Winter 2012

COSC 4F79 Expert systems Assignment #2

Due date: 12:00 noon Monday February 27; lates Thursday March 1 (-25%).
Objectives: To get some experience running a basic expert system shell, as well as Prolog meta-interpreters.
Hand in: Source listing and example dialogs as appropriate.

1. Type in the following simple meta-interpreter:

```
?- dynamic append/3.    % Sicstus: must include all predicates used by 'clause'.

solve( true ) :- !.
solve( not(A) ) :- !, \+ solve(A).
solve( (A,B) ) :- !,  solve(A), solve(B).
solve( A ) :- clause(A, Body), solve(Body).
```

(a) Run this meta-interpreter on a few simple pure Prolog programs. Using the interpreter's trace facility, trace its execution. Make sure you force it to backtrack using the ';' command at the interpreter prompt. Pay attention to how clauses 3 and 4 execute. Note that you might need "dynamic" definitions as above for all the code you intend to interpret.
(b) Try executing the meta-interpreter on a program with a non-pure feature (eg. a program with a **write**) and see what happens. Use Prolog's trace utility to find out where the problem occurs.
(c) Copy the meta-interpreter in (a), and add a clause to enable the interpreter to handle built-in **write** goals. Test it on an appropriate program.
(d) Copy the meta-interpreter in (a), and remove the cuts and trace execution once again. Determine why the cuts are necessary.
(e) Finally, copy the meta-interpreter in (c), and convert it so that it handles the ";" (or) operator. Test it on a few programs.

2. Take the Merritt backward-chaining shell code on the web, **Native** (Bird identification expert system. Make the following modifications.

a) The shell has has two predicates, "prove/2" and "prov/2". Coalesce these predicates into a single "prove/2" predicate, so that it works similarly to the meta-interpreter in question 1.

b) Modify the explanation facility so that a more legible explanation is printed when "why" is input during user prompts. First, only give one level of explanation at once, and so the user must repeatedly ask "why" for further levels. In other words, each "why" goes up one level in the tree. It will stop at the root of the tree. Also, format the output so that the line of reasoning is clear and readable.

c) Add a trace utility, which can be turned on and off from the top-level driver menu. A trace is a run-time reporting of rules that are being executed during the inference. For the trace, write the test "Solving X..." and "Solved X." where X is the current goal being solved or just solved. Also indent the trace listing to show the depth of the inference. The

trace predicate should not echo the **menuask** and **ask** predicates. These are special cases that should be caught by a few new clauses to be added to **trace.**

d) Add a new command **dump** to the driver of the Native shell. When invoked, dump reads in the name of a rule (without arguments), and then prints out all the rules for that named rule. Each rule should be printed in a legible, tidied format, in which ":-" and "," are replaced by "if" and "and", one goal per line, etc..

e) Extend the bird knowledge base by adding some rules, eg. pigeons and chickens. Make sure your new rules have a moderately deep hierarchical structure.

Execution/testing: Test all the above modifications thoroughly, and give printouts of the execution sessions. Use Linux's "script" to record your revisions. When you hand in your output, clearly indicate what modifications above are being tested in each section of the output listing.