Logic Programming

Examples

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria
`kutsia@risc.jku.at`

Outline

Outline

`repeat`

- An extra way to generate multiple solutions through backtracking.
- Comes as a built-in.
- Easy to define:

```
repeat.
repeat :- repeat.
```

`repeat`

Effect:

- If placed in a goal, `repeat` will succeed because of the first fact.
- If after some time backtracking reaches this point in the goal again, the rule for `repeat` will be tried.
- The rule generates the goal `repeat`, which will be satisfied by the first fact.
- If backtracking reaches here again, Prolog will again use the rule it used the fact before.
- To satisfy the generated `repeat` goal, it will use the fact again, and so on.

```
repeat.
repeat :- repeat.
```

How to use `repeat`

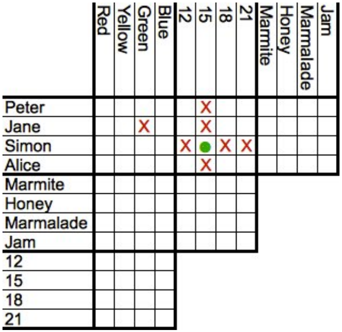
- `repeat` can be useful to organize an interaction with the user.

```
interact :-
    repeat,
        write('Please enter an integer.
               To stop, type 'stop'.'), nl,
        read(I),
    do_something_with_I(I),
    !.

do_something_with_I(stop) :-
    !.
do_something_with_I(I) :-
    integer(I),
    ...
    !,
    fail.
```

Outline

Solving Logic Puzzles

- Logic grid puzzles.
- Given: The set-up to a scenario and certain clues.
- Goal: To find an object (e.g. who owns zebra), or to fill in the missing knowledge.
- Usually given in the form of a grid to be filled in.



	Red	Yellow	Green	Blue	12	15	18	21	Marmite	Honey	Marmalade	Jam
Peter						X						
Jane			X			X						
Simon					X	●	X	X				
Alice						X						
Marmite												
Honey												
Marmalade												
Jam												
12												
15												
18												
21												

Solving Logic Puzzles

- Logic grid puzzles can be easily solved by logic programming.
- Idea: generate-and-test.
- Generate a possible solution.
- Test whether it is really a solution (whether it satisfies all the constraints imposed by the puzzle).
- If yes, finish.
- If not, generate another possible solution and test again.
- And so on.

Solving Logic Puzzles

Example (From www.logic-puzzles.org)

- Figure out the reservation, first name, superhero and language for each person using the clues given.

- Reservations: 5:00pm, 5:30pm, 6:30pm, 7:00pm
- First Names: Caleb, Emanuel, Johnathan, Karen
- Superheros: Batman, Hellboy, Iron Man, Spiderman
- Languages: ASP, Cold Fusion, PHP, Python

Solving Logic Puzzles

Example (Cont.)

Clues:

1. The Batman fan is not Caleb.
2. Of Karen and Caleb, one specializes in PHP applications and the other has the 5:30pm reservation.
3. The Hellboy fan has an earlier reservation than the PHP programmer.
4. Emanuel isn’t well-versed in Python or Cold Fusion.
5. The person with a reservation at 7:00pm specializes in Cold Fusion applications.
6. The Spiderman fan is Karen.

Solving Logic Puzzles

Example (Cont.)

Clues:

7. The ASP programmer doesn't care for Spiderman and is not Karen.
8. Either the Cold Fusion programmer or the PHP programmer collects anything even remotely related to Iron Man.
9. Caleb doesn't care for Iron Man and doesn't have the 6:30pm reservation.
10. The ASP programmer is not Johnathan.
11. The PHP programmer doesn't care for Iron Man.
12. The Spiderman fan has an earlier reservation than the Cold Fusion programmer.

Solving Logic Puzzles

- To generate a possible solution, the information about reservations, first names, superheros, and languages are used.
- Clues are for testing.
- The program should follow this structure.

See the ▸ program at the course Web page.

Outline

`Findall`

Determine all the terms that satisfy a certain predicate.

`findall(X,Goal,L)`: Succeeds if `L` is the list of all those `X`'s for which `Goal` holds.

Example

```
?- findall(X, member(X,[a,b,a,c]), L).

L = [a,b,a,c]

?- findall(X, member(X,[a,b,a,c]), [a,b,c]).

false.
```

More Examples on `Findall`

Example

```
?- findall(X, member(5,[a,b,a,c]), L).
L = []
?- findall(5, member(X,[a,b,a,c]), L).
L = [5,5,5,5]
```

More Examples on `Findall`

Example

```
?- findall(5, member(a,[a,b,a,c]), L).
L = [5,5]
?- findall(5, member(5,[a,b,a,c]), L).
L = []
```

Implementation of `Findall`

`findall` is a built-in predicate.

