Schedule: The Final Exam is on Tuesday, May 10, 11:45am–2:45pm in Hochstetter 114. It will be open-notes but closed-book with no electronics. It will cover all of the course material, Chapters 1–5 and 7, skipping certain parts as noted during the term: the regular-languages Pumping Lemma, the text’s specialized notation for PDAs in section 2.2, section 2.4, and section 5.2. Regarding section 2.2, you should know PDAs as defined via special cases of two-tape Turing machines and the statement in Theorem 2.20 that a language $L$ is a CFL if and only if there is a (nondeterministic) PDA $N$ such that $L(N) = L$. The exact coverage from Chapter 7—possibly including also parts of section 9.3 replacing the proof of the Cook-Levin Theorem given in section 7.4—will be determined after the last lecture of term. There will be one more problem set, due on the last day of class, which is Thursday, May 5. I intend to have an optional pre-exam “Review Session” on the afternoon of Monday, May 9—I will try for 2–4pm that day.

Reading: First, look at section 2.4 only to get the definition of a “deterministic context-free language” (DCFL) as the language of a DPDA. (If you have the 2nd. ed. just take this definition at face value.) The further theorem that the class of DCFLs is closed under complementation already falls under the umbrella of the way this was proved for deterministic Turing machines, as I will remind in lecture: that is, for every DPDA $M$ we can build a DPDA $M'$ such that $L(M') = \sim L(M)$. Technically this requires coding DPDA s so that they always halt but we can take this for granted. It follows that any CFL $L$ whose complement $\sim L$ is not a CFL is not a DCFL. A basic example to keep in mind is:

- $L = \{a^ib^jc^k : i = j \land j = k\}$ is not a CFL.

- $A = \{a^ib^jc^k : i = j\}$ and $B = \{a^ib^jc^k : j = k\}$ are both CFLs and indeed they are DCFLs—DPDAs accepting them just have to count two of the letters, not keep track of all three.

- Thus the intersection of two CFLs—indeed two DCFLs—need not be a CFL. (However, the intersection of a DCFL or CFL with a regular language remains, respectively, a DCFL or CFL.)

- This contrasts with CFLs being closed under unions—and tells you right away that CFLs are not closed under complements.

- Indeed, $A' = \{a^ib^jc^k : i \neq j\}$ and $B' = \{a^ib^jc^k : j \neq k\}$ are likewise DCFLs, and $R = \sim a^*b^*c^*$ is regular since $a^*b^*c^*$ is a regular expression and the class REG of regular languages is closed under complements. Since all regular languages are CFLs—indeed DCFLs since a DFA “Is-A” DPDA that ignores Tape 2—$R$ is a CFL too.

- So $L' = A' \cup B' \cup R$ is a CFL. It equals the complement of $L$, so $L'$ is not a DCFL—which finally tells you that the union of DCFLs need not be a DCFL though it is always a CFL.

- And again, the intersection of two DCFLs need not even be a CFL.

With this taken care of—partly by the Thu. 4/19 lecture—move on and read Chapter 5 for next Tuesday. In section 5.1, note that the Tue. 4/12 lecture already defined accepting and rejecting
computation histories. I used the letters $I_j$ for “ID” (rather than $C_i$ for “configuration”) and “$t$” rather than “$l$” for the length of the computation because “$t$” also defines the time taken. But right there you should instead jump ahead to section 5.3 which has the focal definition of the entire chapter: mapping reducibility. Here it helps that the same 4/12 lecture also already defined computable function. Note how the text then re-casts the earlier examples of section 5.1 as mapping reductions. In lectures I try to minimize the confusion of double-nesting quotations in things like Example 5.24 for $HALT_{TM}$ by drawing reductions as “transformations of machine descriptions” like the text says in example 5.19. Reductions are the last main topic of term that causes difficulty but are a fundamental idea. They have both positive and negative purpose—that is, they are used both to help prove that some problems are decidable (or when the reductions are polynomial-time bounded like later in Chapter 7, prove that some problems belong to P) and to prove some are undecidable (or the following week in Chapter 7, prove them NP-hard or NP-complete). Indeed, several of the text’s algorithms in Chapter 4 are really “reductions to $E_{DFA}$” and to related problems.

**Assignment 9**, due in hardcopy; please staple any multiple-sheet submission

1. Give a decision procedure for the following computational problem:

   **INSTANCE**: A DFA $M$.

   **QUESTION**: Does $L(M)$ equal its reversal?

   Note that the Thu. 3/3 lecture led off with how to convert a DFA $M$ into an NFA $N$ such that $L(N) = L(M)^R$. You do not need to repeat fine details of this conversion but can use it as a line of your procedure the way the text does in section 4.1. Ultimately your algorithm should produce a DFA $M'$ such that deciding the $E_{DFA}$ problem on $M'$ tells you the answer about $M$—it will flow into the text’s Theorem 4.5 but you should show all the steps. (12 pts.)

2. Take the algorithm for determining the set $N$ of variables that derive $\epsilon$ (called NULLABLE in the Chomsky NF notes on the course webpage) and initialize $N = \Sigma$ instead of $N = \emptyset$. Explain in your own words your understanding of how this now gives you the procedure in Theorem 4.8 for deciding $E_{CFG}$. Then apply it to solve the text’s problem 4.28 on page 212 (in the second edition, it is 4.26 on page 184): Sketch in prose a decision procedure for the following problem:

   **INSTANCE**: A CFG $G = (V, \Sigma, R, S)$, and a string $x \in \Sigma^*$.

   **QUESTION**: Does $G$ generate some string $w \in \Sigma^*$, such that $x$ is a substring of $w$?

   You may use the fact that since $x$ is fixed, the language $\Sigma^*x\Sigma^*$ is regular. Then you may use as a step in your sketch the fact that given $G$ and any regular set $R$, an algorithm can build a context-free grammar $G_R$ such that $L(G_R) = L(G) \cap R$. The text finally hints about appealing to the decision procedure for $E_{CFG}$. (9 + 18 = 27 pts.)

3. Consider the following decision problem:

   **INSTANCE**: A deterministic one-tape Turing machine $M$ and an input $x$ to $M$.

   **QUESTION**: Does $M$ on input $x$ at some point overwrite a non-blank char by the blank?

   Explain what lectures have already said about the ability to code one-tape TMs so they only write blanks over non-blank chars during a final “good housekeeping” stage before accepting, and explain why deciding this problem would achieve the impossible by deciding $A_{TM}$. (You need not use a “reduction” per-se. 15 pts., for 54 on the set.)