1) Show a parse tree for Anderson's coordinate grammar applied to this input expression. The grammar rules are in the course reader on pages 217-220, with discussion on pages 198-216. Show your answer as a tree. The root of the parse tree is labeled EXPRESSION; it matches the entire set of symbols. At each node in the tree, write down the number of the production that you use to expand the parent node into child nodes. **Write down the “m” attribute value at each place in the tree where it changes;** the value of “m” at the root of the tree is the output produced by this computation.

Begin by drawing the parse tree. This input has implied multiplication -- there is no explicit symbol such as “*” in the input, to indicate multiplication. So use Anderson’s rules for implied multiplication; avoid rules such as A7 or A9 that only apply to explicit multiplication. Once you have drawn the whole tree, then assign m values, starting at the leaves and working toward the root.

Anderson's parser builds the tree in a top-down manner, with a lot of search and backtracking to find the correct set of rules to apply. You can use whatever method allows you to most quickly identify the correct set of production rules to use. (Don't show any backtracking, just show the final tree.) It is fine to draw the tree by hand or format by computer, whatever you find easiest. As was done in lecture, you can use a short-hand notation for the derivation sequence:

```
EXPRESSION (A5) TERM (A10) ADJTERM (A22) DIVTERM (A25) FACTOR (A32) VARIABLE (A55) ALPHA (A56) LETTER
```

**Question:** What is the scope of the summation? Does the Anderson grammar interpret this input as a summation over the expression "(n * root n)" or does it interpret this expression as a summation over "n", with the result multiplied by root n?

2) Write a PDL expression (page 181 of the course reader) to describe the following figure. Use primitives consisting of circles and straight line segments. Document your choice of primitives (including the “head” and “tail” location of each primitive).

In this figure the line segment for the dog-tail is the same as the line-segment for the leftmost person-arm, and the dog-neck is the same as the other person-arm. The “leash” consists of four person-arm line-segments concatenated together. Clearly show how you define one or more circle primitives to get the correct attachment points for the person-head and dog-head.

3) Page 230 of the course reader shows two plex grammars for describing flow charts: one for flowcharts with no loops, and one for flowcharts with loops. The primitives are START, FUNCT, HALT, and PRED. FUNCT is a function, with connection 1 representing the call of the function, and connection 2 the exit. PRED (predicate) is an IF statement, with connection 1 used to reach the IF; control-flow exits via connection 2 if the predicate succeeds and via 3 if it fails. Here is a copy of the plex grammar for loop-containing flow charts:

```
1. <PROG> → <START> <P> <HALT> (110, 021)
2. <P> (1, 2) → <FUNCT> ( ) (1,2)
3. <P> (1, 2) → <FUNCT> <P> (21) (10,02)
4. <P> (1, 2) → <PRED> <P> (21,12) (12, 30) [Last part is corrected; Fu mistakenly says (21, 30)]
5. <P> (1, 2) → <PRED> <P> <P> (210,301,022) (100, 022)
```

For flowcharts (a) and (b) below, determine if the plex grammar can generate this flowchart. If so, show the derivation. If not, show an additional production rule that would permit derivation of the flowchart.
4) Here is a stochastic grammar.

\[
\begin{align*}
\langle S \rangle & ::= \langle \text{HELLO} \rangle \langle \text{ASK} \rangle \langle \text{ENDING} \rangle \quad \text{Prob.} = 1.0 \\
\langle \text{HELLO} \rangle & ::= \text{Hi} \quad \text{Prob.} = 0.3 \\
\langle \text{HELLO} \rangle & ::= \text{Hello} \quad \text{Prob.} = 0.6 \\
\langle \text{HELLO} \rangle & ::= \text{Howdy} \quad \text{Prob.} = 0.1 \\
\langle \text{ASK} \rangle & ::= \text{How are you?} \quad \text{Prob.} = 0.7 \\
\langle \text{ASK} \rangle & ::= \text{What’s up?} \quad \text{Prob.} = 0.1 \\
\langle \text{ASK} \rangle & ::= \text{What’s happenin’?} \quad \text{Prob.} = 0.1 \\
\langle \text{ASK} \rangle & ::= \langle \text{ASK} \rangle \langle \text{ASK} \rangle \quad \text{Prob.} = 0.1 \\
\langle \text{ENDING} \rangle & ::= \text{Eh?} \quad \text{Prob.} = 0.3 \\
\langle \text{ENDING} \rangle & ::= \text{--silence---} \quad \text{Prob.} = 0.7
\end{align*}
\]

a) Give two different sentences that belong to the language generated by this grammar. Give both the sentence itself, and the probability with which the sentence belongs to the language.

b) Is this a consistent grammar?

Consistency of a stochastic context free grammar is defined in page 251 of the course reader. Informally, a grammar is consistent if the probabilities of all strings in the language sum to 1. A grammar can be inconsistent due to infinitely-long derivations that have non-zero probability. For example, the following grammar is inconsistent:

\[
\begin{align*}
\langle S \rangle & \rightarrow \langle A \rangle \quad \text{Prob.} = 0.3 \\
\langle S \rangle & \rightarrow \langle x \rangle \quad \text{Prob.} = 0.7 \\
\langle A \rangle & \rightarrow \langle A \rangle \quad \text{Prob.} = 1.0
\end{align*}
\]

This grammar uses nonterminal symbols \( S \) and \( A \), and terminal symbol \( x \). The language generated by this grammar contains only one string (“\( x \)”), and that string has probability equal to 0.7.

