Assignment 5

CSE 473 – Introduction to Artificial Intelligence, University of Washington, Spring 2010.

Due Friday, May 14, at 5:00 PM.

Predicate logic is probably the most important single method for knowledge representation and sound inference. This assignment begins with some relatively simple exercises and then provides an opportunity to use logic in the realm of recommendation agents – in particular for suggesting movies appropriate for a group of friends to view together. We use the PROLOG language, both for its convenient query and search capabilities and to see how AI-oriented programming languages can help developers think at a somewhat higher level of abstraction than might be done with more standard languages.

Part I. Warm-up Exercise.

1) The `isa/2` example in the "Applying Prolog" slides does not make a standard ISA hierarchy because it uses "is a" with two different meanings. "Spot is a dog" differs from "A dog is an animal." We probably want to say that "A dog is a dog", but we might not want to say that "Spot is a Spot."

Change the definition of `isa/2` so that `isa(A,A)` is true if A is a class or subclass, but not if A is an instance. Leave `isa/2` ambiguous so that the first argument can still be either an instance or a class.

If you run the query `isa(X,Y)`, your new definition should give all possible correct answers on backtracking, but not given any of them more than once.

2) Write a recursive predicate `all_are/2` that will check whether all elements in some list belong to some class. The query
`all_are([spot,fang,snake],animal)`
should be true, while
`all_are([spot,cactus],animal)`
should be false. If the list is empty, the query should be true.

Part 2. Movie Recommender

For this assignment, you will be designing a "Friends Movie Selection Advisor" that will suggest movies for people to see with their network of friends, and friends of friends. The suggestions should be constrained to suggest only movies that everyone in the group wants to see, or would like. Your program should work with facts about who is friends with whom. To find a network of friends that might want to see a movie together, you need to build a recursive set of friends and friends of friends. Although you will be given a small set of sample data, make sure your code can work with an arbitrarily large network of friends. To keep things simple, we will start by assuming that friendship is reflexive f(A,A), and transitive,

(f(A,B),f(B,C) → f(A,C)) but NOT a symmetric relationship (f(A,B) ↛ f(B,A)).

Friends

Encode the following sample information as a set of facts in a file called friends.pl. Use lower-case for the names.

```
Chris is friends with Lee.
Dale is friends with Myrna.
Chris is friends with Spike.
Pat is friends with Chris.
Myrna is friends with Sean.
```

In this file, also define a recursive predicate `friends/2` where `friends(A,B)` means that A is friends with B. This predicate will be used for testing, so keep the exact name for this recursive predicate. You can call your base level "fact" predicates anything you want. Because we will be backtracking through thousands of movie "facts" you want to make sure your definition does not give redundant answers. The unit tests (see below) will check for this. You can make up your own set of friends, but keep their names disjoint from this group. This group will be used for testing.

Preferences

Encode the following sample information as a set of rules in a file called preferences.pl. Most of the rules should use information from the movie database predicates described below.

```
Pat wants a comedy.
Sean likes adventures.
Dale likes dramas.
Lee is ok with anything that Dale would not like.
Myrna wants to see a family movie from the 1950s.
Spike just wants to see something from this century.
Chris wants to see any of the actors who were in one of the Lord of the Rings
movies – but acting in some other movie.
```

Again, you are encouraged to come up with your own set of preferences for your own groups of friends, but keep the preferences described here as exact as you can. These will be used for testing.

Movie Advisor

In a file called advisor.pl define a `suggest/2` predicate that will suggestion a movie that a particular person, and all her friends, will enjoy. For example:

```
?- suggest(chris,Movie).
Movie = 'Star Wars: Episode II - Attack of the Clones (2002)'
```

This top level predicate will be used for testing, so name it as given. Auxiliary predicates you can design and name as you want.

Movie Data

To supplement our locally defined information, we have a set of files based on IMDB movie data. This is similar to the database info used in CSE444. However, we have cut it back to only 10,000 movies, along with the matching actors, cast listings and genres. For those unfamiliar with them, the schemas are as follows:

```
/* movie(MovieId,MovieName,Year,Rating).
* actor(ActorId,FirstName,LastName,Gender).
* cast(ActorId,MovieId,Role).
* genre(MovieId,Genre). */
```

The movie data files are located in the /projects/instr/10sp/cse473/Prolog directory, available on attu, in the files: movie.pl, actor.pl, cast.pl, and genre.pl. The movie and actor databases tend to append various odd things to names, so you might be better off looking for substrings, or, as we say in Prolog, subatoms. SWI-Prolog has the `sub_atom/5` predicate. For example, the database has a movie with the name 'Lord of the Rings: The Fellowship of the Ring, The (2001)'. For our purposes, we are just interested in part of that.

```
movie(Mid,MName,Year,Rating),
sub_atom(MName,_Before,_Length,_After,'Lord of the Rings').
```

Do you want to know more? SWI-Prolog has a help system that you can invoke from the interpreter command line. For example:

```
help(sub_atom).
```

To get the graphical version of help when using attu, you may have to explicitly allow X forwarding:

```
ssh -X attu
```

Loading Files and Unit Tests

To simplify the process of loading multiple files, you can have one file that contains directives (rules with a body but no head) that load other files. An example is provided. By convention, you can consult one or more files from inside the interpreter by "calling" their names inside a list. So if you have a file named go.pl you can consult or reconsult it just by typing [go]. at the interpreter prompt. SWI-Prolog has some facilities for doing unit tests. A sample is provided. This example is set up to be run from the Linux command line on attu. For more information, see the SWI references on Unit Tests and Using PrologScript.

This assignment was designed by David Broderick.