Instructions:

• Each question is worth 10 points.

• Please put your answers, including programs and program fragments, in a single file. I prefer to get answers in a plain, ascii, text file. If you want, however, you may submit answers in a PDF file. Please do not submit Word or other editor-specific files.

• Clearly mark the question number corresponding to each answer.

• Send your answer file by email to cram@cs.sunysb.edu

• For programs, you may use XSB’s library predicates unless expressly prohibited. You may also use predicates from Bratko’s book, but for each predicate, please remember to cite the page number in the text where the predicate is defined.

• For programs, please make sure you indent the programs to make them readable, to avoid grading problems.
1. Write a Prolog predicate \texttt{histogram(L, H)}, that, given a list of atoms \( L \), computes \( H \), a list of pairs which represents the histogram of \( L \). In a pair \((a, n)\) in the histogram list \( H \), \( a \) represents an atom in the given list \( L \), and \( n \) is the number of occurrences of \( a \) in \( L \). For instance, \texttt{histogram([a,b,a,b,c,a,c], H)} succeeds with \( H = [(a,3), (b,2), (c,2)] \).

2. Let list \( X \) contain the set of all positions in a tic-tac-toe board where “X”s occur. For instance, \( X=[(1,1),(1,2),(1,3)] \) represents a board which has three “X”s, all in the first row; \( X=[(1,3),(1,2),(1,1)] \) also represents the board which has three “X”s, all in the first row; \( X = [(1,1),(2,2),(3,3)] \) represents a board which has three “X”s, in a diagonal from top left to bottom right. Write a Prolog predicate \texttt{wins(X)}, that, given a list of positions with “X”s, succeeds if and only if “X” wins: i.e. there are three “X”s in the same row, column, or diagonal.

3. Consider the language represented by the following grammar:

\[
E \rightarrow E \ a \ T \\
E \rightarrow T \\
T \rightarrow T \ m \ F \\
T \rightarrow F \\
F \rightarrow x \ E \ y \\
F \rightarrow n
\]

In the above grammar, the symbols \( a, m, x, y, \) and \( n \) are terminal symbols, the rest are nonterminals, and \( E \) is the start symbol of the grammar.

Write a parser in Prolog (using DCGs) to recognize strings (represented by list of terminal symbols) in the above language. If your parser needs tabling, be sure to specify it.

4. Write a Prolog predicate \texttt{gen(N, L)}, that, given a positive integer \( N \), gives the set of all sentences of length \( N \) or less generated by the following grammar:

\[
S \rightarrow S \ S \\
S \rightarrow a \ S \ b \\
S \rightarrow \epsilon
\]

For example, \texttt{gen(6, L)} should give the following answers by backtracking (not necessarily in the same order, but answers should not be repeated):

- \( L = [] \)
- \( L = [a,b] \)
- \( L = [a,a,b,b] \)
- \( L = [a,b,a,b] \)
- \( L = [a,a,a,b,b,b] \)
- \( L = [a,a,b,a,b,b] \)
- \( L = [a,a,b,b,a,b] \)
- \( L = [a,b,a,a,b,b] \)
- \( L = [a,b,a,b,a,b] \)
- \( L = [a,b,a,b,a,b] \)
5. Consider the following normal logic program:

\[
\begin{align*}
t(0). \\
t(s(s(X))) & :- t(X). \\
f(s(0)). \\
f(s(s(s(X)))) & :- f(X).
\end{align*}
\]

(a) Using SLDNF resolution, compute the first three answers (upon backtracking) for the query \( t(X), \text{not } f(X) \).

(b) Can SLDNF answer the query \( \text{not } f(X), t(X) \)? Explain.

6. Consider the following normal logic program:

\[
\begin{align*}
vc(X) & :- edge(X,Y), \text{not } vc(Y). \\
edge(1,2). \\
edge(1,3). \\
edge(2,3). \\
edge(2,4). \\
edge(3,4).
\end{align*}
\]

Enumerate all the stable models of the above program.

7. Consider the following definite logic program

\[
\begin{align*}
sg(X,X). \\
sg(X,Y) & :- parent(X,U), parent(Y,V), sg(U,V).
\end{align*}
\]

defined over a \texttt{parent} relation that represents complete a binary tree of height \( N \) for some given \( N \). For instance, for \( N = 2 \) the following defines the \texttt{parent} relation:

\[
\begin{align*}
\text{parent}(2,1). \\
\text{parent}(3,1). \\
\text{parent}(4,2). \\
\text{parent}(5,2). \\
\text{parent}(6,3). \\
\text{parent}(7,3).
\end{align*}
\]

(a) Consider the \texttt{parent} relation for \( N = 2 \). When the query \( sg(A,B) \) is evaluated using OLDT resolution (i.e., with tabling), list all the calls that will be stored in the call table.

