Programming Languages Homework 4: Prolog

July 26 2015

Alex Pine

Part 2: More Prolog

1. Reorder the facts (i.e., not the rules) above to provide faster execution time when querying goal(X).

```
foo(roberta).
foo(ashwin).

hello(roberta).
hello(brock).
hello(john).

world(roberta).
world(ashwin).
```

2. Explain in your own words why reordering the facts affects total execution time. Show evidence of the faster execution time (provide a trace for each).

Fulfilling the goal with the original original ordering of the facts required the interpreter to backtrack, as evidenced by the trace below. Reordering the facts so that no backtracking is required results in fewer steps taken, requiring less time to execute.

Original Trace:

```
?- goal(X).
   Call: (7) goal(_G868) ? creep
   Call: (8) sub1(_G868) ? creep
   Call: (9) foo(_G868) ? creep
   Exit: (9) foo(ashwin) ? creep
   Exit: (8) sub1(ashwin) ? creep
   Call: (8) sub2(ashwin) ? creep
   Call: (9) hello(ashwin) ? creep
   Fail: (9) hello(ashwin) ? creep
   Fail: (8) sub2(ashwin) ? creep
```

```
   Redo: (9) foo(_G868) ? creep
   Exit: (9) foo(roberta) ? creep
   Exit: (8) sub1(roberta) ? creep
   Call: (8) sub2(roberta) ? creep
   Call: (9) hello(roberta) ? creep
   Exit: (9) hello(roberta) ? creep
   Call: (9) world(roberta) ? creep
   Exit: (9) world(roberta) ? creep
   Exit: (8) sub2(roberta) ? creep
   Exit: (7) goal(roberta) ? creep
X = roberta.
```

Trace after reordering rules

```
?- goal(X).
   Call: (7) goal(_G868) ? creep
   Call: (8) sub1(_G868) ? creep
   Call: (9) foo(_G868) ? creep
   Exit: (9) foo(roberta) ? creep
   Exit: (8) sub1(roberta) ? creep
   Call: (8) sub2(roberta) ? creep
   Call: (9) hello(roberta) ? creep
   Exit: (9) hello(roberta) ? creep
   Call: (9) world(roberta) ? creep
   Exit: (9) world(roberta) ? creep
   Exit: (8) sub2(roberta) ? creep
   Exit: (7) goal(roberta) ? creep
X = roberta
```

3. Going back to the original ordering of facts, now suppose we rewrite goal sub1 to read: sub1(X) :- foo(X),!. Upon returning to query mode and querying goal(X), the interpreter will display false. Explain why.

The cut operator prevents backtracking. foo(ashwin) succeeds, so the interpreter tries to resolve sub2(ashwin), which fails when it tries to resolve hello(ashwin), because that was not a given fact. Since the cut operator prevents the interpreter from backtracking, it cannot retry unifying sub1(X) with a different fact, so it gives up and returns “false”.

4. Going back to the original ordering of facts, now suppose we rewrite goal sub2 to read: sub2(X) :- hello(X),!,world(X). Upon returning to query mode and querying goal(X), the goal this time will succeed. Explain why.

"roberta" is the only atom that unifies with both foo(X) and hello(X), so by the time the interpreter encounters the cut operator, it has already unified X with "roberta", which also satisfies world(X).

Part 3: Unification. For each pair below that unifies, show the bindings. Circle any pair that doesn't unify and explain why it doesn't.

1. d(15) & c(X)

Does not unify. d and c are different functors, so they do not unify.

2. a(X, b(3, 1, Y)) & a(4, Y)

Unifies as long as the "occurs check" is false.

```
X = 4, Y = b(3, 1, Y).
```

3. a(X, c(2, B, D)) & a(4, c(A, 7, C))

```
X = 4, B = 7, D = C, A = 2.
```

4. a(X, c(2, A, X)) & a(4, c(A, 7, C))

Does not unify. It tries to bind A to 2 and A to 7, and 2 and 7 do not unify.

5. e(c(2, D)) & e(c(8, D))

Does not unify. 2 does not unify with 8.

6. X & e(f(6, 2), g(8, 1))

```
X = e(f(6, 2), g(8, 1)).
```

7. b(X, g(8, X)) & b(f(6, 2), g(8, f(6, 2)))

```
X = f(6, 2).
```

8. a(1, b(X, Y)) & a(Y, b(2, c(6, Z), 10))

Does not unify. The first functor has a different arity than the second.

9. d(c(1, 2, 1)) & d(c(X, Y, X))

```
X = 1, Y = 2.
```