**DFS with prolog - Unification as Search**

**Example**: The [farmer, wolf, goat, and cabbage problem (FWGC):](http://www.mathcats.com/explore/river/crossing.html)

*A farmer with his wolf, goat, and cabbage come to the west side of a river they wish to cross. There is a boat at the river’s edge, but, of course, only the farmer can row it. The boat can carry two things (including the rower) at a time. If the Wolf is ever left alone with the goat, with or without the cabbage, the wolf will eat the goat. Similarly, if the goat is left alone with the cabbage, the goat will eat the cabbage. Device a sequence of actions such that all four characters arrive safely on the east side of the river.*

<http://watson.latech.edu/WatsonRebootTest/ch14s3p1.html>

**Data Structure**

Each problem state can be represented as [FL, WL, GL, CL], where $L is the location of the character $. Ex., the initial state could be represented by [w, w, w, w] indicating everyone is on the w (west) side

GOAL

fwgc([w,w,w,w], [e,e,e,e],SP). % fwgc(StartState, GoalState, SolutionPath)

Preparations:

opposite(e, w).

opposite(w, e).

The moves: we define a functor named move/2

move(Current, Next) is true (i.e., acceptable) if

Next = Current + action

^

Farmer takes self,

Farmer takes wolf

Farmer takes goat

Farmer takes cabbage

and Next is safe or not(unsafe).

Which one is easier to describe, safe or unsafe? unsafe

Ex.,

?- move([e, e, w, e], Next).

Next=[w,e,w,e] ;

Next=[w,w,w,e] ;

Next=[w,e,w,w] ;

false

% 1. Farmers take self

%move([FL,WL,GL,CL], [FL1, WL,GL, CL])

%:- opposite(FL, FL1), not(unsafe([FL1, WL,GL,CL]).

% 2. Farmers take wolf

%move([FL,FL,GL,CL], [FL1, FL1,GL,CL])

%:- opposite(FL,FL1), not(unsafe([FL1, FL1, GL,CL]) .

Defining unsafe – specify the state that are unsafe, one state at a time

In all of the following cases, the goat eats the cabbage

unsafe([e,e,w,w]).

unsafe([w,w,e,e]).

unsafe([e,w,w,w]).

unsafe([w,e,e,e]).

unsafe([FL, \_, GL, GL]) :- not(FL=GL).

unsafe([FL, WL, WL, \_\_]) :- opposite(FL, WL).

Solving the puzzle

?- fwgc([e,e,e,e],[w,w,w,w],Path).

Path=[[e,e,e,e],[w,e,w,e],[e,e,w,e],[w,w,w,e],[e,w,e,e],[w,w,e,w],[e,w,e,w],[w,w,w,w]]

Path=[[e,e,e,e],[w,e,w,e],[e,e,w,e],[w,e,w,w],[e,e,e,w],[w,w,e,w],[e,w,e,w],[w,w,w,w]]

This is relatively simple!

% base case – solution found, current is goal, blah, blah, yada, yada

fwgc1(Current, GoalState, PathSoFar, SolutionPath) :-

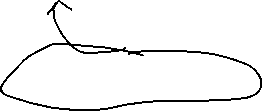
Current=GoalState, reverse(PathSoFar, SolutionPath).

% recursive case, making moves, one at a time.

fwgc1(Current, GoalState, PathSoFar, SolutionPath)

:- move(Current, Next), not(member(Next, PathSoFar)),

fwgc1(Next, GoalState, [Next|PathSoFar], SolutionPath).



Add this to limit search depth:

length(PathSoFar, L), L<10

What’s the initial value of PathSoFar? A list of one state, the initial state

fwgc1([w,w,w,w],[e,e,e,e], [[w,w,w,w] ], SolutionPath)

% interface, from start

fwgc(Start, GoalState, SolutionPath) :-

fwgc1(Start, GoalState,[Start], SolutionPath).

Who does what search? DFS by swi-prolog

[e,e,e,e] (initial)

[w,e,e,e] UnSafe [w,w,e,e] US [w,e,w,e] Safe [w,e,e,w] US

[e,e,w,e] S [] [e,e,e,e] S & Repeat []

What’s our responsibility?

* Specify the rules for changing from one state to another, i.e., the moves
* Keep a history of the states visited (at least in the current path)

Can we have more control in the search process?