COSC 4P79 Expert systems **Assignment #3**

Due date: 12:00 noon Friday March 30.
Late date: 12:00 noon, Monday April 2 (-25%).

Objectives: More shell programming.

Hand in: Printouts of shell code and example executions. Examples of modified KB (if any). Note: don't hand in the entire KB. Electronic submission via submit4p79.

1. OOPS: (Merritt ch. 5, appendix C). OOPS is a simple forward chaining system discussed in class. The shell source code and example KB's are available on the 4p79 Web page.

(a) Modify the execution strategy so that a heuristic is used when more than one rule can fire. The heuristic is specificity -- the more conditions there are in a rule, the higher the rule's priority. A rule with more conditions will be selected over a rule with fewer. If more than one rule has a maximum number of conditions, then select the first rule found in the KB (i.e. rule order). You will need to collect a conflict set of all the rules that can fire at some point. Then select the rule from the conflict set according to this criteria. The built-in **findall** predicate is very useful here.

Permit this rule specificity strategy be turned on and off by the user, using a new driver command. If it is off, then use the original inference strategy of OOPS. Also let the specificity heuristic be turned on and off within the rules themselves, as an action command. Test this modification on the forward-chaining 'animal' knowledge base. Make sure there are some rules in which the use of the specificity heuristic will result in different results.

(b) As discussed in class, explanation is difficult to implement for production systems, because the state of the inference is too large and complex to save. However, it is easy to add a **how** utility. Implement a simple **how** that gives a list of rules fired during the entire session**.** Add **how** to the shell command driver (use either of the previous shells above). You don't need to implement how to work for specific goals.

Hint: You already have an automatic trace (predicate **go**). You simply need to save this trace information somehow.

(over)

2. NATIVE: (ch. 2, appendix A). NATIVE was studied in assignment 2.

Take the bird expert system & shell from assignment 2, and modify it so that text explanation is given, instead of the current terse value-attribute pairs. The following are two possible ways in which it might be done. Please implement one of them.

(i) Introduce a phrase in each rule that should substitute for the rule goal in explanation. For example,

```
family(albatross)  :-
        $ 'the family is albatross',
        order(tubenose),
        size(large),
        wings(long_narrow).
```

Here, whenever "family(albatross)" was to be written in explanation, the text beside the new operator "$" is generated instead.

(ii) Two new operators, "is" and "are", are introduced into rules (and possibly others). They are defined with the built-in "op" facility. Using these operators, the rule becomes:

```
family is albatross  :-
        order is tubenose,
        size is large,
        wings are long_narrow.
```

Note that if you replace ":-" with **if**, and "," with **and**, you basically have English text.

For both of the above, when rules are being printed in explanation, text such as "because" or "it follows that" is needed instead of ":-" or bare indentation (see trace history output). You should only print 1 goal or phrase per line of output. The I/O routines might need modification too. Hence an explanation "how" for the above rule may look like:

```
Because the order is tubenose
        and the size is large
        and the wings are long and narrow
it follows that the family is albatross.
```

You do not need to modify the entire bird knowledge base to use text explanation, but just enough so that it can be adequately tested. Note that this explanation style could be of great value in your term project.