Lectures and Reading. This week’s lectures are transitioning from computability and undecidability to complexity and \( NP \)-completeness. The Wed. 10/24 lecture about \( P \) and \( NP \) connects the \( A_{DFA} \) and \( A_{NFA} \) examples in section 8 of Debray’s notes and others to examples at the outset of section 14. The rest of Debray’s notes will be covered in parallel with my CRC Handbook chapters written with Eric Allender and Michael Loui (ALR notes). This homework touches on some of the topics from the intervening sections of Debray’s notes, but the coverage is self-contained in the problems—one of which is for extra credit.

For Friday’s lecture, it is important to read up through the proof of the Cook-Levin Theorem in “Ch. 28, section 3” of the ALR notes. Note that the proof is less than one page long. It is based on Boolean circuits drawing on ALR “Ch. 27, sections 3.1–3.3”—in particular, Theorem 3.1 on page 20 and Figure 1 on page 22—but the point can be understood more simply as the principle that “software can be efficiently burned into hardware.” The proof in Debray’s notes replaces circuits by the notion of an “accepting tableau” in a way that mediates between ‘traditional’ proofs based on Turing machines which you may find in other sources (Sipser’s chapter 7 if you have it) and the circuit-based proof (which actually appears in Sipser’s chapter 9). Well, there are further important features of “my” proof—actually first presented by Claus-Peter Schnorr in 1978—which will be brought out next week. For next week, we will cover examples of \( NP \)-complete problems from Debray’s sections 15–16 and ALR ch. 28, the long section 4.

The Second Prelim Exam is tentatively set for Wed. Nov. 14 but there is a possibility of postponing it to Fri. Nov. 16 based on both personal and flow-of-class-material factors.

Assignment 5, due Wed. 10/31 in class

(1) Lectures briefly mentioned the idea of a two-head DFA (2H DFA). I’ve posted some side notes about 2HDFA’s and their relation to Post’s Problem, but the only fact you need to know for this exercise is that given any TM \( M \), we can build a 2H DFA \( H \) such that

\[
L_H = V_M = \{ \langle x, \vec{c} \rangle : T(M, x, \vec{c}) \},
\]

where \( T(M, x, \vec{c}) \) is the predicate from the Monday 10/22 lecture expressing that \( \vec{c} \) is a valid accepting computation of \( M \) on input \( x \). More simply put: 2HDFAs are capable of checking any other machine’s computations.

Show that in consequence, the following decision problem with the standardized name “\( E_{2H DFA} \)” is undecidable:

**Instance:** A 2H DFA \( H \).

**Question:** Is \( L(H) = \emptyset \)?

This is quite a contrast with \( E_{DFA} \) being not only decidable but decidable in “easy” polynomial time by breadth-first search. Also say whether the \( E_{2H DFA} \) language is computably enumerable. (15 + 3 = 18 pts.)
(2) Prove that the following decision problem is undecidable:

**INSTANCE:** A DTM $M$ that is total and a constant $c \geq 1$.

**QUESTION:** Does $M$ run within time $cn^2$?

The cases $x = \epsilon$ and $|x| = 1$ are usually exempted from the requirement that $M(x)$ halt within $c|x|^2$ steps exactly—and you may ignore them—but otherwise it holds strictly. A niggling detail is that if you build a machine $M'$ that runs $M(w)$ for another machine $M$ (say, one with more tapes) then doing $t$ steps of $M(w)$ may take more than $t$ steps by $M'$. But you may word the “Delay Switch” idea to say that you run $M(w)$ for $t = |x|$ steps by the host $M'$. This means $M'$ has to time itself—and this leads to an icky technical detail called “time-constructible functions”—but there is more slack in this particular problem with “$cn^2$” than you may expect and this detail can be effectively ignored. (The upshot of the problem is that there is no general way to estimate the running time of a program, even when you know in advance that it always halts. Note that your reduction will need to guarantee that the machine $M'$ it creates is total, so you can’t simply do “simulate $M(w)$; if and when it halts...” as in the “all-or-nothing switch” style. 18 pts.)

