Question 1

Consider the KB that has the following fact: \{p(X, Y)\}

Give a semantic proof that \( KB \models p(a, b) \).

Note that as \( p \) is a 2-ary predicate, \( \pi(p) \) will map to a set of pairs of domain elements. For example, if John and Paul are in the domain, it might be the case the \( (\text{John}, \text{Paul}) \in \pi(p) \). Note that the angle brackets are simply be used to form pairs of objects.

Question 2: Exercise 3.1a from the textbook

To get you on track, think of the sentence “dogs are animals” as being “if something is a dog, then it is an animal.”

Question 3: Exercise 3.1b from the textbook

Question 4: Exercise 3.1c from the textbook

Question 5: Exercise 3.5a from textbook

Below is the standard definition for concatenating two lists, where \( \text{concat}(A, B, C) \) is true if \( C \) is the concatenation of \( A \) and \( B \). The definition uses the standard Prolog square bracket notation.

\[
\text{concat([], Z, Z).}
\]

\[
\text{concat([A|X], Y, [A|Z]) \leftarrow}
\]
\[
\text{concat(X, Y, Z).}
\]

I am using the predicate name ‘concat’, while the textbook follows the Prolog convention and calls it ‘append’. However, that causes confusion with python, as python uses ‘append’ to add a single element onto a list. Hence, whenever you see ‘append’ in the textbook, think of it as ‘concat’.

What is the result of the following two queries. Give all solutions. Feel free to use Prolog to help you determine the answers, but make sure you understand how Prolog derived the answer.

\[?\text{concat(A, [a|B], [b, a, d, a, c, a])}\.\]
\[?\text{concat(X, [A, B|Y], [a, b, c, d, e])}\.\]

Question 6: Exercise 3.5b from textbook

Question 7: Exercise 3.6a from textbook

Question 8: Exercise 3.6b from textbook

Question 9: Exercise 3.6c from textbook

Subsequence does not have to be consecutive. So, \( \text{subsequence}(A, B) \) should be true if there is a way of removing elements from the list \( B \) to get the list \( A \). So, \( \text{subsequence}([a, c], [a, b, c, d, e]) \) returns TRUE.
Note that \texttt{subsequence([c,a],[a,b,c,d,e])} should return false, as the book specified that the sequence should be “in the same order”.

It is recommended that you verify your results in Prolog.

\textbf{Number Representation}

For the following questions, we will be using the number representation from class, in which the constant symbol \texttt{0} maps to the number 0, \texttt{s(0)} maps to 1, \texttt{s(s(0))} maps to 2, etc. In class, we defined \texttt{subtract} and \texttt{lt} to use this representation. For your convenience, this is repeated here.

\begin{verbatim}
  lt(X,s(X)).
  lt(X,s(Y)) <- lt(X,Y).

  subtract(X,0,X).
  subtract(X,s(Y),Z) <- subtract(X,Y,s(Z)).
\end{verbatim}

\texttt{lt(X,Y)} is true if \(X < Y\) and \texttt{subtract(X,Y,Z)} is true if \(X - Y = Z\).

\textbf{Question 10}

Define the predicate \texttt{plus}. \texttt{plus(X,Y,Z)} should be true if \(X + Y = Z\).

Note that your predicate should only return a single solution. In Prolog, if you press “;” after an answer, it should not give another one. (You can avoid this by not defining redundant base cases.)

\textbf{Question 11}

Define predicates \texttt{times} and \texttt{neq}. \texttt{times(X,Y,Z)} is true if \(X \times Y = Z\) and \texttt{neq(X,Y)} is true if \(X \neq Y\). Note that you can use \texttt{plus}, \texttt{subtract} and \texttt{lt} in defining them.

\textbf{Question 12}

Define the predicate ‘is(X,Expr)’. Assume that expression is in prefix format, and is some combination of the functions \texttt{plus(_,_)}, \texttt{times(_,_)} and \texttt{subtract(_,_)}.

Note that these 2-ary \texttt{plus}, \texttt{times} and \texttt{subtract} are different from the 3-ary ones before. The 3-ary ones are predicate symbols. The 2-ary ones are function symbols. In datalog, you do not directly define functions. Instead you define them in terms of the predicates that use them, which in this case is the \texttt{is} predicate.

So, \texttt{is(X,plus(s(s(0)),times(subtract(s(s(s(0))),s(0)),s(s(1))))))} is valid.

