \ldots, y_p \)—can be labeled by a logical variable \( v \). If a NAND gate \( g \) has incoming wires \( t \) and \( u \) and all of its output wires labeled by the same variable \( v \) (because they will all hold the same value \( t \) NAND \( u \)), then it works properly provided these three clauses all hold:

\[
(t \lor v) \land (u \lor v) \land (\bar{t} \lor \bar{u} \lor \bar{v}).
\]

Try all 8 assignments to the three variables—4 of them are correct and the other 4 give the wrong value \( v \) for NAND—and you’ll see this formula accurately expresses that the NAND gate is working correctly. Finally make a Boolean formula \( \phi \) that does a big AND of these threesomes over all the gates \( g \), finally AND-ed with \( (w_o) \) as a clause by itself. The punch line:

\[
N \text{ accepts } w \iff \text{the text's tableau has an accepting computation} \iff \text{some setting of the nondeterministic inputs } y_1, \ldots, y_p \text{ makes the corresponding Boolean circuit output } w_o = 1 \iff \text{that setting determines values for all the other wire variables } t, u, v, \ldots \text{ that make each NAND gate check out} \iff \phi \text{ is satisfiable}.
\]

The whole translation from \( \langle N, w \rangle \) to the formula \( \phi \) is basically computable in \( n^k \times n^k = n^{2k} \) time, which is polynomial, so \( L(N) \leq_P \text{SAT} \). Since \( L(N) \) can be any language in \( NP \), and SAT is in \( NP \), this means SAT is \( NP \)-complete.
If this “clicks” for you—besides having a little flavor of VLSI logic verification—then it saves you the remaining 5 painful pages of the text’s “traditional” proof in section 7.4. I may even post a handout to help connect “tableau” to the second half of section 5.1. If it doesn’t click, don’t worry—the fact of Theorem 7.27 which was thereby proved is what you need to know. The last lecture is necessarily “ipso-factoid” in nature anyway since its contents cannot be covered by the last problem set. The detailed proofs in section 7.5 will be skipped—as always happens—but there may be time for the simple reductions from CLIQUE to Vertex Cover and back.

Assignment 10—due Thu. 5/7 in lecture

(1a) Prove that the following decision problem is undecidable:

INSTANCE: A Turing machine $M$ with input alphabet $\Sigma = \{0, 1\}$.

QUESTION: Does $M$ accept any string of the form $0^n1^n$ where $n \geq 1$?

First state what the language $L$ of this problem is using ordinary set-builder notation like in the text and lectures. Then prove it is undecidable by reduction from $A_{TM}$ (or from $K$ if you prefer). Finally answer: is $L$ recognizable (synonymously: recursively enumerable, computably enumerable, r.e., c.e.)? (3+18+6 = 27 pts.)

(1b) Prove that the language $\{\langle M \rangle : M$ is a DTM and $L(M) = L(M)^R\}$ is undecidable. Note that the only difference from the decidable problem (1) of PS9 is that $M$ is a deterministic TM not a DFA, but it’s a big difference. Condition: Your proof must already follow from the reduction construction in problem (1a). So this is worth just 6 pts. for the tweaked analysis.

(2) Consider the text’s algorithm for deciding the $A_{NFA}$ problem in Theorem 4.2. As a language, $A_{NFA} = \{\langle N, w \rangle : N$ is an NFA and $w \in L(N)\}$. Let $n = |w|$ and let $m$ be the number of states in $N$, so that the overall length of the input $x = \langle N, w \rangle$ is roughly (order-of) $m+n$. Now the text’s algorithm can take time order-of $n2^m$ in worst case because the NFA $N$ might blow up to a DFA of about $2^n$ states. Work out a different algorithm that decides the problem in time roughly $O(mn)$ or at worst $O(m^2n)$. Here “roughly” means that you may ignore $O(\log m)$ factors—that is, treat each state of $N$ as a unit although it really has a binary-number label of length $O(\log m)$. Since this is quadratic or at worst cubic in $m+n$, what you have done is classify the $A_{NFA}$ problem into deterministic polynomial time. (Think of how you would actually simulate all the possible states as $w$ is read bit-by-bit. 12 pts.)

(3) Now consider the following problem which is not quite the same as what the text’s general notation scheme would call “$ALL_{NFA}$”:

INSTANCE: An NFA $N = (Q, \Sigma, \delta, s, F)$ and a number $n$, where $1 \leq n \leq |Q|$.

QUESTION: Is $\{0, 1\}^n \subseteq L(N)$?

This is just asking whether $N$ accepts all binary strings of the given length $n$, not whether it accepts all strings as with “$ALL_{NFA}$.”

(a) State the language $L$ of this problem and also state its complement $L'$—ignoring inputs that are not valid codes $\langle N, n \rangle$. (6 pts.)

(b) Show that $L'$ belongs to NP. That is, use your answer to problem (2) to give a polynomial-time verifier for the cases where $N$ fails to accept some string of length $n$. (6 pts.)

(c) Show that $L$ polynomial-time many-one reduces to $ALL_{NFA}$. Given $N$ and $n$, you need to create an NFA $N'$ that also accepts all strings of length strictly less than $n$ and all strings of length strictly more. Sketch abstractly how to build $N'$ to accomplish the reduction. (9 pts., for 21 on the problem and 66 on the set; the language $L'$ is in fact NP-complete.)