Assignment One

out February 2, due February 14, 1993

This assignment is intended to be a very simple one which will give you an opportunity to get your feet wet with Prolog. You should review chapter eight in the *SICStus Prolog User's Manual* which provides a terse introduction to Prolog as well as Chapters One (How to run Prolog) and Two (Debugging). You may want to consult the book *Programming in Prolog* by Clocksin and Mellish, which provides a good introduction to Prolog.

1 Append - Warm Up Exercise

This is just to help those who are very new to Prolog get started. Define the append predicate by typing the clauses directly into the Prolog interpreter. Figure ?? shows a sample session with Sicstus Prolog. The first line (*[user].*) tells Prolog to *consult*, or read new clauses to be added to the database, the user. That is, clauses will be read from the terminal and added to the database. The next two lines are the two clauses that will define *append/3* (i.e., a predicate named *append* with three arguments). The (control-D) is how one signals an *"end of file"* to the Prolog reader. In this case it causes the Prolog reader to stop *"consulting the user"*. Note that the periods at the end of the first three lines are required.

Now try to append the list *[a,b,c]* to the list *[1,2,3]* by typing in the goal *append([a,b,c],[1,2,3],X).*. If this is successful, the binding for the variable X will be displayed and the interpreter will wait for you to type a carriage-return (indicating that you want no more answers) or a semi-colon (indicating you want to to seek another answer). To help you understand how this definition of *append* works, you can cause the interpreter to trace the append predicate by issuing the command *trace(append/3).* This will cause the interpreter to print informative messages when it is executing append/3. Note that it pauses whenever the predicate is called, typing an h will show the possible responses and typing a carriage-return will cause it to continue ("creep").

Now try some other ways to use your append predicate.

Show what happens when you ask Prolog to solve each of the following goals

1. append([a,b,c],X,[a,b,c,d,e,f]).
2. append(X,[d,e,f],[a,b,c,d,e,f]).
3. append(X,Y,[a,b,c,d,e,f]).
4. append(X,[d,e,f],Y).
5. append(X,Y,Z).

```
SICStus 2.1 #6: Mon Dec 14 18:09:19 EST 1992
| ?- [user].
| append([],X,X).
| append([Head|Tail],X,[Head|Rest]) :- append(Tail,X,Rest).
| ^D {user consulted, 0 msec 480 bytes}

yes
| ?- append([a,b,c],[1,2,3],X).

X = [a,b,c,1,2,3] ?

yes

| ?- trace.
{The debugger will first creep -- showing everything (trace)}

yes
{trace}
| ?-  append([a,b],[c,d],X)
     .
   1  1  Call: append([a,b],[c,d],_97) ?
   2  2  Call: append([b],[c,d],_276) ?
   3  3  Call: append([],[c,d],_405) ?
   3  3  Exit: append([],[c,d],[c,d]) ?
   2  2  Exit: append([b],[c,d],[b,c,d]) ?
   1  1  Exit: append([a,b],[c,d],[a,b,c,d]) ?

X = [a,b,c,d] ?

yes
```

Figure 1: A sample session using Sicstus Prolog

2 Set manipulations

Suppose we chose to represent sets of objects in Prolog as lists. We could define the predicate *union/3* as follows:

```
union([],Set,Set).
union([X1|Rest1],Set2,Union) :-
  union(Rest1,Set2,Union),
  member(X1,Union).
union([X1|Rest1],Set2,[X1|Union]) :-
  union(Rest1,Set2,Union),
  \+ member(X1,Union).
```

For this definition,

1. Does this program work correctly if the first two arguments are instantiated and the third is not?
2. What happens if we try the goal *union(X,[1,2,3],[1,2,3,4,5])*?
3. Write a predicate *intersection(S1,S2,S3)* which is true if the intersections of the two sets S1 and S2 is equal to the set S3. You can assume that the arguments will be well formed sets (i.e. lists of ground terms without duplicates) and that the first two arguments will always be instantiated.

