Programming Paradigms CSI2120 – Winter 2018

Jochen Lang

EECS, University of Ottawa

Canada

Logic Programming in Prolog

- **Advanced Examples**
  - Collecting solutions of a Goal
  - River crossing puzzle

Built-in Predicates bagof/3 and setof/3

- **bagof/3 finds all the solution and enters them in a list**
- **Example**

```
grade(ana,5).
grade(heather,4).
grade(liz,5).
```

- **Queries**

```
?- bagof(N,grade(N,5),L).
L=[ana,liz].
?- bagof([N,G],grade(N,G),L).
L=[[ana,5],[heather,4],[liz,5]].
```

- **setof/3 is similar but eliminates duplicates and sorts the result**

Example: Bag of Numbers

- **Database**

```
bag(2,4,1).
bag(3,5,2).
bag(7,8,2).
bag(4,3,1).
bag(5,2,4).
bag(2,1,4).
bag(2,2,4).
bag(7,3,5).
bag(7,3,3).
```

- **Queries**

```
?- bagof(Z,bag(X,Y,Z),B).
?- bagof(Z,(bag(X,Y,Z),Z>2),B).
```

- **Not binding a variable in a goal with the existential operator ^**

```
?- bagof(Z,X^bag(X,Y,Z),B).
?- setof(Z,X^bag(X,Y,Z),B).
?- bagof(Z,X^Y^bag(X,Y,Z),B).
```

- **Not binding any variable in the goal (same as one line above).**

```
?- findall(Z,bag(X,Y,Z),B).
```

Example: Street Turns

- **Database:**

```
turn(elgin,wellington).
turn(elgin,catherine).
turn(elgin,laurier).
turn(qed,laurier).
turn(qed,bank).
turn(bank,qed).
turn(bank,sommerset).
turn(bank,gladstone).
turn(bank,wellington).
```

- **Queries:**

```
?- setof(X,turn(X,Y),B).
?- setof(Y,turn(X,Y),B).
?- setof(Y,X^turn(X,Y),B).
?- bagof(Y,X^turn(X,Y),B).
?- setof([X,Y],turn(X,Y),B).
```

Example: More grades

- **Database**

```
grade(nick,8).
grade(rachel,4).
grade(peter,3).
grade(monica,7).
grade(samantha,4).
```

- **Queries:**

```
?- setof(A,N^grade(N,A),B).
?- setof(A,N^grade(N,A),[H|T]).
?- setof(A,N^grade(N,A),[H|_]).
?- setof([A,N],grade(N,A),[[_,J]|_]).
?- grade(P,A1),\+((grade(_,A2),A2<A1)).
```

River Crossing Puzzle

- **The fox, chicken, and bag of grain puzzle**
  - a farmer must cross a river transporting a fox, a chicken, and a bag of grain
  - the farmer has a boat which can hold besides him only one of them, either the fox, the chicken, or the bag of grain
  - the fox will eat the chicken if the fox is left with the chicken on one side of the river without the farmer
  - the chicken will eat the bag of grain if the chicken is left with the bag of grain on one side of the river without the farmer

State of the Puzzle

- **We need to represent the location of the farmer, the fox, the chicken, and the bag of grain during the puzzle.**
  - use a state vector of size 4

    `[Farmer,Fox,Chicken,Bag_of_Grain]`
- **State at the beginning of the puzzle**

    `[left,left,left,left]`
- **Goal state at the end of the puzzle**

    `[right,right,right,right]`

Puzzle: Enumerate the Crossings

- **Facts about possible crossings**
  - state before and after
  - add a name for the state transition
- **Farmer by himself**

```
cross(state([left,X,Y,Z]),state([right,X,Y,Z]),
      farmer_cross).
cross(state([right,X,Y,Z]),state([left,X,Y,Z]),
      farmer_returns).
```

- **Farmer with the chicken**

```
cross(state([left,X,left,Z]),state([right,X,right,Z]),
      farmer_brings_chicken).
cross(state([right,X,right,Z]),state([left,X,left,Z]),
      farmer_returns_chicken).
```

- **Farmer with the fox and with the bag of grain as above**

```
cross(state([left,left,X,Y]),state([right,right,X,Y]),
      farmer_brings_fox).
```

etc.

Puzzle: Forbidden States and Top Level

- **Forbidden states (i.e., states where the fox eats the chicken or the chicken the bag of grains).**
  - e.g., fox and chicken on left side and farmer on right side or the other way round
    ```
    forbidden(state([right, left, left, _])).
    forbidden(state([left, right, right, _])).
    ```
  - Can be combined to:
    ```
    forbidden(state([X, Y, Y, _])) :- X \== Y.
    ```
  - And similarly (chicken and bag of grain):
    ```
    forbidden(state([X, _, Y, Y])) :- X \== Y.
    ```
- **Top level**

```
puzzle(P) :- initial(StartState),
             final(EndState),
             crossRiver(StartState, EndState, P).
```

Puzzle: Searching for a "Plan"

- **State transitions necessary to go from state A to B will require a sequence of crossings**
  - Staying in the same state requires no crossings

```
crossRiver(A,A,[]).
```

- **Use Prolog's depth first search; rule out invalid states**

```
crossRiver(A,B,P) :-
          cross(A,C,Action),
          not(forbidden(C)),
          crossRiver(C,B,Plan),
          P = [Action|Plan].
```

- **This will fail. Why?**

Puzzle: Searching for a "Plan"

- **Loops in the search**
  - E.g., the farmer and chicken cross back and forth an infinite number of times, alternating between
    `[left,X,left,X]` and `[right,X,right,X]`
- **Solution: Rule out loops, i.e., only go to states that we have not visited yet**
  - need to record a list of states that we have visited

```
crossRiver(A,A,_,[]).
crossRiver(A,B,V,P) :-
            cross(A,C,Action),
            not(forbidden(C)),
            not(member(C,V)),   % built-in
            crossRiver(C,B,[C|V],Plan),
            P = [Action|Plan].
```

Summary

- **Advanced Examples**
  - Collecting solutions of a Goal
    - `findall/3`
    - `bagof/3`
    - `setof/3`
  - Prime numbers
  - River crossing puzzle