Simon Fraser University
School of Computing Science

CMPT 383

Assignment 3 (Prolog)

Due date: November 22, 2005

1) Convert the following predicate calculus to Horn clause(s).
   ∀g ((logician(g) ∩ ∀a(argument(l,a)⊃ sound(a))) ⊃ happy(g))

2) Given the following Prolog program

```
no_doubles([],[]).
no_doubles([X|Xs],Ys) :- member(X,Xs), no doubles(Xs,Ys).
no_doubles([X|Xs],[X|Ys]) :- nonmember(X,Xs), no_doubles(Xs,Ys).

nonmember(X,[]).
nonmember(X,[Y|Ys]) :- X\==Y, nonmember(X,Ys).

member(X,[X|_]).
member(X,[Y|T]) :- X \== Y, member(X,T).
```

   Describe the complete execution trace, using a graphic representation, of the following goal:

```
?- no_doubles([a,b,a,c,b],X).
```

3) Write the predicate **`difference/3`** that defines the set subtraction relation, where all three sets are represented as lists. For example:

```
?- difference( [a,b,c,d], [b,d,e,f], D ).
D = [a,c]
```

4) Write the predicate **`merge/3`** to merge two sorted lists producing a third list. For example:

```
?- merge( [2,5,6,6,8], [1,3,5,9], L ).
L = [1,2,3,5,5,6,6,8,9]
```

5) Write the predicate **`split/3`** to split a list of numbers into two lists: positive ones (including zero) and negative ones. For example:

```
?- split( [3,-1,0,5,-2], P, N ).
P = [3,0,5]
Q = [-1,-2]
```

6) Define the predicate **`palindrome(List)`**.A list is a palindrome if it reads the same in the forward and in the backward direction. For example:

```
?- palindrome([m,a,d,a,m]).
```

```
yes
```

7) Define two predicates **`evenlength(List)`** and **`oddlength(List)`** so that they are true if their argument is a list of even or odd length respectively. For example, the list `[a,b,c,d]` is 'evenlength' and `[a,b,c]` is 'oddlength'.

8) Assume that a rectangle is represented by the term `rectangle(P1,P2,P3,P4)` where the `P's` are the vertices of the rectangle positively ordered. Define the predicate **`regular(R)`**, which is true if `R` is a rectangle whose sides are vertical and horizontal.

9) Write the predicate **`simplify/2`** to symbolically simplify summation expressions with numbers and symbols (lower-case letters). Let the predicate to rearrange the expressions so that all the symbols precede numbers. For example:

```
?- simplify( 1+1+a, E ).
E = a+2
?- simplify( 1+a+4+2+b+c, E ).
E = a+b+c+7
?- simplify( 3+x+x, E ).
E = 2*x+3
```

10) Define the predicate **`between(N1,N2,X)`** which, for two given integers `N1` and `N2`, generates through backtracking all integers `X` that satisfy the constraints $N1 \leq X \leq N2$.

Programming Assignment: *Kinship Relations*

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.
- *grandfather(X,Y)* - true if X is a grandfather 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's brother) 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.
- *relative-by-blood(X,Y)* - true if X is a blood relative of Y (i.e. related through some combination of offspring relations).
- *relative(X,Y)* - true if X and Y are related somehow (i.e. through some combination of offspring and marriage relations).
- *young_parent(X)* – true if X has a child but does not have any grandchildren.

The goal is to devise a good representation -- one that is intuitive and natural and would be easy to extend.

- Choose a set of primitive relationships to encode as facts that will allow you to define the rest of the relations as rules. You might choose, for example, parent/2, male/1, female/1, spouse/2 as the predicates to enter as facts and define the rest using rules.
- You may need to (or want to) define other relations, e.g., a gender/2 relation.
- Test out your database design by 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.