(3) Consider the following decision problem, which might be called “$NE_{DFA}^3$”:

**INSTANCE:** Three DFAs $M_1, M_2, M_3$, each having the same number of states.

**QUESTION:** Is $L(M_1) \cap L(M_2) \cap L(M_3) \neq \emptyset$?

(a) Sketch a polynomial-time algorithm for this problem. Letting $m$ stand for the number of states in the given DFAs (which can vary), state your running time in terms of $m$. Your sketch can say things like “complement the machines” or “do Cartesian product” without giving the details of how things like that are done—we’ve seen how earlier in the course. But you do need to say how things would work in enough detail to give the order of the running time. Often—with reductions too—the time will have the same order as the possible size of an object built from the given components—here, a machine built from $M_1, M_2, M_3$.

(b) Now do the same thing for the problem “$NE_{DFA}^k$” which has a variable number $k$ in place of 3. You are given $k$ DFAs $M_1, M_2, \ldots, M_k$, each with $m$ states, and need to decide whether $L(M_1) \cap L(M_2) \cap \cdots \cap L(M_k) \neq \emptyset$. You may suppose $k = m$ for concreteness. What happens to the running time—is it still polynomial in $m$? (9 + 6 = 15 pts., for 51 regular-credit points on the set)

An extra-credit problem is on the next page
(4) (Extra Credit) Following on from problem (1), now consider $T(M, x, \vec{c})$ to be an actual formalized predicate in some formal system $F$ of logic. Indeed, clauses (1) and (3) of the definition of “$F$ is interesting” on page 34 of Debray’s notes (Definition 11.5) are pretty much equivalent to saying that $F$ can formalize the $T$-predicate in such a way that for every true case, $F$ proves $T(M, x, \vec{c})$, and for every false case (where $\vec{c}$ is bad), $F$ proves $\neg T(M, x, \vec{c})$.

The remaining clause (2) of that definition, with regard to $F$, goes by the name effective or (in older usage) recursively axiomatizable. It says that proofs are checkable the same way computations are: the predicate

$$T_F(S, P) \equiv P$$

is decidable. Note how the following becomes an example of Theorem 10.2 on page 29 of Debray’s notes, which was covered in-tandem with Theorem 13.12 on p43 of those notes.

$S$ is a theorem $\iff (\exists P)T_F(S, P)$.

It follows that the language of theorems of $F$ is c.e. This goes also when the applicable statements $S$ are restricted to be those of a particular form, such as for any given Turing machine $M$:

$$S_M =_{	ext{def}} (\forall x)(\forall \vec{c}) \neg T(M, x, \vec{c}).$$

That is, if we define $PE_T = \{\langle M \rangle : (\exists P)T_F(S_M, P)\}$, then $PE_T$ is c.e. and becomes the set of TMs $M$ such that $F$ can prove $L(M) = \emptyset$—i.e., that have a “provably empty” language. Finally we demand that $F$ is sound—meaning in this case that if $F$ proves the statement $S_M$, then $L(M)$ really is empty. This makes $PE_T$ a subset of $E_T$.

(a) Can $PE_T$ ever equal $E_T$? Say why or why not.
(b) If $\langle M \rangle \in E_T \setminus PE_T$, then what two things can we say about the statement $S_M$?
(c) Conclude that for any formal system that is interesting, effective, and sound, there are true statements that it cannot prove.

Thus you have proved Gödel’s First Incompleteness Theorem, though with a little catch. We haven’t actually given a particular $M$ such that $S_M$ is true but unprovable; we’ve merely shown that such an $M$ must exist. There’s a whiff of paradox here: if you gave such an $M$ then your homework answer would become a proof of $S_M$ after all—but if $F$ is “the formal system of you” then that would be an actual contradiction. This has led some philosophers to conclude that a human being cannot be subsumed by a formal system in any “strong AI” sense.

The paradox is sharpened because if we actually write out the axioms and rules of $F$ and apply the Recursion Theorem to them, then we do get a particular $M$. In a context where statements $S$ were being operated on directly without their being statements about Turing machines, Gödel showed how to do this—this is why his full paper was going to have over 100 pages of numerical details including about his “Gödel Numbers.” But the above is enough to see how the essence is “not c.e.” versus “c.e.”—and to emphasize, this problem does not require reading the parts about the Recursion Theorem. (6+6+3 = 15 pts.)