SCHOOL OF COMPUTER SCIENCE
COURSEWORK ASSESSMENT PROFORMA

MODULE & LECTURER: CM0377, Andrew Jones

DATE SET: 8th March 2010

SUBMISSION DATE: 26th April 2010

SUBMISSION ARRANGEMENTS: Hand in at the coursework submission desk on 26th April 2010

TITLE: Exercise 2

This coursework is worth 15% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity. Your work should be submitted using the official Coursework Submission Cover sheet.

INSTRUCTIONS
Answer the first two questions in the attached coursework specification, and optionally the third.

CRITERIA FOR ASSESSMENT

Credit will be awarded against the following criteria.

Ability to demonstrate logical consequence from first principles and by applying resolution, in the context of a relational clausal logic program.
Ability to understand and extend a Prolog program that reasons with Prolog clauses.
Ability to demonstrate effectively that a Prolog program works.

Feedback on your performance will address each of these criteria.

FURTHER DETAILS

Use the laboratory classes to get guidance and assistance in doing this exercise.
It will also be possible to make time during some of the lectures for us to discuss the exercise together.
Please e-mail me if you have any queries you would like to discuss with me individually.

Feedback on your coursework will address the above criteria and will be returned on **6th May 2010**
This will be supplemented with oral feedback via…. **Revision session planned for 6th May 2010**

Knowledge-Based Systems
Exercise 2 (assessed)

Please submit your solutions to this exercise by Monday 26th April 2010. Remember to hand in evidence that these programs work, in addition to the program listings. This exercise is worth *half* of the total continuous assessment marks available for CM0377.

Question 1

[NB This question is worth 40% of the total marks available for this exercise]

Consider the following program *P:*

```
loner(eeyore).
has_friend(tigger, winnie).
:- has_friend(X, X).
has_friend(X, Y):-has_friend(Y, X).
```

and the following clause *C:*

```
has_friend(winnie, tigger)
```

(i) Use proof by refutation to prove *C* from *P.*
(ii) What is the Herbrand Universe of *P?*
(iii) What is the Herbrand Base of *P?*
(iv) Determine which Herbrand Interpretations of *P* are models of *P.*

HINT for part (iv): there are certain elements of the Herbrand Base of *P* that must *obviously* be assigned to **true** in any model of *P,* and some that must obviously be assigned to **false**. Also, there are some atoms where whether they are assigned to **true** or not in a given interpretation makes no difference to whether the interpretation is a model of *P.* You may find that for *every* element of the Herbrand Base in this particular question it is reasonably easy to deduce that either it *must* be **true**, or it *must* be **false**, or that whether it is **true** or **false** *makes no difference*. If so, construction of a truth table will be unnecessary.

(v) Which of these models is a *minimal* model?
(vi) Hence show that *C* is a logical consequence of *P.*

Question 2

[NB This question is worth 45% of the total marks available for this exercise. The meta-interpreter is on Blackboard in the program files for Lecture 9, with name **interp.pl**]

(a) Modify the Prolog meta-interpreter given in lecture 9, slide 26, so that:

(i) a new predicate *commence_proof* is defined which asks the user whether the program clauses are to be interpreted from left to right (as before) or from right to left, and then interprets the program accordingly, and
(ii) if there is no clause matching a given goal *G* the user is asked whether *G* is true. If (s)he replies 'yes' then the proof of *G* should succeed; else it should fail.

(b) Devise a small computer fault diagnosis knowledge base containing rules such as:

```
power_problem:-no_noise, switched_on.
os_problem:-windows_vista_installed.
operator_error_problem:-user_is_andrews_son.
```

and show your meta-interpreter in action with your knowledge base. (NB you have effectively developed a primitive backward-chaining inference engine for an expert system.)

HINTS for question 2:
For part (a)(i) it may be easiest to replace all the *prove* clauses by a set of *prove* clauses for the left-to-right case and another, differently-named set of *prove* clauses for the right-to-left case.

For part (a)(ii), create another clause similar to the long, fourth clause already in the program, but which deals with the case in which `clause(Goal, Body)` does *not* succeed.

Use `read(Term)` to read a term from the keyboard and `write(Term)` to display a term on the screen; `nl` to commence a new line of output. Note that when typing in a term at the keyboard you will need to terminate it with a full stop.

Question 3

[This question is worth 15% of the total marks available for this exercise]
Extend your answer to question 2 so that the answer given regarding any goal *G* is asserted in a clause of the form:

```
answer_to(G, Ans)
```

Your meta-interpreter should be modified to check these clauses first to see whether the answer to a question is already known. If it is already known then the user should not be asked the question again. *commence_proof* should be modified to ask the user whether (s)he wishes to clear answers stored from the previous proof before continuing.

Andrew Jones. 8th March 2010