Lectures and Reading.

First, I am away this coming Friday, Oct. 11. This lecture will be made up on **Tuesday, Oct. 15**, 2–3pm. in Davis 338—which will also serve as Q & A time for this assignment. Then **Prelim I** will be in class period on **Monday, Oct. 21**. It will cover the domain of these first three assignments.

We have crossed over into the second batch of Debray’s notes which were given out in hardcopy, from page 25 onward. I have decided this time to include a lecture related to the Post Correspondence Problem (PCP), but emphasizing first the (non-)emptiness problem for a “two-tape DFA” which starts with its input $x$ on both of its tapes and is allowed to keep one head stationary while the other moves right. Such a machine $M_2$ can decide whether a given sequence of IDs of a (one-tape) Turing machine $M$ represents a valid accepting computation. We can convert $M$ to $M_2$ pretty directly by a code mapping $f$. Then $L(M_2)$ is non-empty if and only if $M$ has a valid accepting computation (on some input string), which is iff $L(M)$ is non-empty. So whereas lecture showed that (non-)emptiness of the language of a 1-tape DFA is decidable, for a 2-tape DFA it is undecidable, since $f$ reduces $NE_{TM}$ to that. Thus please do read the section on PCP but treat it as a “skim.” This will be on the 14th or perhaps the make-up lecture on the 15th. Before then, Monday will cover reductions involving target languages that are neither c.e. nor co-c.e. (synonomous to say: neither r.e. nor co-r.e.; not in $RE \cup co\-RE$), and Wednesday’s lecture will present Rice’s Theorem as an example of reductions—note that it comes after PCP in Debray’s notes. It will also present reductions that are “about” high-level language programs rather than “about” Turing machines.

Before we get there, though, it is important to understand the basic mechanism of a reduction. This goes especially for understanding the reduction from $A_{TM}$ to $NE_{TM}$ which was presented in the second half of the Wed. Oct. 2 lecture, and whose “All-Or-Nothing Switch” property can be re-used to carry out numerous other reductions. I have decided again to recommend the notes


by John Watrous which are also linked from the course webpage. My overall main point is that reductions $A \leq_m B$ have three levels of reasoning:

(a) There is the “outer level” where the reasoning goes: “If there were a Turing machine $S$ that decides $B$, then the reduction would give us a total Turing machine $R$ that decides $A$. $R$ would have the form, ‘On input $x$, run the reduction function to get $y = f(x)$ and run $S$ on $y$. If $S$ accepts, accept. If $S$ rejects, reject.’ But we know $A$ is undecidable, so $R$ cannot exist. Thus $S$ cannot exist. So $B$ is undecidable too.” **Well, please do not ever write this.** The “Reduction Theorem” at the end of the Monday 9/30 lecture takes care of this reasoning once-and-for-all.

(b) The “middle level” is the logical reasoning that establishes the equivalence $x \in A \iff f(x) \in B$, often best pictured as two implications $x \in A \implies f(x) \in B$ and $x \notin A \implies f(x) \notin B$.

(c) The “inner level” is the code editing used to design $f$ which makes the implications come true.

Most sources I find online strike me as blending (a) and (b) together in a way that detracts from (c), but Watrous’s notes not only treat (b) and (c) cleanly but also match other aspects of my lectures well. Hence please also read chapters 15–17 of his notes, focusing on ch. 16. Here are some things to watch for:

- In ch. 15, his $A_{DTM}$ is the same as $A_{TM}$ since a TM defaults to being deterministic. The decidable language $S_{DTM}$ which he gives in equation (15.2) will figure into the two-tape DFA lecture mentioned above, and later into the equivalence of two definitions of the class $NP$. (It
is a variant of the historical predicate and function combo \( T(M, w, \vec{c}) \land U(\vec{c}) = y \)—meaning that \( \vec{c} \) is a string encoding a valid halting (or accepting) computation of the TM \( M \) on input \( w \) and its output is \( y \)—called the Kleene T-predicate after the same Stephen Kleene who proved the theorems on regular expressions and FAs covered earlier.

• Note that Watrous likewise uses the diagonal language as his first example of undecidability, calling it DIAG rather than \( D_{TM} \) or just \( D \). He does not define \( K_{TM} \) as the complement of \( D_{TM} \) but shows the undecidability of \( A_{TM} \) directly, using “\( K \)” to refer to a transformed machine which makes an appearance in ch. 16 too as the edited code for the mapping reduction from \( A_{TM} \) to \( HP_{TM} \) (which he calls HALT).

• Section 15.4 has Theorem 15.7—covered in my lectures as \( \text{RE} \cap \text{co-RE} = \text{REC} \)—and the trick of “hard-coding an input string.” I used that trick earlier to convert the “Universal RAM Simulator” \( U(P, x) = P(x) \) into a Turing machine \( M_P \) which simulates a particular program \( P \) by way of writing \( P \) on its own tape using hard-coded states and then going to the start state of \( U \), so as to make \( M_P(x) = U(P, x) \).

