1 The facts have to be reordered in the below order for making the execution of goal(X) faster.

foo(roberta).
foo(ashwin).

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

world(roberta).
world(ashwin).

We are trying to bring up roberta as the first fact for foo, hello and world.

2. Reordering reduces the backtracking that prolog has to do to find matching facts. Since 'roberta' satisfies sub1(X) :- foo(X) and sub2(X) :- hello(X), world(X); bringing the facts corresponding to roberta up, prolog can find the matching facts without backtracking. This can be seen clearly in the trace below.

<u>Before reordering.</u>

```
[trace]  ?- goal(X).
   Call: (7) goal(_G1059) ? creep
   Call: (8) sub1(_G1059) ? creep
   Call: (9) foo(_G1059) ? 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(_G1059) ? 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]  ?- goal(X).
   Call: (7) goal(_G1059) ? creep
   Call: (8) sub1(_G1059) ? creep
   Call: (9) foo(_G1059) ? 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. The built-in Prolog predicate ! (the exclamation mark), called cut, offers a direct way of exercising control over the way Prolog looks for solutions. Cut has two important characteristics.

a. Cut is a goal that always succeeds.
b. Cut commits prolog to any choices that were made since the parent goal was unified with the LHS of the rule.

When we change Sub1 as sub1(X) :- foo(X), !. we define sub1(X) to whatever fact that first satisfy foo(X). Since foo(ashwin) satisfies the criteria, we have X as ashwin. But then the fact ashwin fails for hello. So we get false as the return value.

4. Now that we know what a cut predicate does, the change we made to sub2 can be written as, for whatever X that matches the fact hello, try to find a matching world fact. If no world fact matches the X, then return false. From the given rules, we can easily see that, sub1 will first return ashwin as X, which will fail the hello fact. So sub1 will again be executed for roberta which matches hello and world and hence we get roberta as the value of X. This can also be seen in the trace call stack.

```
[trace]  ?- goal(X).
   Call: (7) goal(_G1059) ? creep
   Call: (8) sub1(_G1059) ? creep
   Call: (9) foo(_G1059) ? 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(_G1059) ? 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. **Unification**
   a. d(15) & c(X)
      false -- Unification not possible as the fact d and c are different.
   b. a(X, b(3, 1, Y)) & a(4, Y)
      X = 4,
      Y = b(3, 1, Y).
   c. a(X, c(2, B, D)) & a(4, c(A, 7, C)).
      X = 4,
      B = 7,
      D = C,
      A = 2.
   d. a(X, c(2, A, X)) & a(4, c(A, 7, C))
      false -- Unification cannot happen as A gets bound to 2 and 7 simultaneously.
   e. e(c(2, D)) & e(c(8, D))
      false -- Unification cannot happen as the constants 2 & 8 doesn't match. Changing 8 to 2 will return true.
   f. X & e(f(6, 2), g(8, 1))
      X gets bind to the whole expression e(ff(6,2), g(8,1)).
   g. b(X, g(8, X)) & b(f(6, 2), g(8, f(6, 2)))
      X gets bound to f(6,2).
   h. a(1, b(X, Y)) & a(Y, b(2, c(6, Z), 10))
      false -- Unification cannot happen as the relation b doesn't have same number of arguments in LHS & RHS.
   i. d(c(1, 2, 1)) & d(c(X, Y, X))
      X = 1,
      Y = 2.