However, one can implement it in PROLOG as well:

```
findall(X, G, _) :-
    asserta(found(mark)),
    call(G),
    asserta(found(X)),
    fail.

findall(_, _, L) :-
    collect_found([], M),
    !,
    L=M.
```

```
collect_found(S, L) :-
    getnext(X),
    !,
    collect_found([X|S], L).

collect_found(L, L).

getnext(X) :-
    retract(found(X)),
    !,
    X \== mark.
```

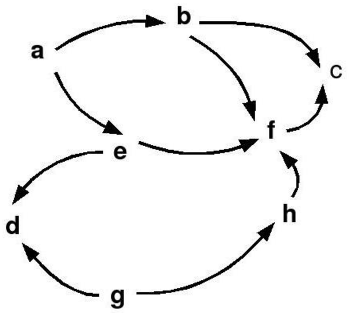
Sample Runs

```
?- findall(X, member(X,[a,b,c]), L).

L = [a,b,c]

?- findall(X, append(X,Y,[a,b,c]), L).

L = [[],[a],[a,b],[a,b,c]]

?- findall([X,Y], append(X,Y,[a,b,c]), L).

L = [[[],[a,b,c]], [[a],[b,c]], [[a,b],[c]],
[[a,b,c],[]]]
```

Outline

Representing Graphs

```
a(g,h).
a(g,d).
a(e,d).
a(h,f).
a(e,f).
a(a,e).
a(a,b).
a(b,f).
a(b,c).
a(f,c).
```



Moving Through Graph

Simple program for searching the graph:

-
  ```
  go(X, X).
  go(X, Y) :- a(X, Z),go(Z, Y).
  ```
- Drawback: For cyclic graphs it will loop.
- Solution: Keep trial of nodes visited.

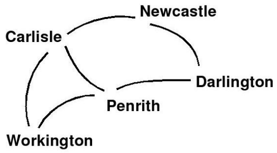
Improved Program for Graph Searching

`go(X, Y, T)`: Succeeds if one can go from node `X` to node `Y`. `T` contains the list of nodes visited so far.

```
go(X, X, T).
go(X, Y, T) :-
    a(X, Z),
    legal(Z, T),
    go(Z, Y, [Z|T]).

legal(X, []).
legal(X, [H|T]) :-
    X \= H,
    legal(X, T).
```

Car Routes

```
a(newcastle, carlisle).
a(carlisle, penrith).
a(darlington, newcastle).
a(penrith, darlington).
a(workington, carlisle).
a(workington, penrith).
```



Car Routes Program

```
go(Start, Dest, Route) :-
    go0(Start, Dest, [], R),
    reverse(R, Route).

go0(X, X, T, [X|T]).
go0(Place, Dest, T, Route) :-
    legalnode(Place, T, Next),
    go0(Next, Dest, [Place|T], Route).
```

Car Routes Program, Cont.

```
legalnode(X, Trail, Y) :-
    (a(X, Y) ; a(Y, X)),
    legal(Y, Trail).

legal(_, []).
legal(X, [H|T]) :-
    X \= H,
    legal(X, T).

reverse(L1, L2) :- reverse(L1, [], L2).

reverse([X|L], L2, L3) :-
    reverse(L, [X|L2], L3).
reverse([], L, L).
```

Runs

```
?- go(darlington, workington, X).

X = [darlington,newcastle,carlisle,
     penrith,workington];

X = [darlington,newcastle,carlisle,
     workington];

X = [darlington,penrith,carlisle,workington];

X = [darlington,penrith,workington];

false.
```

Deficiencies of the Program

- Can not survey the complete list of possibilities.
- Remained options are implicit in the backtracking structure of Prolog.
- They are not explicit in the structure that the program examines.

Deficiencies of the Program

- Can not survey the complete list of possibilities.
- Remained options are implicit in the backtracking structure of Prolog.
- They are not explicit in the structure that the program examines.
- We try to come up with a more general-purpose solution.

Findall Paths

```
go(Start, Dest, Route) :-
    go1([[Start]], Dest, R),
    reverse(R, Route).

go1([First|Rest], Dest, First) :-
    First = [Dest|_].
go1([[Last|Trail]|Others], Dest, Route) :-
    findall([Z,Last|Trail],
        legalnode(Last, Trail, Z),
        List),
    append(List, Others, NewRoutes),
    go1(NewRoutes, Dest, Route).
```

Depth First

```
?- go(darlington, workington, X).

X = [darlington,newcastle,
     carlisle,penrith,workington];

X = [darlington,newcastle,
     carlisle,workington];

X = [darlington,penrith,
     carlisle,workington];

X = [darlington,penrith,workington];

false.
```

Depth, Breadth First

```
go1([[Last|Trail]|Others], Dest, Route) :-
    findall([Z,Last|Trail],
        legalnode(Last, Trail, Z),
        List),
    append(List, Others, NewRoutes),
    go1(NewRoutes, Dest, Route).

go1([[Last|Trail]|Others], Dest, Route) :-
    findall([Z,Last|Trail],
        legalnode(Last, Trail, Z),
        List),
    append(Others, List, NewRoutes),
    go1(NewRoutes, Dest, Route).
```

Breadth First

```
?- go(darlington,workington,X).
X = [darlington,penrith,workington];
X = [darlington,newcastle,
     carlisle,workington];
X = [darlington,penrith,
    carlisle,workington];
X = [darlington,newcastle,
    carlisle,penrith,workington];
false.
```