This note describes the trace of SLD resolution, with depth-first search strategy, for append/3 (see file 'append.pl'):

The program contains one rule and one fact:

```
append(cons(S,X),Y,cons(S,Z))  :-  append(X,Y,Z).
append(nil,X,X).
```

The goal (i.e query) is:

```
?-  append(X,Y,cons(a,cons(b,cons(c,nil)))).
```

## SLD Resolution

*Stepwise trace of the SLD resolution algorithm.*

*Step 1* - Select the leftmost subgoal in the current goal (i.e. a literal) and scan the sequence of clauses in the program to find one that matches. The rule is found (position # 1 in the sequence):

```
% Goal
append(X,Y,cons(a,cons(b,cons(c,nil)))).
% Rule
append(cons(S1,X1),Y1,cons(S1,Z1))  :-  append(X1,Y1,Z1).
% MGU
[X=cons(a,X1),  Y=Y1,  S1=a,  Z1=cons(b,cons(c,nil))]
% Resolvent
append(X1,Y1,cons(b,cons(c,nil)))
```

Notice the standardization of variables before applying the rule. The MGU has been derived with the Martelli and Montanari's algorithm.

*Step 2* - The resolvent above becomes the current goal (i.e. depth-first strategy). The leftmost subgoal is the only literal in it. Scan the sequence of clauses in the program to find one that matches:

```
% Goal
append(X1,Y1,cons(b,cons(c,nil)))
% Rule
append(cons(S2,X2),Y2,cons(S2,Z2))  :-  append(X2,Y2,Z2).
% MGU
[X1=cons(b,X2),  Y1=Y2,  S2=b,  Z2=cons(c,nil)]
% Resolvent
append(X2,Y2,cons(c,nil))
```

*Step 3* - The resolvent above becomes the current goal. Scan the sequence of clauses in the program to find one that matches:

```
% Goal
append(X2,Y2,cons(c,nil)))
```

```
% Rule
append(cons(S3,X3),Y3,cons(S3,Z3)) :- append(X3,Y3,Z3).
% MGU
[X2=cons(c,X3), Y2=Y3, S3=c, Z3=nil]
% Resolvent
append(X3,Y3,nil)
```

*Step 4* - The resolvent above becomes the current goal. Scan the sequence of clauses in the program to find one that matches, notice that in this case the rule does not match, so the fact is attempted (position # 2 in the sequence):

```
% Goal
append(X3,Y3,nil)
% Fact
append(nil,X4,X4).
% MGU
[X3=nil, Y3=nil, X4=nil]
% Resolvent
{ }
```

Success. Apply all MGUs to each other, in reverse order, up to step 1. This yields the first solution:

```
% Answer
[X=cons(a,cons(b,cons(c,nil))), Y=nil]
```

*Step 5* -By pressing semicolon (;) a *bactracking* is forced. Go back to step 4 and try finding another clause that matches the goal. This action fails, as there are no further clauses in the program. Backtrack to step 3 and apply the fact, instead :

```
% Goal
append(X2,Y2,cons(b,cons(c,nil)))
% Fact
append(nil,X5,X5).
% MGU
[X2=nil, Y2=cons(c,nil), X5=cons(c,nil)]
% Resolvent
{ }
```

Success. Apply all MGUs to each other, in reverse order, up to step 1. This yields the second solution:

```
% Answer
[X=cons(a,cons(b,nil)), Y=cons(c,nil)]
```

*Step 6* -By pressing semicolon (;) again, another *bactracking* is forced. Go back to step 5 and try finding another clause that matches the goal. This action fails, as there are no further clauses in the program. Backtrack to step 3 and try finding another clause. This fails also. Backtrack to step 2 and apply the fact, instead :

```
% Goal
append(X1,Y1,cons(b,cons(c,nil)))
% Fact
append(nil,X6,X6).
% MGU
[X1=nil, Y1=cons(b,cons(c,nil)), X6=cons(b,cons(c,nil))]
% Resolvent
{ }
```

Success. Apply all MGUs to each other, in reverse order, up to step 1. This yields the third solution:

```
% Answer
[X=cons(a,nil), Y=cons(b,cons(c,nil))]
```

*Step 7* -By pressing semicolon (;) again, another *bactracking* is forced. Go back to step 6 and try finding another clause that matches the goal. This action fails, as there are no further clauses in the program. Backtrack to step 2 and try finding another clause. This fails also. Backtrack to step 1 and apply the fact, instead :

```
% Goal
append(X,Y,cons(a,cons(b,cons(c,nil)))).
% Fact
append(nil,X7,X7).
% MGU
[X=nil, Y=cons(a,cons(b,cons(c,nil))), X7=cons(a,cons(b,cons(c,nil)))]
% Resolvent
{ }
```

Success. The MGU above contains the fourth solution.