3 Kinship Relationships

This problem will give you some experience designing a simple logic programming database in terms of primitive facts and rules. The domain is one we are all familiar with - kinship relations.

Our western-European culture considers a number of kinship relations to be especially meaningful and gives them names. Other cultures have developed a much richer set of relationships. Most of these relations can be defined in terms of a small number of primitive relations and attributes, such as those having to do parenthood, marriage, gender, age, etc. Your task is to:

- Choose a set of primitive relationships to encode as facts that will allow you to define the rest of the relations as rules.
- Test out your database design be entering the facts about your own immediate family and trying out each of the relations.
- Hand in a listing of your database (facts and rules), and a sample run showing a query for each predicate defines.

The relationships you must define are the following:

- *child(X,Y)* - true if X is a child of Y.
- *daughter(X,Y)* - true if X is a daughter of Y.
- *parent(X,Y)* - true if X is a parent of Y.
- *mother(X,Y)* - true if X is the mother of Y.
- *sibling(X,Y)* - true if X and Y are siblings (i.e. have the same biological parents). Be sure your definition does not lead to one being one's own sibling.
- *brother(X,Y)* - true if X is a brother of Y.
- *grandparent(X,Y)* - true if X is a grandparent of Y.
- *grandmother(X,Y)* - true if X is a grandmother of Y.

- *uncle(X,Y)* - true if X is an uncle of Y. Be sure to include uncles by marriage (e.g. your mother's husband) as well as uncles by blood (e.g. your mother's brother).
- *sister-in-law(X,Y)* - true if X is a sister-in-law of Y.
- *mother-in-law(X,Y)* - true if X is a of Y.
- *spouse(X,Y)* - true if X and Y are married.
- *wife(X,Y)* - true if X is the wife of Y.
- *ancestor(X,Y)* - true if X is a direct ancestor of Y (i.e. a parent or an ancestor of a parent).
- *descendant(X,Y)* - true if X is a descendant of Y (i.e. a child or an ancestor of a child).
- *relative-by-blood(X,Y)* - true if X is a blood relative of Y (i.e. related through some combination of offspring relations). Hint: your blood relatives are just those people with whom you share a common ancestor.
- *relative(X,Y)* - true if X and Y are related somehow (i.e. through some combination of offspring and marriage relations). Hint: this is easy to define in terms of relative-by-blood and spouse!
- *male(X,Y)* - true if X is male.

The goal is to devise a good representation – one that is intuitive and natural and would be easy to extend. You may need to (or want to) define other relations, e.g., a gender/2 relation. Watch out for "loops", or, more accurately, unterminating recursion. For example, you do not want a rule like `spouse(X,Y) :- spouse(Y,X)`.

4 Extra Credit

This is an optional problem. Study the following popular American (I think) folk song. Try to model it with the predicates you developed in the previous problem and any extensions that are necessary. Do you reach reach the same conclusion as the singer?

> Many, many years ago when I was seventeen,
> I was married to a widow who was pretty as could be.
> This widow had a grown up daughter who had hair of red.
> My father fell in love with her and soon they, too, were wed.
>
> This made my dad my son-in-law and it changed my very life.
> For my daughter was my mother though she was my father's wife.
> To complicate the matter, even though it brought me joy,
> I soon became the father of a bouncing baby boy.
>
> This little baby then became a brother-in-law to Dad.
> And so became my uncle, though it made me very sad.
> For if he was my uncle then it also made him brother.
> To the widow's grown up daughter who, of course, was my stepmother.
>
> Father's wife then had a son who kept them on the run,
> And he became my grandchild 'cause he was my daughter's son.
> My wife is now my mother's mother and it makes me blue,
> Because, although she is my wife, she's my grandmother, too.
>
> And if my wife is my grandmother, I am her grandchild,
> And every time I think of it, oh, it nearly drives me wild.
> For now I have become the strangest case you ever saw
> As husband to my grandmother, I am my own grandpaw,... oh ...
>
> I am my own grandpaw.
> Yes, I am my own grandpaw.
> It sounds funny, I know,
> But it really is so.
> I am my own grandpaw.