Quick Prolog Primer

For this homework assignment, you will need to install prolog. I recommend swi-prolog, as it is actively being developed, and runs on a range of operating systems, including Windows, Macs, and linux.

To run swi-program, put your KB into a text file (with extension .prolog). Instead of ←, use ‘:-’ as the separator between the head of a rule and its body. Rather than ∧, use a comma to separate atoms in the body. Also, put a period after the end of each clause. Any variables that only appear once in a clause should be replaced by ‘.’. Style-wise, I recommend you put the atoms on in the body on separate lines, indented as the following shows:

```prolog
memb(Top,p(Top,_)).
memb(Top,p(X,Rest)) :-
    memb(Top,Rest).
```

I called this predicate ‘memb’ as prolog has a built-in predicate that does the same functionality. (Note that prolog’s built-in member predicate uses prolog’s list notation: [X|Y].)

Let’s assume your text-file that contains your KB is called hw3.prolog. To start prolog with your KB, you can type the following into a command line:

```
swipl hw3.prolog
```

Alternately, you can start swipl, and type the following into its command window, after the ‘?-’ prompt.

```
['program.prolog'].
```

Once your code is loaded, you can ask queries after the ‘?-’ prompt, such as:

```prolog
memb(a,p(1,p(2,p(a,nil))))).
```

Note that there is a period after the end of your query.

After typing this in and pressing enter, prolog should return true. Notice that prolog is waiting for you to do something. You can either press enter to end the query or press ‘;’. The latter will cause prolog to see if there are any other answers, to which prolog will answer false.

Now type in the following query.

```prolog
memb(X,p(1,p(2,p(a,nil))))).
```

Here, prolog will return 1. If you press ‘;’, prolog will search for alternate solutions, and return ‘2’, then ‘a’, and then false. When you press ‘;’, prolog treats the first answer as a fail, and backtracks and looks for alternate answers.

To exit out of prolog, you can simply press control-D.

**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 \( \langle \text{John}, \text{Paul} \rangle \in \pi(p) \). 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

You can use prolog to verify that your query and KB is correct.

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 \texttt{concat}(A,B,C) is true if C is the concatenation of A and B. The definition uses the standard Prolog square bracket notation.

\begin{verbatim}
concat([],Z,Z).
concat([A|X],Y,[A|Z]) <-
    concat(X,Y,Z).
\end{verbatim}

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. Download a version of Prolog to help you determine the answers, but make sure you understand how Prolog derived the answer. In Prolog, press ‘;’ after a solution to have it search for other answers.

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

Question 6: Exercise 3.5b from textbook

Make sure to give all answers.

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, \texttt{subsequence}(A,B) should be true if there is a way of removing elements from the list B to get the list A. So, \texttt{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.

Number Representation

For the following questions, we will be using the number representation from class, in which the constant symbol 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
lt to use this representation. For you 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}

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

\textbf{Question 10}

Define the predicate \textit{plus}. \textit{plus}(X,Y,Z) \textit{should be true if} \textit{X} + \textit{Y} = \textit{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 \textit{times} and \textit{neq}. \textit{times}(X,Y,Z) \textit{is true if} \textit{X} \times \textit{Y} = \textit{Z} and \textit{neq}(X,Y) \textit{is true if} \textit{X} \neq \textit{Y}. Note that you can use \textit{plus}, \textit{subtract} and \textit{lt} in defining them.

\textbf{Question 12}

In this question, we are going to make a predicate similar to the built in prolog predicate \textit{is}.

Define the predicate \textit{is1}(X,Expr). The second argument \textit{Expr} is an input variable that is a mathematic expression, using a combination of the \textbf{function} symbols \textit{plus}(\_\_,\_), \textit{times}(\_\_,\_) and \textit{subtract}(_\_\_,\_). The \textit{is1} predicate should determine the value of \textit{Expr} and unify it with \textit{X}. This predicate will use numbers expressed using the function symbol \textit{s} and the constant \textit{0}. For example, the following:

\textit{is1}(X,\textit{plus}(\textit{s}(\textit{s}(\textit{0})),\textit{times}(\textit{subtract}(\textit{s}(\textit{s}(\textit{s}(\textit{0}))),\textit{s}(\textit{0})),\textit{s}(\textit{s}(\textit{1}))))))

should unify \textit{X} with \textit{s}(\textit{s}(\textit{s}(\textit{0}))).

To define the \textit{is}, you should use the 3-ary \textbf{predicate} symbols that we have previously defined, for \textit{plus}, \textit{times} and \textit{subtract}.

Basically, you are building the equivalent of prolog’s is predicate, but using function notation for numbers. Remember, the 2-ary plus, times and subtract are just function symbols, which in prolog are \textbf{not} defined, but defined via predicates that act on them, which in this case is our \textit{is} predicate.

Also note: \textit{Expr} can be complex, like in the example above, so you will need a recursive definition for \textit{is}, along with a base case. You should also make sure that your routines only produce a single answer.

\section*{Programming in Python}

\textbf{Question 13: Occurs Check}

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

\begin{verbatim}
def occurscheck(var,term):
\end{verbatim}
# 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 some examples that
fail the occurscheck and some that pass the occurscheck. You should have examples that span from not
embedded at at, to being 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

from hw3standard import *

You do not need to hand in hw3standard. You will also use your occurcheck procedure from the previous
question.

To help you out further, the following code is also hw3standard. The routine diff 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):
    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.

To test your code and show the substitution list, use the following code (which is not in hw3standard).

```python
def print_unify(A,B):
    res = unify(A,B)
    if res == 0:
        print("%s and %s do not unify" % (prettyexpr(A), prettyexpr(B)))
    else:
        str = "%s and %s unify with" % (prettyexpr(A), prettyexpr(B))
        for v in res:
            str += " %s/%s" % (v, prettyexpr(res[v]))
        print(str)
```

Hand in your code of unify and show that it works for Exercise 2.10. Also, show several complex examples where it should not unify. Make sure you use print_unify 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.