Assignment 4

CSU520, Spring 2008
Due: Wednesday, March 19

Part I. Build a small rule-based deduction system that can be used to draw logical inferences on a topic of your choice. For this you need to first create a set of rules and a set of facts like those discussed in class that can be used with the forward and backward chaining programs provided for you in the `programs/logic/predicate/` subdirectory. You should have at least 5 rules and at least 10 facts, and these should permit some nontrivial conclusions to be drawn from them.

Note: These programs assume that a slight variant of the standard Lisp-style logical syntax is used for rules. For example, rules expressed as follows in the strict logical syntax

```
(if (on ?x ?y)
    (above ?x ?y))

(if (and (above ?x ?y)
         (above ?y ?z))
    (above ?x ?z))
```

are instead expressed this way for input to these programs:

```
(if   (on ?x ?y)
 then (above ?x ?y))

(if   (above ?x ?y)
      (above ?y ?z)
 then (above ?x ?z))
```

That is, conjunctive antecedents are not explicitly surrounded by `(and ...)` and `then` is always used to separate all the antecedents from the consequent. Be sure your rules are in this form for these programs or they will not run properly.

First run the forward chainer to derive all possible conclusions from your knowledge base (but don't add the concluded facts to the fact base). Then pose a few interesting queries to your knowledge base using the backward chainer.

Turn in dribble files that document your interaction with the programs. You do not need to turn in any source files with this assignment; instead, show me the rules and facts you have created by evaluating (show-rules) and (show-facts) at appropriate times with dribble active, just as in the examples discussed in class.

More details on how to do this assignment:

First, download `rule-systems.tar.gz` from the `/programs/logic/predicate` subdirectory to your machine and uncompress it. You will see that this creates a directory containing the following 6 files:

```
load-rule-sys.lisp
compile-rule-sys.lisp
fchain.lisp
```

```
bchain.lisp
unify.lisp
rule-utils.lisp
```

along with a few sample data files. (If you are using a Lisp implementation that expects a different extension for Lisp source files, you will have to rename these files accordingly.)

To load the four programs that together implement the overall rule-based deduction system into Lisp, simply evaluate

```
    (load "load-rule-sys")
```

in this directory, and they will then all be loaded in, to run interpreted.

If you want to create compiled versions of these files, simply evaluate

```
    (load "compile-rule-sys")
```

and this should create compiled versions of the four files in this directory that implement the system. Once the files have been compiled, evaluating

```
    (load "load-rule-sys")
```

will then load these compiled files instead, which should improve the running time if that turns out to be an issue.

The sample data files all have names of the form `*-data.lisp`. These are provided to give you examples to experiment with to help you before working with your own data. Simply load any one (or more) of these into the Lisp environment into which you've already loaded the rule systems program.

For this assignment, you should create a single Lisp file containing a single call to `define-rules` and a single call to `define-facts` with all the rules and facts you've created and load it into Lisp after loading the rule system programs. Immediately before or after loading it, evaluate `(dribble "<filename>")` to capture all the output. After activating dribble and loading your facts and rules, immediately evaluate `(show-facts)` and `(show-rules)` so your facts and rules appear near the beginning of your dribble file. Turn in only the dribble file.

Note: The backward chainer has a trace feature that helps show how it works, and the default is for this to be inactive when the program is initialized. To switch it on or off, just evaluate `(setq *trace* t)` or `(setq *trace* nil)`, respectively.

Please make sure the backward chainer trace is switched OFF in the interaction you turn in. I do not want to see lengthy trace output.

Part II. Draw the search tree that the backward chainer would search when answering the question ”What does Mary eat?” given the facts

```
F1:     (imitates mary john)
F2:     (eats john pizza)
```

and rules

```
R1:     (if (eats ?x pizza)
            (eats ?x spaghetti))

R2:     (if (and (imitates ?x ?y)
                 (eats ?y ?z))
            (eats ?x ?z))
```

Be sure to label the arcs of this tree with the appropriate fact and rule numbers, just as was done in class.

You may use the trace output of the backward chainer to help you determine the tree structure, by first creating the appropriate input file and running the backward chainer with trace on. This will give you the correct information when the arc label is a rule number, but be warned that it does not directly show what the arc label should be when a goal unifies with a fact; you will have to figure this out for yourself. (What the program displays is what rule was being checked at that time.)

Part III. Do Exercise 9.18 in the textbook (p. 318) on resolution. To get you started, here is the solution to part (a) using Lisp-style syntax:

```
Premise:    (forall (x) (if (horse x) (animal x)))
Conclusion: (forall (x) (if (exists (y) (and (headof x y) (horse y)))
                            (exists (z) (and (headof x z) (animal z)))))
```

(To understand this, think of the conclusion as being ”Any head of any horse is the head of some animal.”)