The purpose of this question is to make you aware of the definition of consistency. Don’t spend a lot of time on this question, unless you want to: it’s easy to make mistakes in analyzing this. I typically get some students writing convincing-looking proofs that this grammar IS consistent, and others writing convincing-looking proofs that the grammar is NOT consistent.

5) [This question gives you practice in using grammars to describe two-dimensional shapes. You don’t have to spend a lot of time on this; if you are not sure how to proceed, write down some thoughts, and describe problems you encounter.] A quadtree is a multiscale image representation. In a quadtree, a uniform image region is represented by a large square showing the average gray level in that area. Image areas that contain a lot of variations are represented by smaller squares.

Example of a quadtree subdivision:

Write a grammar whose language consists of the set of all possible quadtree subdivisions. You can choose whether to use a string grammar, tree grammar, set grammar or graph grammar. Describe how you map a quadtree to a string or tree or set or graph. Also describe any labels or attributes that are associated with the string or tree or set or graph. Your grammar should prevent repeated subdivisions of the same square: subdivide each square at most once. (Node or edge labels can be used to ensure that the subdivision-production is not applied repeatedly to the same square.) Use attributes to describe the physical dimensions of the subdivisions.
6) Review problem for Bayes classifier. Similar to problem 8 on assignment 1.
Consider a two-class, single-feature problem where \( p(x \mid \omega_1) \) is uniform in the range \([1,3]\) and \( p(x \mid \omega_2) \) is uniform in the range \([2,5]\). For each of the priors (a) and (b), state the Bayes decision rule ("choose \( \omega_1 \) when \( x \) is <something>"), and state the error rate. For the error rate, state a specific value, like 35%, not an expression or integral.

(a) \( P(\omega_1) = 1/2 \) (in this case, \( P(\omega_2) = 1/2 \))
(b) \( P(\omega_1) = 1/4 \) (in this case \( P(\omega_2) = 3/4 \))

——— Review Information for Handling Uncertainty and Error in Structural PR Systems ———
For the final exam, you should be able to briefly describe the following approaches to incorporating errors and uncertainties into structural pattern recognition systems. These approaches are summarized in the course reader pp. 238–239, with a review given below. (You don’t have to hand anything in for this part of the assignment.)

- error correcting parsing
- transformational grammars to map noisy patterns to less-noisy patterns
- stochastic grammars
- fuzzy grammars

**Error correcting parsing** (for string grammars) is based on a distance measure between strings. For strings that belong to the grammar under consideration, parsing proceeds as usual. For strings that don’t belong to the grammar, standard parsing simply gives up. Error-correcting parsing, on the other hand, finds a parse to the string-in-the-language that is closest to the given string. This can be implemented by adding error productions to the language (to model the various distortions/errors that can occur) and then modifying the parser to use as few of these error productions as possible. Error-correcting parsing tends to be expensive, requiring a lot of computation time during the parse.

**Production rules that transform a noisy pattern into a less-noisy pattern** are illustrated by the transformational grammar on page 188 of the course reader. Another example is the circuit diagram recognition system by Bunke (reference is on page 238 of the course reader): here the lines in the circuit diagram are converted into a graph. Then “error correcting” graph productions are applied to transform the noisy input into a less-noisy input: these graph productions are designed to close small gaps in lines, and to trim off lines that overhang a bit (common situations in hand drawn circuit diagrams).

In this approach, the system author has to write productions to deal with particular, anticipated types of errors. This type of error correction can be effective and efficient, but it only works for errors that the system designer anticipates and explicitly codes for. It also assumes that the error-correcting production rule can correct the problem properly. There is no room for expressing uncertainty, such as “There is a gap in this line. Maybe this gap is intentional, or maybe it is the result of noise.” The error-correcting production tests some criteria (the distance across the gap should be fairly small, and the stroke direction across the gap should be consistent); if the criteria are met, the gap is closed. Otherwise it stays open.

**Stochastic grammars** use probabilities to model noisy and distorted patterns. Each production rule has an associated probability. The probability of a derivation is computed by multiplying all the probabilities of the productions used during the derivation. The language defined by the grammar consists of strings (or trees or graphs), each with an associated probability of belonging to the language. The probability values can be used to resolve ambiguity. Derivation probabilities are monotone decreasing as production rules are applied. Fuzzy grammars get around this restriction.

**Fuzzy grammars** are a variant of stochastic grammars. They use "fuzzy membership" instead of probabilities. Fuzzy memberships, like probabilities, are drawn from the range \([0, 1]\), but they don't have constraints about "summing to one". In a fuzzy grammar, each production rule has a fuzzy membership in the set of productions. The fuzziness value of a derivation as a whole is computed by combining the fuzziness values of the productions used during the derivation. The example in the course reader uses two methods for performing this combination: page 259 uses the MIN of the fuzzy values, whereas page 260 uses a ratio of summed \( g \) and \( h \) (numerator and denominator) values.