CSCI 162 Lab 7

Purpose: To Connect Prolog to Predicate Logic.

Objectives: After this lab you should be able to

- Create Horn clauses in Prolog.

Background:

Remember that in logic, we like to represent predicates with capital letters (e.g., E(x, y) represents x eats y). Consider this implication: (P(x, y) ^ P(y, z)) -> G(x, z). This says: If P(x, y) is true and P(x, z) is also true, then G(x, z) is true. This is exactly what our grandparent rule from the last lab says in Prolog! The Prolog syntax is just backwards. Think of the :- symbol in Prolog as representing the word "if", and the connection becomes clear.

Internally, Prolog represents rules in a form known as a Horn clause. A Horn clause is a disjunction of predicates in which at most one of the predicates is not negated. Sounds odd, but it matches well with Prolog's needs.

Consider the grandparent clause again:

```
grandparent(X,Z) :- parent(X,Y) , parent(Y,Z).
```

Rewritten as in our logical notation, including quantification:

∀x ∀y ∀z ((P(x, y) ^ P(y, z)) -> G(x, z))

We know that p -> q is logically equivalent to !p ∨ q, and so the above is equivalent to:

$$\forall x\ \forall y\ \forall z\ (!(P(x, y) \wedge P(y, z)) \vee G(x, z))$$

And De Morgan's Laws tell us that we can replace the AND with an OR as the negation is pulled through, producing a Horn clause:

$$\forall x\ \forall y\ \forall z\ (!P(x, y) \vee !P(y, z) \vee G(x, z))$$

Here's why having a Horn clause is important: Prolog has facts in its database about parents. By employing a simplified version of the resolution rule of inference, Prolog can use a fact to remove a predicate from the Horn clause and get closer to an answer. For example, for the query grandparent(hank,X), the resulting Horn clause would be not parent(hank,A) or not parent(A,B) or grandparent(hank,B). The database tells Prolog that parent(hank,ben) is a fact, and Prolog can resolve that fact with the Horn clause if A = ben. That assignment is called unification, and the result is the Horn clause not parent(ben,B) or grandparent(hank,B). Finding that ben is the parent of carl lets Prolog use resolution and unification once more to conclude that grandparent(hank,carl) is true.

Yes, Virginia; there are good reasons to learn about logic if you want to be a computer scientist!

Instructions:

Copy the family file in Moodle to your directory.

You are to write the following:

1. sibling(X,Y)

2. brother(B,X)

3. sister(B,X)

4. grandparent(G,X).

5. cousin(C,X). Neither self nor sibling is a cousin. You can use the operator `not(P)', which will be satisfied only if P is not -- e.g.,

```
male(X):- not female(X).
```

or

```
male(X):- not(female(X)).
```

```
Note: The GNU-prolog manual does not mention a
not() predicate. The unary operator \+ does
negation-by-failure, so this should be:
```

```
male(X) :- \+ female(X).
```

6. ancestor(X,Y). I am my own ancestor, and every ancestor of my parent is also my ancestor; nobody else is my ancestor.

7. Construct a structure that is satisfied if a person has three distinct grandchildren. Use `;' to determine all the whales in the database who have three distinct grandchildren. Write them down and hand in the result.

What to hand in:

Hand in a listing of your updated family file for parts 1-7.