Tutorial 7

November 8
1 Prolog programming example - family tree

male(tom).
male(peter).
male(doug).
male(david).
female(susan).
parent(doug, susan).
parent(tom, william).
parent(doug, david).
parent(doug, tom).

1) mother(X, Y) :- parent(X, Y), female(X).

2) father(X, Y) :- parent(X, Y), male(X).

3) sibling(X, Y) :- parent(Z, X), parent(Z, Y).

4) grandfather(GP, GC) :- male(GP), parent(GP, X), parent(X, GC).

5) second_cousin(X, Y) :-
   grandchild(X, Z),
   grandchild(Y, V),
   sibling(Z, V).
2  Prolog programming example - Trip planning

Here we define a new kind of database to deal with "trips", and develop Prolog predicates to compute certain things about the trips. We start with simply facts such as the following:

plane(to, ny).
plane(ny, london).
plane(london, bombay).
plane(london, oslo).
plane(bombay, katmandu).
boat(oslo, stockholm).
boat(stockholm, bombay).
boat(bombay, maldives).

We now develop the following predicates:

a) cruise(X,Y) -- there is a possible boat journey from X to Y.
   
   cruise(X,Y) :- boat(X,Y).
   cruise(X,Y) :- boat(X,Z), cruise(Z,Y).

b) trip(X,Y) -- there is a possible journey (using plane or boat) from X to Y.
   
   leg(X,Y) :- plane(X,Y).
   leg(X,Y) :- boat(X,Y).
   
   trip(X,Y) :- leg(X,Y).
   trip(X,Y) :- leg(X,Z), trip(Z,Y).

Note how we use multiple clauses for or'ing subgoals. Note that an advantage of using "leg" is that it makes it easier to extend the knowledge base to have other modes of transport.

c) stopover(X,Y,S) -- there is a trip from X to Y with a stop in S.

First, assume that neither X nor Y can equal S.

   stopover(X,Y,S) :- trip(X,S), trip(S,Y).
Now, assume $S$ could be $X$ or $Y$ (or even both):

\[
\begin{align*}
hop(X, X). \\
hop(X, Y) &:- \text{trip}(X, Y). \\
\text{stopover}(X, Y, S) &:- \text{hop}(X, S), \text{hop}(S, Y).
\end{align*}
\]

d) $\text{plane_cruise}(X, Y)$ -- there is a trip from $X$ to $Y$ that has at least one plane leg, and at least one boat leg.

\[
\begin{align*}
\text{plane_cruise}(X, Y) &:- \text{plane}(X, Z), \text{boat}(Z, Y). \\
\text{plane_cruise}(X, Y) &:- \text{boat}(X, Z), \text{plane}(Z, Y).
\end{align*}
\]

\[
\begin{align*}
\text{plane_cruise}(X, Y) &:- \text{leg}(X, Z), \text{plane_cruise}(Z, Y). \\
\text{plane_cruise}(X, Y) &:- \text{leg}(Z, Y), \text{plane_cruise}(X, Z).
\end{align*}
\]

The interesting thing about this solution is to see that to get a "mixed" trip of planes and boats, you will at some point have a plane followed by a boat or vice versa (the base cases). Once you have that, the condition is met, and you can simply add either plane or boat legs on either side to create the full journey.

Think about why we need to have the second rule of $\text{plane_cruise}$ be

\[
\begin{align*}
\text{plane_cruise}(X, Y) &:- \text{leg}(Z, Y), \text{plane_cruise}(X, Z).
\end{align*}
\]

instead of:

\[
\begin{align*}
\text{plane_cruise}(X, Y) &:- \text{plane_cruise}(X, Z), \text{leg}(Z, Y).
\end{align*}
\]

The latter, while it may be the intuitive way to write it, gives an infinite recursion!!!

e) $\text{cost}(X, Y, C)$ -- there is a trip from $X$ to $Y$ that costs less than $C$.

We need to add costs to each of the plane and boat predicates, such as plane(to, ny, 300).

\[
\begin{align*}
\text{leg}(X, Y, C) &:- \text{plane}(X, Y, C). \\
\text{leg}(X, Y, C) &:- \text{boat}(X, Y, C).
\end{align*}
\]

\[
\begin{align*}
\text{trip}(X, Y, C) &:- \text{leg}(X, Y, C). \\
\text{trip}(X, Y, C) &:- \text{leg}(X, Z, C_1), \text{trip}(Z, Y, C_2), C \text{ is } C_1 + C_2.
\end{align*}
\]

Now "cost" is simple, because "trip" is doing the addition for us:

\[
\begin{align*}
\text{cost}(X, Y, C) &:- \text{trip}(X, Y, C_{\text{trip}}), C_{\text{trip}} < C.
\end{align*}
\]
Note that we have made use of arithmetic in our solution. For now, you should know the following:

1. `is` acts like mathematical equality, with the restriction that everything on the right-hand side of `is` must be instantiated (no variables!)

2. `+`, `−`, `<`, `>` are all defined in the usual way
3 The Prolog search tree

Need to number the predicates. We also change the database a bit, so that our tree is not too big.

1 parent(doug, susan).
2 parent(tom, william).
3 parent(doug, tom).

4 sibling(X, Y) :- parent(Z, X), parent(Z, Y).

?- sibling(X, Y).

X = susan
Y = susan

See Figure 1 for a snapshot. Now we type ",;".
sibling (X, Y)

4  X=X, Y=Y

parent(Z, X), parent(Z,Y)

parent(doug,Y)

\[\begin{array}{l}
\text{Z=doog} \\
\text{X=susan}
\end{array}\]

\[\begin{array}{l}
1 \\
Y=susan \\
success \\
3 \\
Y=tom \\
success
\end{array}\]

Figure 2: X=susan Y=tom

\[\begin{array}{l}
X = \text{susan} \\
Y = \text{tom}
\end{array}\]

See Figure 2 for a snapshot. Now we type ",".
X = william
Y = william

See Figure 3 for a snapshot. Now we type ",".
Figure 4: $X=\text{tom}$ $Y=\text{tom}$

$x = \text{tom}$
$y = \text{susan}$

See Figure 4 for a snapshot. Now we type ",, ."
Figure 5: No

\[ X = \text{tom} \]
\[ Y = \text{tom} \]

See Figure 5 for a snapshot. Now we type ",".
Figure 6: X=susan Y=tom

No
?

See Figure 6 for a snapshot. Now we type ",".