• Note that Watrous devotes time in his Chapter 16, Example 16.6, to the \( A_{TM} \leq_m HP_{TM} \) reduction, which was covered at the end of the Friday Oct. 4 lecture. His example 16.7 follows from that lecture upon realizing the equivalence to the mirror-image reduction of their complements: \( K_{TM} \leq_m \text{NE}_{TM} \). His example 16.8 is an example of re-using the “all-or-nothing switch” reduction. His example 16.9 will be presented this week as a way to build on the “switch” mechanisms.

• The material in sections 17.1–17.3 has mostly been touched on, while the last section 17.4 will be covered in-tandem with Theorem 10.2 from Arun Debray’s notes and the analogous theorem for \( \text{NP} \), which is Theorem 13.12 beginning the last section of Debray’s notes which was also given out last Friday.

So the reading for now in brief is • the newly-given Watrous chapters 15–17, and • the corresponding coverage in sections 8 and 9 of Debray’s notes (but it is rather telegraphic). Skim the Post Correspondence Problem for now and jump ahead to Theorem 10.11 (“Rice’s Theorem”) and its proof. Then skip the Recursion Theorem and sections 11–12; part of the latter is covered in a different way in an extra-credit problem on this assignment. After the coverage of reductions finishes on the 14th or 15th, we will start on complexity theory using section 13 of Debray’s notes and parallel notes by myself and Eric Allender and Michael Loui which will be given out in hardcopy. That material will not be on the first exam—only through this homework and section 10 of Debray’s notes.

Assignment 3, due Wed. 10/16 in class

(1) Work out in full the converse reduction from \( HP_{TM} \) to \( A_{TM} \). (This is the one the Fri. 10/4 lecture did not do.) Please use a reduction diagram similar to the lecture. (12 pts.)

(2) The feature of allowing a Turing machine to keep its head stationary (\( S \) for “stay” in an instruction) is a useful convenience but not necessary: An instruction \( (p, c, d, S, q) \) could always be replaced by making a special state \( p' \) and giving the instructions \( (p, c, d, R, p'), (p', -, -, L, q) \) instead, where the ‘\( - \)’ is a “wildcard”—or rather, is shorthand for actually having an instruction \( (p', a, a, L, q) \) for every \( a \in \Gamma \). Executing an instruction with “stay” is hence an elective rather than essential capability of programs. Now consider the following languages:
(a) $L_a = \{\langle M, w \rangle : M(w) \text{ executes a stay instruction}\}$.

(b) $L_b = \{\langle M \rangle : \text{ for some string } w, \ M(w) \text{ executes a stay instruction}\}$.

(c) $L_c = \{\langle M \rangle : \text{ for all strings } w, \ M(w) \text{ executes a stay instruction}\}$.

(d) $L_d = \{\langle M \rangle : \text{ for all strings } w, \ M(w) \text{ never executes a stay instruction}\}$.

(e) $L_e = \{\langle M \rangle : \text{ for all strings } w, \ M(w) \text{ can never execute a stay instruction, for the simple reason that it has no stay instructions in its code}\}$.

Classify each language according to whether it is (i) decidable, (ii) c.e. but not decidable, (iii) co-c.e. but not c.e., or (iv) neither c.e. nor co-c.e. Once you give a reduction for the first one or two undecidable cases, you should be able to handle the others with minimal further effort, making use of a close analogy with the $A_{TM}$ and/or $E_{TM}$ problems which your first reduction(s) should establish. (Note: The answer to $L_d$ can be “yes” even if the computation $M(w)$ never halts, so long as it never executes an instruction with $S$, $6+9+6+9+6 = 36$ pts.)

(3) Consider the following decision problem:

**INSTANCE:** A deterministic Turing machine $M = (Q, \Sigma, \Gamma, \delta, \square, s, q_{acc}, q_{rej})$.

**QUESTION:** Is there an input $x$ that causes $M$ to visit every one of its states at least once before halting?

Prove by reduction from $A_{TM}$ that this problem is undecidable. (18 pts. **Hint:** You are allowed to make $\Gamma$ bigger in your $M'$ than it is in the given $M$. But for 6 pts. extra credit, explain how you could adapt the reduction in case all TMs were limited to the characters 0, 1, and the blank—ideas from problem (2) might come in handy.)

(4) Given a language $L$, define $L^R = \{x^R : x \in L\}$, where $x^R$ means reversing the string $x$. Note that if every string in $L$ is a palindrome, then $L = L^R$ automatically, but it is possible to have $L = L^R$ without every string in it being a palindrome, e.g., $L = \{01, 10\}$. Prove that the language of the following decision problem is neither c.e. nor co-c.e.

**INSTANCE:** A deterministic Turing machine $M$.

**QUESTION:** Is $L(M) = L(M)^R$?

**Hint:** This problem, too, has a high “re-usability” quotient—it is AOK to almost violate the University policy against re-submitting previously-used materials, provided you explain how and why the reduction(s) apply to this case. (24 pts., for 90 on the set)