**Part 2: More Prolog**

1. Listed below is the reordered facts that provide faster execution time when queried for goal(X). As asserted facts for “Roberta” will pass for all three facts foo, hello, world it will satisfy the sub goal and avoid unnecessary backtracking.

**foo(roberta).**

**foo(ashwin).**

**hello(roberta).**

**hello(brock).**

**hello(john).**

**world(roberta).**

**world(ashwin).**

**goal(X) :- sub1(X),sub2(X).**

**sub1(X) :- foo(X).**

**sub2(X) :- hello(X),world(X).**

1. **Reasoning for why reordering affects the execution time of the query.**

The order of the execution of the asserted facts definitely plays a role in the execution time. This is because Prolog will try to satisfy all the sub goals for a given instantiated variable. If the sub goal is not satisfied, it will have to backtrack to the next instantiated variable and redo the execution by looking up the call tree. This back and forth of execution control increases the overall execution time.

In our case, the initial ordering of facts has ‘ashwin’ first. But, there are no hello fact for ashwin, which will cause the goal(X) query to execute sub goals for ashwin first (that will fail) and then backtrack to the tree and redo the sub goals execution for ‘roberta’.

Instead, when we reorder the facts in such a manner that facts(roberta) always stated first (because of the assertion of the facts for roberta is true for all three facts foo,hello and world). While executing, the query goal(X), sub goals for roberta will be executed first and results in a less execution time as it satisfies all the sub goals.

**Before Reordering:**

*[trace] 6 ?- trace.*

*true.*

*[trace] 6 ?- goal(X).*

*Call: (7) goal(\_G4270) ? creep*

*Call: (8) sub1(\_G4270) ? creep*

*Call: (9) foo(\_G4270) ? 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(\_G4270) ? 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.*

**After Reordering:**

*[trace] 7 ?- trace.*

*true.*

*[trace] 7 ?- goal(X).*

*Call: (7) goal(\_G398) ? creep*

*Call: (8) sub1(\_G398) ? creep*

*Call: (9) foo(\_G398) ? 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 .*

1. *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.*

This is because of the “CUT” operator. Automatic backtracking is one of the most characteristic features of Prolog. But backtracking can lead to inefficiency. CUT offers a more direct way of exercising control over the way prolog looks for solutions.

In this case, when goal(X). is executed, it starts with executing sub1(Ashwin) and then with the further asserted facts.

*[trace] 9 ?- goal(X).*

*Call: (7) goal(\_G398) ? creep*

*Call: (8) sub1(\_G398) ? creep*

*Call: (9) foo(\_G398) ? 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*

*Fail: (7) goal(\_G398) ? creep*

*false.*

But, eventually, this fails because hello(ashwin) returns false, but Prolog which would otherwise proceed with the backtracking and redo the sub goal for Roberta cannot perform now because the CUT operator forbids it from proceeding with further backtracking and return the whole program execution thus returning false.

1. *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.*

In this case, the hello(ashwin) call fails and thus it does not reach the CUT operator and then Prolog backtracks to hello(Roberta) which succeeds and reaches the CUT operator which exits from further execution and returns true because hello(robeerta) returned true. Here the cut operator will be reached when hello(X) succeeds. This happens for Roberta and not for ashwin. Thus the goal succeeds.

*[trace] 10 ?- goal(X).*

*Call: (7) goal(\_G404) ? creep*

*Call: (8) sub1(\_G404) ? creep*

*Call: (9) foo(\_G404) ? 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(\_G404) ? 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.*