\textbf{Programming in Python}

\textbf{Question 13: Occurs Check}

Make a procedure called \texttt{occurscheck} that takes a variable and a term as input. It should succeed (return \texttt{True}) if the variable does not appear in the term, and should fail otherwise (return \texttt{False}).

Note that your procedure needs to work whether the term is a variable, constant or an arbitrarily deep function. Here is some code to start you off. Do this without recursion (without having occurscheck call itself) and without calling \texttt{findvariables}.
def occurscheck(var, term):
    # need to deal with case where term is not a function
    if not (type(term) is list):
        #YOUR CODE HERE
        todo = term[1:]
    while not todo == []:
        first = todo[0]
        todo = todo[1:]
        #YOUR CODE HERE
        # note that as soon as find the var in the term
        # you can return the value True right away

        # we haven’t seen the var in the term, so return True
        return True

Hand in the code and show that it works properly on a few examples. You should choose your examples so that they reasonably demonstrate that your code works correctly. You should have examples that both should fail the occurscheck and that pass the occurscheck, and that are are not deeply embedded, and that are deeply embedded.

Question 14: Programming: Unification for Datalog with Functions

In class, we discussed the unification algorithm. Unify should take two atoms and return 0 if they cannot be unified, or return the most general unifier if they can. In this question, you will program the unification algorithm.

As with the previous homework, you will represent the substitution set with a Python dictionary. The unify procedure will work for any pair of atoms, and arbitrary nesting of functions inside of the atoms.

In your code, the most important issue is that the code is simple and easy to understand. Do not worry too much about the efficiency of the code.

In the class webpage, right after the link for this homework, there is a link for some code for you to use in this homework. It includes code for the substitute procedure, which takes as input an atom or a term. Note that it will allow illegal Datalog sentences (variables used as the predicate or function symbol), but as long as you give it legal Datalog sentences, it will work correctly. There are also a few other procedures to make your life easier. test will call your unify and nicely format the result of the unification.

In your Python program, add the following line at the beginning of your code to automatically include the code from hw3standard.py using execfile(hw3standard.py). You do not need to hand in hw3standard. You will also use your occurcheck procedure from the previous question.

To help you out further, I am giving you the following piece of code. This code takes two atoms and looks for the first difference between them. If there is no difference, it returns True. Otherwise, if the first difference can be resolved by a substitution, it returns the substitution (as a list with the variable followed by its value). Otherwise, the procedure returns False. Note that it does not do the occurs check.

def diff(a, b):
    if a == b:
        return True
    if isVar(a):
        return [a, b]
    if isVar(b):
        return [b, a]
    if (not type(a) is list) or (not type(b) is list):
        return False
    if not len(a) == len(b):
        #YOUR CODE HERE
return False
for (parta,partb) in zip(a,b):
    result = diff(parta,partb)
    if result == False:
        return False
    if not result == True:
        return result
return True

With the above diff procedure, the code for unify is drastically simplified. unify will use diff to determine all of the substitutions necessary in order to unify the two atoms. unify(a,b) will work as follows. It will repeated call diff. For each difference that is found, unify will make sure that the new substitution does not violate the occurs check. It will then apply the new substitution to both of its inputs, and so removing one difference. It will then apply the new substitution to the substitutions already found.

Hand in your code and show that it works for Exercise 2.10. Make sure you use test to format your results.

Also, comment on why this is a very inefficient way of writing the unify procedure.

**Question 15: Renaming**

To use the unify procedure as part of a top-down proof procedure, we need a way to rename the variables in an expression, so we don’t get any variable clashes when unifying.

Let’s allow a variable to also start with “_”. We will reserve these for system use. When we rename a variable, we will start it with “_” followed by a unique number. Use the global variable NextVariable to keep track of the next unique number to use. Each time you need a new variable, you will form it from “_” and NextVariable, and then you will increment NextVariable.

Make a procedure freshvariables that first uses findvariables (in hw3standard) to find all of the variables in the expression. It then creates a substitution list that will map the found variables to new variables that begin with “_” followed by a unique number. It then uses substitute to apply the substitutions. It should take as input the original expression and return a new expression.

Hand in a copy of your code, and show its results on several examples. Make sure your examples are complicated enough to test your procedure. You should format your answers using prettyexpr, which is also in hw3standard.