CSE396, Spring 2017  Problem Set 10  Due Thu. 5/11** in class

The Final Exam is set for Tuesday, May 16 in the lecture room, Cooke 121, 11:45am–2:45pm. We have decided to repeat the terms used for last year’s exam in the identical room: one notes binder is allowed. The textbook is not allowed, and all electronic devices are forbidden. If you must bring a backpack or other bulky bag into the room at all, it must be stowed along the side of the room.

Reading and Coverage for the Last Week (identical to last year)

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. Then:

$N$ accepts $w \iff$ the text’s tableau has an accepting computation $\iff$ some setting of the nondeterministic inputs $y_1, \ldots, y_p$ makes the corresponding Boolean circuit output $w_o = 1 \iff$ that setting determines values for all the other wire variables $t, u, v, \ldots$ that make each NAND gate check out $\iff \phi$ 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 \( \text{NP} \), and \( \text{SAT} \) is in \( \text{NP} \), this means \( \text{SAT} \) is \( \text{NP} \)-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. 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 \( \text{CLIQUE} \) to Vertex Cover and back.

Assignment 10, due in hardcopy and in class Thu. 5/11*
(* except R4 may submit Fri. 5/12 before 2pm to my Davis 326 office)
Please write your name, Student ID#, and recitation attended atop your HW.

(1) Consider the following decision problem:

**Exception Throw**

*Instance:* A Java program \( P \) and an input stream \( x \in \text{ASCII}^* \).
*Question:* Does running \( P \) on \( x \) throw an `ArrayIndexOutOfBoundsException`?

Show that this decision problem is undecidable—by arguing concretely that if it were decidable, then the Acceptance Problem for Turing machines would be decidable, which it isn’t. (It may help you to know that the Turing Kit program, though it has other bugs, never throws this exception. 18 pts.)

(2) Suppose we have a computable function \( f(\langle M, w \rangle) = \langle M' \rangle \) with the following properties for any given TM \( M \) and string \( w \):

- If \( M \) accepts \( w \), then \( L(M') = \Sigma^* \).
- If \( M \) does not accept \( w \), then \( L(M') = \emptyset \).

Explain why this not only mapping-reduces \( A_{TM} \) to \( NE_{TM} \), it also mapping-reduces \( A_{TM} \) to \( ALL_{TM} \) and to \( K_{TM} \). (Note that the latter supplements the simple way \( K_{TM} \) was reduced to \( A_{TM} \) via the simple function \( f'(x) = \langle x, x \rangle \), so that \( K_{TM} \) and \( A_{TM} \) are mapping equivalent—more usually called many-one equivalent and written \( \equiv_m \) either way. 6 + 12 = 18 pts.)

(3) Show that the language \( \text{REGULAR}_{\text{CFG}} = \{ \text{CFGs } G : L(G) \text{ is regular} \} \) is undecidable. Note that this is not the same as deciding whether the rules of \( G \) obey the format of a “regular grammar” as described on HW7. Rather, the question is whether the language of \( G \) is regular. Use the fact that the language of accepting computation histories of a single-tape TM is not regular unless it is empty\(^1\) and do a reduction from \( E_{TM} \). (18 pts., for 54 on the set)

---

\(^1\) Technically, say if there is only one accepting computation \( C \), then the language \( \text{ACH} \) of accepting computation histories would be finite and hence regular. To fix this, we re-define \( \text{ACH} \) to allow configurations to have any number \( m \) of extra trailing @ chars, provided all configurations in the history have the same \( m \). This and the fact that any halting \( C \) has at least two configurations in it makes \( \text{ACH} \) “embed” the language \( \{ @^m \# @^m : m \geq 0 \} \) so that \( \text{ACH} \) is non-regular, unless it is empty.