(b) Consider the \texttt{parent} relation for an arbitrary \( N \) (\( N \leq 1 \)). When the query \( sg(A,B) \) is evaluated using OLDT resolution (i.e., with tabling), how many calls will be stored in the call table? (Write this as a function of \( N \).)
8. Consider programs in an imaginary object-oriented language. Let `extends/2` be a relation that defines the class hierarchy for some program: `extends(S1,S2)` means that `S1` is an immediate subclass of `S2` (in Java terms, `S1` extends `S2`).

Let `attr/3` be a relation that defines the values of class attributes (“static” fields, in Java terminology): `attr(C, A, V)` means that an attribute `A` is defined in class `C` and has value `V`. Note that `attr/3` represents only those attribute values defined in a class and not those that are inherited from super classes.

Write a logic program (use tabling if needed) to define predicate `value(C, A, V)` that, given a class `C` and an attribute `A`, computes the value of the static attribute `A` for `C`. Unlike `attr/3`, `value` computes the attribute values that may be inherited from super classes.

For example, given the following definitions of `extends/2` and `attr/3`,

```
extends(boo, zoo).
extends(goo, zoo).
extends(foo, boo).

attr(zoo, a, 1).
attr(boo, a, 2).
attr(boo, b, 3).
attr(goo, b, 4).
attr(foo, c, 5).
```

- query `value(foo, a, V)` should succeed with `V = 2`;
- query `value(foo, b, V)` should succeed with `V = 3`;
- query `value(goo, a, V)` should succeed with `V = 1`;
- query `value(goo, A, V)` should succeed with (upon backtracking): `A = a, V = 1`; and `A = b, V = 4`.
- query `value(foo, A, V)` should succeed with (upon backtracking): `A = a, V = 2`; `A = b, V = 3`; and `A = c, V = 5`.

Clearly state the assumptions under which your `value` relation works as defined.

Your definition of `value/3` must work regardless of the size of the `extends` and `attr` relations.
9. Let list $A$ represent a sequence; a subsequence $B$ of $A$ is a list where the elements of $B$ occur in the same order in $A$ (but possibly separated by other elements of $A$). For instance, $[t, i, r, a]$ is a subsequence of $[t, o, g, i, c, p, r, o, g, r, a, m, m, i, n, g]$.

The following program computes the longest common subsequence of two lists:

```prolog
:- table lcs/4.
% lcs/4: first two arguments are the two given lists.
% the third argument is the length of the longest common subsequence
% the fourth argument is a longest common subsequence.
lcs([], _, 0, []).
lcs(_, [], 0, []).
lcs([X|Xs], [X|Ys], N, [X|Zs]) :-
  lcs(Xs, Ys, Zs, M),
  N is M + 1.
lcs([X|Xs], [Y|Ys], N, Zs) :-
  X \= Y,
  lcs(Xs, [Y|Ys], M1, Zs1),
  lcs([X|Xs], Ys, M2, Zs2),
  longer(M1, M2, Zs1, Zs2, M, Zs).

longer(M1, M2, L1, L2, M, L) :-
  (M1 > M2
   -> L = L1, M = M1
   ;   L = L2, M = M2
  ).
```

(a) Consider evaluating the query $\text{lcs}(L1, L2, N, L)$ for two given lists $L1$ and $L2$ using tabled evaluation. How many different calls will be placed in the call table during query evaluation? Give your answer as a function of the lengths of $L1$ and $L2$.

(b) For each call in the call table, how many different answers will be computed?

(c) How long does it take to compute a single answer, assuming that all other answers already exist in the tables?

(d) How long does it take to evaluate a query $\text{lcs}(L1, L2, N, L)$ for two given lists $L1$ and $L2$? Give your answer in terms of worst case times, as a function of the lengths of $L1$ and $L2$.

10. (a) List the WAM instructions that are used to create a query term in the heap.

(b) List the WAM instructions that are used to unify a program term with a query term already present in the heap.

(c) Consider the modification of a Prolog implementation that does unification with occurs check. Which of the above instructions need to be modified? Explain.