Lecture 18

GProlog
Search and Backtracking

fun(X) :-
    red(X),
    car(X).

fun(X) :-
    blue(X),
    bike(X).

red(apple_1).
red(block_1).
red(car_27).
blue(monkey).

car(desoto_48).
car(car_27).
bike(apple_1).

?- fun(X).

Materials adapted from Geraint Wiggins
Search and Backtracking

\[
\text{a}(X) :- \\
\text{b}(X), \text{c}(X), \text{d}(X).
\]

\[
\text{a}(X) :- \\
\text{c}(X), \text{d}(X).
\]

\[
\text{a}(X) :- \\
\text{d}(X).
\]

\[
\text{b}(1). \\
\text{b}(a). \\
\text{b}(2). \\
\text{b}(3). \\
\text{d}(10). \\
\text{d}(11). \\
\text{c}(3) \\
\text{c}(4)
\]

\[- \text{a}(X). \\
\text{Materials adapted from Geraint Wiggins}\]
Recursion

Determine whether you can reach Rome from a location on_route(rome).

Materials adapted from Geraint Wiggins
Recursion

Determine whether you can reach Rome from a location.

```
on_route(rome).

on_route(Place):-
    move(Place,Method,NewPlace),
    on_route(NewPlace).

move(home,taxi,halifax).
move(halifax,train,gatwick).
move(gatwick,plane,rome).
```
Recursion

parent(john,paul).  /* paul is john's parent */
parent(paul,tom).   /* tom is paul's parent */
parent(tom,mary).   /* mary is tom's parent */

Materials adapted from Geraint Wiggins
Recursion

parent(john,paul). /* paul is john's parent */
parent(paul,tom). /* tom is paul's parent */
parent(tom,mary). /* mary is tom's parent */

ancestor(X,Y):- parent(X,Y). /* someone is your ancestor if there are your parent */

ancestor(X,Y):- parent(X,Z), /* or somebody is your ancestor if they are the parent */
    ancestor(Z,Y). /* of someone who is your ancestor */

Materials adapted from Geraint Wiggins
Recursion

town1---->----town2---->----town3---->----town4---->----town5---->----town6

A one way road links 6 towns. Write a program that can work out if you can travel on that road. For example. Here are two sample program behaviours.

?- can_get(town2, town5).

yes

?- can_get(town3, town1).

no

Materials adapted from Geraint Wiggins
Consider the following fact:

\[ p([H|T], H, T). \]

?- \( p([a, b, c], X, Y). \)
\( X = a \)
\( Y = [b, c] \)
yes

?- \( p([a], X, Y). \)

?- \( p([], X, Y). \)

**Materials adapted from Geraint Wiggins**
List

How do you determine whether an element is on a list of not:

? - on(a, [1, 2, 3, a, b, c]).

Materials adapted from Geraint Wiggins
List

How do you determine whether an element is on a list of not:

?- on(a, [1, 2, 3, a, b, c]).

on(Item, [Item|_]).
List

How do you determine whether an element is on a list of not:

?- on(a, [1, 2, 3, a, b, c]).

on(Item, [Item|_]).

on(Item, [_|Rest]) :- on(Item, Rest).

Materials adapted from Geraint Wiggins
List

How do we produce a predicate that combines two lists:

?- append([1, 2, 3], [a, b, c], Result).
Result=[1, 2, 3, a, b, c]

append([], Listb, Listb).

Materials adapted from Geraint Wiggins
List

How do we produce a predicate that combines two lists:

?- append([1, 2, 3], [a, b, c], Result).
Result=[1, 2, 3, a, b, c]

append([], Listb, Listb).

append([H|Rest], Listb, [H|Result]) :- append(Rest, Listb, Result).

Materials adapted from Geraint Wiggins