The **First Prelim Exam** will be held in class period on **Tuesday, April 19**. It will have the same rules as Prelim I. It will cover up to chapter 3, focusing mostly on sections 2.1 and 2.3 and the two sets of auxiliary notes on induction proofs and Chomsky normal form.

**Reading:** Tuesday’s lecture will finish the remaining parts of Chapter 3: formal definition of computations and \( L(M) \) for a Turing machine \( M \), equivalence to other models, the Church-Turing Thesis specifically, and how TMs relate to programming. The following terms are *all synonyms*:

- Turing-recognizable (this is our text’s term, but some other sources equate “recognizable” with “decidable”)
- Turing-acceptable (no ambiguity with any sources)
- recursively enumerable (r.e.) (this is the historical term owing to “Enumerators”—I will skim/skip Theorem 3.21)
- computably enumerable (c.e.) (the standard term nowadays).

Also historically, “recursive” is a synonym for the text’s “decidable” which I will use. Nevertheless, the historical terms persist in the names \( \text{RE} \) and \( \text{REC} \) for the classes of c.e. and decidable languages, respectively, to go with \( \text{CFL} \) for the class of CFLs and \( \text{REG} \) for the class of regular languages. Tuesday’s lecture will end in section 4.1. Chapter 4 will not be covered on Prelim II, but the Thursday 4/14 lecture will come to an important crunch time in section 4.2. The crunch is Theorem 4.11 that the language “\( A_{TM} \)” is undecidable—whose proof appears a few pages later after a digression about uncountability that you may *skim*. What I do is break up the proof by pulling out the “\( D(\langle M \rangle) \)” part as being about the *language*

\[
D_{TM} = \{ \langle M \rangle : M \text{ does not accept } \langle M \rangle \}.
\]

The end of the proof really shows that \( D_{TM} \) is not even Turing-recognizable, let alone decidable. Then I will go on to show the conclusions for \( A_{TM} \) and its complement (Corollary 4.23). Here’s the nub about \( D_{TM} \) in brief but full: Suppose \( D_{TM} \) were Turing-recognizable. Then there would be a TM \( Q \) such that \( L(Q) = D_{TM} \). This \( Q \) has a code \( \langle Q \rangle \). What does \( Q \) do on input \( \langle Q \rangle \)? If \( Q \) accepts \( \langle Q \rangle \), then \( \langle Q \rangle \in D_{TM} \)—since \( L(Q) \) is supposed to equal \( D_{TM} \)—but that means \( Q \) does not accept \( \langle Q \rangle \). But if \( Q \) does not accept \( \langle Q \rangle \), then \( \langle Q \rangle \in D_{TM} \), so \( Q \) must accept it. This Quixotic Quandary contradicts \( L(Q) = D_{TM} \); so no such \( Q \) can exist, which means the “diagonal language” \( D_{TM} \) is not Turing-recognizable. The rest of the text’s proof then amounts to saying that if \( A_{TM} \) were decidable then so would be \( D_{TM} \), but the latter is not even recognizable, so \( A_{TM} \) is undecidable.

1. Consider the following three languages over the alphabet \( \Sigma = \{a, b, c, d\} \), where by default \( i, j, k, \ell \) are non-negative integers (can be 0):

\[
\begin{align*}
L_1 &= \{ a^i b^j c^k d^\ell : i < j \land j < k \land \ell \} \\
L_2 &= \{ a^i b^j c^k d^\ell : i < k \land j \land \ell \} \\
L_3 &= \{ a^i b^j c^k d^\ell : i < \ell \land j < k \}.
\end{align*}
\]
One of these is not a CFL; the other two are CFLs. Give context-free grammars for the
two that are CFLs, and a CFL Pumping Lemma proof for the one that is not a CFL. (You
need not prove your grammars correct, but their plan should be clear. 6+6+18 = 30 pts.)

(2) Let $E$ be the language of \textit{nonempty} strings of balanced parentheses. First show that
the following CFG $G$ is \textit{not} comprehensive for $E$ (that it is sound is pretty immediate so you
need not prove it).

\[ S \rightarrow (S)S \mid () \]
Then add one or two more rules (that is, one or two more right-hand sides for $S$) to make
a grammar $G'$ that is comprehensive; presuming your rules are sound, you’ll get $L(G') = E$
exactly. Then prove your answer in one of two ways:

- Prove $E \subseteq L(G')$ by induction on strings (again we’ll regard the $L(G') \subseteq E$ part as
  granted—so long as you made it true).

- Consider the grammar $G''$ given by $S \rightarrow (S)S \mid \epsilon$. It is technically \textit{unsound}
because it derives $\epsilon$, but it is well-known to be comprehensive. Then do some of the conversion of
$G''$ to Chomsky normal form—how might this confirm your $G'$ (or even give it to begin with)?

The latter option is quicker and makes this $3+6+9 = 18$ pts. total, but the former will still
be shown on the answer key.

(3) Design either a one-tape TM or a two-tape DTM $M$ such that
$L(M) = \{ a^ib^j : i < j, i \geq 0 \}$. If you do the latter—which not only runs more quickly but also has cleaner
code IMHO—make it obey the condition of being a deterministic pushdown automaton: no
character changes or left-moves on the input tape, and any left-move on the second tape must
write the blank. A well-commented arc-node diagram is expected—not (just) a pseudocode
strategy.\footnote{My take on the last part of chapter 3 is that the arc-node details are important to understand that-and-
how a TM can simulate “other models” (most in particular, a mini-assembler/“random-access machine” code),
but once you do—next week—then we can dispense with them.} It is OK to use the text’s diagram style, but especially if you choose to do the
2-tape TM (which a DPDA “Is-A”) you may find my “stacked instruction labels” have a
more-uniform look-and-feel. (15 pts., for 63 total on the set)