Lectures and Reading. I have decided to prioritize the bridge from reductions at the level of \( \text{RE} \) and above to reductions for \( \text{NP} \)-hardness. The former are IMHO a weakness of Sipser’s book and it carries through in the scribe notes by Debray. 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” in the Monday 10/8 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 \( x \not\in A \implies f(x) \not\in B \).

(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). I was happy to find that the notes by John Watrous for Waterloo’s version of CSE396, while less suitable for this course in general, not only treat (b) and (c) cleanly but also match other aspects of my lectures well. Hence for the upcoming week please read chapters 15–17 of his notes as given out in class on Friday, focusing on ch. 16. Here are some things to watch for:

- In ch. 15, his \( A_{\text{DTM}} \) is the same as \( A_{\text{TM}} \) since a TM defaults to being deterministic. The decidable language \( S_{\text{DTM}} \) which he gives in equation (15.2) will figure into the coverage of logic later on. (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 strong 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 uses the diagonal language as his first example of undecidability, calling it \( \text{DIAG} \) rather than \( D_{\text{TM}} \) or just \( D \). He does not define \( K_{\text{TM}} \) as the complement of \( D_{\text{TM}} \) but shows the undecidability of \( A_{\text{TM}} \) 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 \( A_{\text{TM}} \) to \( \text{HALT} \).

- 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 \( M_P(x) = U(P, x) \).

- In Chapter 16, my lectures essentially covered Example 16.6. I will cover Examples 16.7–16.9 in a similar way but with slightly different details using a particular reduction from \( A_{\text{TM}} \) to \( \text{ALL}_{\text{TM}} \) called the “all-or-nothing switch” as a focal point. Rice’s Theorem will be treated as a further example of this.

- 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.
We will probably not reach P and NP this coming week. Regarding the Allender-Loui-Regan “Chapter 27” which was given out last month, some of the “canonical classes” in section 2.1 were diagrammed in lecture already. That section also skims over the same characterization of NP as in Theorem 13.2 from Debray’s notes. Section 2.5 was touched on alongside proving that the class of decidable languages is closed under complementation. We will jump from there to sections 3.1–3.3 in order to set up the machinery for the main proof about NP-completeness. This is in sections 1–4 of the ALR “Chapter 28” notes given out last Friday, which parallel Debray’s sections 14-15-16. That will carry us into November. Besides a one-sheet printout of some 2-tape TMs, the remaining major set of notes for the term given out on Friday was the first half of my quantum algorithms textbook with Richard Lipton—I will specify sections of that to read when the time comes.

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. Going into next week, • pick up Debray in section 13 and • read the parts of the ALR chapters 27 and 28 outlined above.

Assignment 4, due Wed. 10/24 in class

(1) Prove by a mapping reduction from $A_{TM}$ that the (language of the) following decision problem is undecidable:

**INSTANCE:** A deterministic Turing machine $M$ with 2 tapes, and an input $x$ to $M$.

**QUESTION:** Does $M(x)$ eventually write a non-blank character on Tape 2?

Also explain why the language of this problem is computably enumerable. (18 + 3 = 21 pts.)

(2) Prove by mapping reductions that the language of the following decision problem is neither c.e. nor co-c.e. You may use any suitable problems as sources of your reductions, though the basic choices $A_1 = A_{TM}$ (or $K_{TM}$) and $A_2 = D_{TM}$ can always be made to work.

**INSTANCE:** A deterministic one-tape Turing machine $M$ with alphabet $\Sigma = \{0, 1\}$.

**QUESTION:** Is $L(M) = 0^*$, i.e., does $M$ accept exactly the binary strings without a ‘1’ in them?

(Note that quoting Rice’s Theorem would only give you undecidability. $2 \times 18 = 36$ pts.)

(3) We change a condition from the statement of problem (1). The **INSTANCE** is the same: a deterministic Turing machine $M$ with 2 tapes and an input $x$ to $M$. The new **QUESTION**, however, is: Does $M(x)$ eventually change a character on either tape?

Describe in prose an algorithm to decide this problem. Note that if $M(x)$ gets into an infinite loop, but you can tell the loop will never change any of the cells it loops in, then your algorithm should answer no. Also estimate the running time of your algorithm given $M$ with $m$ states and an input $x$ of length $n$. For intuition you may suppose that $m$ and $n$ are roughly equal but you should give a running time bound in terms of both $m$ and $n$. You may ignore factors of log $m$ and/or log $n$ which may arise from using the binary number labels $1, \ldots, m$ of the states and/or the indices $i = 1, \ldots, n$ of the bits of $x$—there is a special notation $\tilde{O}(N)$ which means doing just that. (18 pts. for the algorithm and 6 for the running time, making 24 points on the problem and 81 on the set.)