Exam Time: 75 minutes. Each problem is worth 10 points. 50 pts total. Choose any 5 problems. Specify which 5 problems you want graded.

1. Context-Free Grammars. Determine whether each of the languages $L_1 = \{a^i b^j c^i d^j \mid i, j \geq 0\}$ and $L_2 = \{a^i b^j c^i d^j \mid i, j \geq 0\}$ is context-free and present proof of your answer.

**Soln.**:
- a) $L_1$ is CF and you can easily construct a CF Grammar for it. Start with $S \rightarrow AB$ and go on from there.
- b) $L_2$ is not CF, and you can use a Pumping Lemma. In particular, let $K > 0$ be the integer guaranteed by the assumption that $L_2$ is CF. Let $w \in L_2$ with $w = a^K b^K c^K d^K$. Then $w = uvxyz$, where the decomposition satisfies the conditions or Lemma 3.43, p. 143. The length restriction $|vxy| \leq K$ guarantees that $vxy$ does not contain more than two symbols. Go on from there.

2. PDAs and CFGs. Following the procedure in the text, construct a PDA that accepts the language of arithmetical expressions ($E$ is the start symbol):

$$
E \rightarrow E + T \mid E - T \mid T,
$$

$$
T \rightarrow T \ast F \mid T \div F \mid F,
$$

$$
F \rightarrow (E) \mid s \mid b.
$$

**Soln.**: Theorem 3.35 and Example 3.36 (pp. 122-123) give the procedure and details of implementation for a similar problem.
3. Turing Machines. Define Deterministic Turing Machine and sketch the construction of a Deterministic one-tape Turing Machine that decides the language \( \{ w \in \{a, b\}^* \mid \#_a(w) > \#_b(w) \} \), where \( \#_a(w) \) denotes the number of occurrences of the letter \( a \) in the string \( w \).

Soln.:

a) See p.160, starting from the “quintuple” at the bottom, and describe its various parts.

b) i) Start from the \( B \) at the right of the input \( w \), and keep moving left, looking for an \( a \).

ii) If you reach a \( B \), fail; else, if you reach an \( a \), change it to \( a' \) and continue scanning to the left \( B \).

iii) When you reach the left \( B \), start moving right, looking for a \( b \).

iv) If you reach the right \( B \), halt and accept (you have at least one more \( a \) than \( b \)); else change the \( b \) to \( b' \) and continue scanning to the right until the \( B \); go to i) above.

v) If you fail in ii), move to the right \( B \), scan back changing every character to a \( B \), until you reach the left \( B \), move right leaving the \( B \), write a \( 0 \) and move right into the halt state.

vi) If you succeed in iv) do the same as in v) above, writing a \( 1 \) instead.

Note: you must indicate how you deal with both success and failure: the TM decides.

4. Turing Machines. Construct a multi-tape DTM to accept the language

\[ \{ w \in \{a, b\}^* \mid \#_a(w) = \#_b(w) \} \]

Discuss how much time your machine saves over a one-tape DTM using the same algorithm.

Soln.:

a) Use a two-working-tape machine.

i) Move the working-tape heads one place to the right (so you leave a \( B \) at the beginning). The input head scans left from the \( B \) at the end of the input (if you don’t like this, move it all the way to the left \( B \) and scan going right); if it finds an \( a \), the tape-1 head writes an \( a \) and moves right; if it finds a \( b \), the tape-2 writes a \( b \) and moves right.

ii) When the input head reaches the right \( B \), the tape-1 and tape-2 heads start moving left simultaneously. As long as both characters are \( a \) and \( b \) respectively, move left.

iii) If both characters are \( B \), halt and accept.

iv) If one of the characters is \( B \), while the other is not, fail. This latter stage can be skipped, since the machine would then hang - and we have specified only an acceptor.

b) The cost of this algorithm is \( O(n) \), since we just scan the input, going left, and then scan at most half the input (going left again) before accepting or hanging (or rejecting). A single-tape machine would have to use a slight modification of algorithm of Problem 3 above (if, in ii) you find a \( B \) without finding an \( a \), you don’t fail - scan to the right looking for a \( b \) and fail if you find one) doing a full scan for each character: \( O(n^2) \).
5. **Recursive Functions.** Define *Primitive Recursive*. Show that \( \text{factorial}(n) = n! \) is Primitive Recursive. You may need to claim primitive recursiveness for some intermediate functions: use the fewest such and be explicit.

**Soln.:**

a) See p. 200 and p. 201. The pattern for *Primitive Recursion* is critical is you want to define a function recursively in terms of itself. There is no other way...

b)

i) \( \text{fact}(0) = \sigma(\zeta(n)) \),

ii) \( \text{fact}(n + 1) = h(n, \text{fact}(n)) \), and we have the problem of constructing a function \( h \) that is Primitive Recursive. We know that \( \sigma \) (the successor function) is Primitive Recursive (by definition), \( \text{mult} \) is Primitive Recursive (proven through Examples 4.21 and 4.22), and \( \pi_1^2 \) and \( \pi_2^2 \) are Primitive Recursive (by definition). Since \( o \) (composition) is also Primitive Recursive (by definition), we define \( h(p, q) = \text{mult}(\sigma \circ \pi_1^2, \pi_2^2)(p, q) \), giving us

\[
\text{fact}(n + 1) = h(n, \text{fact}(n)) = \text{mult}(\sigma \circ \pi_1^2, \pi_2^2)(n, \text{fact}(n)).
\]

6. **R.E Sets and Recursive Sets.** Prove: if \( A \) and \( B \) are recursive sets, then \( A \cup B, A \cap B \) and \( \overline{A} \) are all recursive. What can you say if we only assume them to be r.e.?

**Soln.:**

a) Since \( A \) and \( B \) are recursive, they have total (recursive) characteristic functions, say \( \chi_A \) and \( \chi_B \). We also know that \( \text{and}, \text{or} \) and \( \text{neg} \) are Primitive Recursive (and hence recursive). We can define \( \chi_{A \cup B} = \text{or}(\chi_A, \chi_B) \), \( \chi_{A \cap B} = \text{and}(\chi_A, \chi_B) \) and \( \chi_{\overline{A}} = \text{neg}(\chi_A) \). Thus all three sets are recursive.

b) We can start from the fact that \( A \) and \( B \) possess semi-characteristic functions and argue from there to conclude that \( A \cup B \) and \( A \cap B \) are r.e., or use the Projection Theorem (in either way, you use the recursiveness of \( \text{and} \) and \( \text{or} \)). We know that there are r.e. sets whose complements are not r.e., and so we cannot say anything about \( A \). I did not ask for proofs of this last part - just awareness.
7. **Undecidability and Ambiguity.** State the *Post Correspondence Problem* and use it to show that the problem of determining whether a Context-Free Grammar is ambiguous is undecidable. You must construct a many-to-one reduction (define the term) from PCP to the set AMB of ambiguous CFGs.

**Soln.:** See p. 272 for a description of the *Post Correspondence Problem*; Example 5.47 (p. 276) for the undecidability of CF Grammars. This material is also in the class notes.