1 Lab

```
buy( Person, Thing) :-
   needs( Person, Thing),
   price( Thing, Price),
   can_pay( Person, Price).

price(cornflakes,2).
price(frosties,3).
price(branflakes,2).
price(chocolate_cake,1).

needs(antonia, frosties).
needs(mark, branflakes).

can_pay(antonia,4).
can_pay(mark,2).
```

Queries :

buy(mark,frosties)
buy(mark,_)

2 The ladder rule

- Everytime a ladder encountered, go *down it*, and to the *right* as far as possible (still going down any ladder encountered)
- If can't go further right, remember the current 'floor', climb back up to the top and keep on going right
- If something fails, backtrack - i.e. go back to last ladder encountered; go down it to the *next* floor, after the one last visited, and then back up to the right

3 Rules of Execution of Prolog programs

1. Put all goals in the query into a list of goals to be satisfied
2. When all goals are satisfied, execution stops
3. Take first goal (left most), and search program in a top-down manner until a clause whose head matches the goal is satisfied
4. If such a clause exists :
   Make a variant of the clause
   Match the head of the variant with the goal
   Put Goals from the body of the variant at the beginning of the list of goals to be satisfied - go to 3
5. If such a clause does not exist :
   Uninstantiate all the variables that were instantiated in the last matching
   Return to the last successfully matched goal (climbing up the ladder)
   Search for another clause whose head matches the goal and go to 4

If the no *previously matched goals* criteria in (4) above can be satisfied, then the goals in question cannot be satisfied, and execution stops

4 Forcing Backtracking

```
miles(sketches_of_spain)
miles(a_kind_of_blue)
```

- If Prolog is asked
  miles(X)

  it will answer with :
  X = sketches_of_spain
- However, we may be interested in obtaining alternate answers
- To obtain another answer, we *force* Prolog to invoke *backtracking*
- Prolog continues execution by failing the last goal that was satisfied and by backtracking as in normal execution - to find another clause whose head matches goal
- Hence :
  miles(X)
  it will answer with :
  X = sketches_of_spain
  X = a_kind_of_blue
  no

5 Different types of equality

```
X = Y
(X is equal to Y, if X and Y match)
X == Y
(X is literally equal to Y, if X and Y
 are identical)
X \== Y
(X is not literally equal to Y - for
 arbitrary terms X and Y)
X =:= Y
(the value of X equals the value of Y)
X =\= Y
(the value of X is not equal to the
 value of Y)
X is Y
(if X matches the value of Y, where X
 is a variable or a constant, and Y is
 an arithmetic expression)
```

6 Membership

```
member1( X, [ X | _]).
member1( X, [ _ | Tail]) :-
 member1( X, Tail).

member2( X, [ _ | Tail]) :-
 member2( X, Tail).
member2( X, [ X | _]).
```

Consider the execution traces for each, when presented with :

- member1(2,[1,2,3,4])
- member2(2,[1,2,3,4])

Which is better ?

Q : holiday(Student), party(Student)
A : jane
no more solutions

Search Tree for this

7 Mamma's and Pappa's

```
ancestor(X,A) :-
  parent(X,A).

ancestor(X,A) :-
  parent(P,X),
  ancestor(P,A).

parent(a,b).
parent(b,c).
```

Q : ancestor(X,A)
A : X = b, A = a
X = c, A = b,
X = c, A = a
no more solutions

If the ancestor relation is defined differently :

```
ancestor(X,A) :-
 ancestor(P, A),
 parent(P,X).

ancestor(X,A) :-
 parent(A,X).

parent(a,b).
parent(b,c).
```

Q : ancestor(X,A)
A : (WAIT!)

- What is happening in this case ?
- Is computation taking place, or are we in a loop ?

8 Golden Rules of recursion

- Declare the **BASE CASE(S)** of a recursion relation **BEFORE** the recursive case(s).
- Make sure that the recursive call within the body of the relation will have arguments that differ from the head of the relation at the moment of execution

  This depends on the type of arguments used in the goal, e.g. in the last query (ancestor) - only had variables as arguments