Puzzles!

Problem #1:
logic puzzle

Assignment 11: four prolog puzzles…

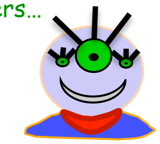
Posted after the exam

Due after Thanksgiving (but not too far after)

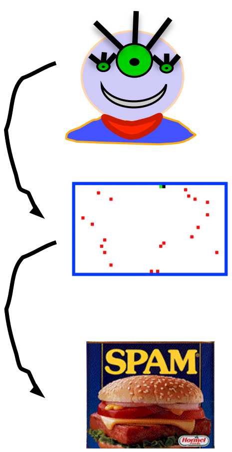
Line count:

30 lines	15 lines	15 lines	30 lines
#1	#2	#3	#4

The new generation
of logical thinkers…



Problem #2: 24



Keeping the CS 42
ecosystem balanced…

Problem #3:
Towers of Hanoi

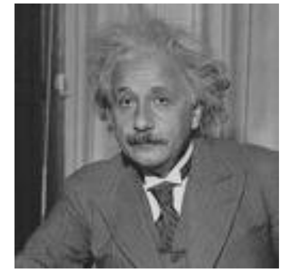
Problem #4: the island of knights and knaves…

Where we're headed...

This week:	Nov 17…22	Exam 2
HW 11:	Due Nov 29 …	Prolog Puzzles!
HW 12	Due Dec 5…	*"Theocomp"* ~ formal models of computation DFA, NFA, Regexes,
HW 13:	Due Dec 9(12)…	More on DFA, NFA, Regular expressions, also Turing Machines, reduction-style uncomputability

There are five houses:

The **nationalities** are `norwegian, brit, swede, dane, german`
The **pets** are `dog, bird, zebra, cat, horse`
The **cigars** are `pallmall, winfield, dunhill, rothmans, marlboro`
The **beverages** are `tea, coffee, milk, water, beer`
The **house colors** are `red, green, yellow, blue, white`



Einstein's Zebra puzzle:
a ***logic*** challenge

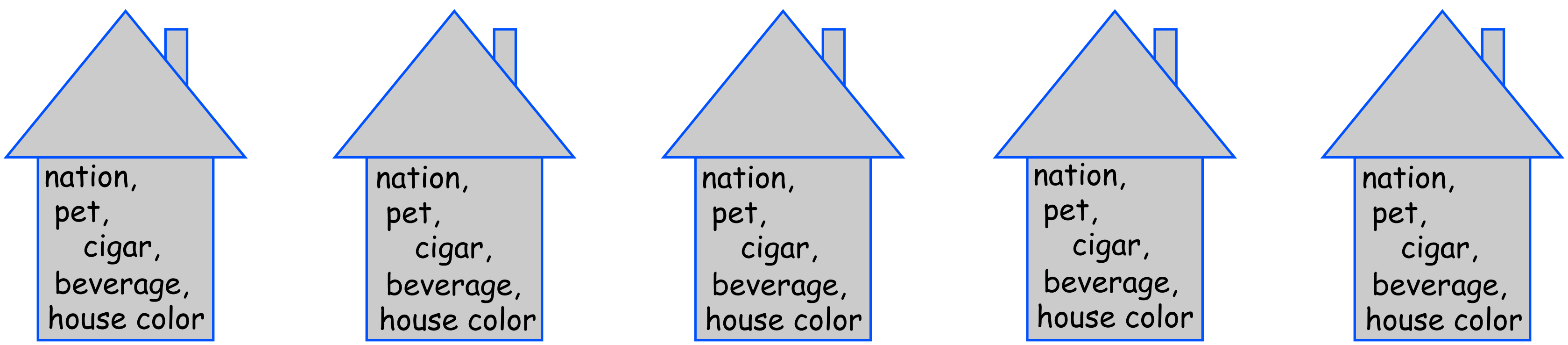
~ 1900

We know that

1) The Norwegian lives in the first house.

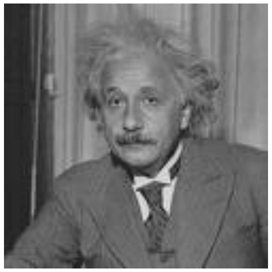
(plus fourteen more constraints)

Who owns the zebra?



There are five houses:

The **nationalities** are `norwegian, brit, swede, dane, german`
The **pets** are `dog, bird, zebra, cat, horse`
The **cigars** are `pallmall, winfield, dunhill, rothmans, marlboro`
The **beverages** are `tea, coffee, milk, water, beer`
The **house colors** are `red, green, yellow, blue, white`



Einstein's Zebra puzzle:
a ***logic*** challenge

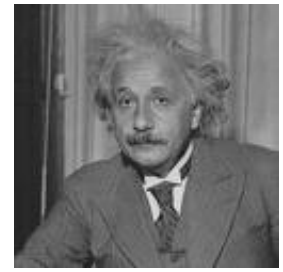
We know that

1) The Norwegian lives in the first house.
2) The person living in the center house drinks milk.
3) The Brit lives in a red house.
4) The Swede keeps dogs as pets.
5) The Dane drinks tea.
6) The Green house is next to, and on the left of the White house.
7) The owner of the Green house drinks coffee.
8) The person who smokes Pall Mall rears birds.
9) The owner of the Yellow house smokes Dunhill.
10) The man who smokes Marlboro lives next to the one who keeps cats.
11) The man who keeps horses lives next to the man who smokes Dunhill.
12) The man who smokes Winfields drinks beer.
13) The German smokes Rothmans.
14) The red house is to the right of the blue.
15) The Norwegian doesn't live by the red, white, or green houses

Who owns the zebra?

There are five houses:

The **nationalities** are `norwegian, brit, swede, dane, german`
The **pets** are `dog, bird, zebra, cat, horse`
The **cigars** are `pallmall, winfield, dunhill, rothmans, marlboro`
The **beverages** are `tea, coffee, milk, water, beer`
The **house colors** are `red, green, yellow, blue, white`



Einstein's Zebra puzzle:
a logic challenge

We know that

1) The Norwegian lives in the first house.
2) The perso
3) The Brit l
4) The Swed
5) The Dane
6) The Green
7) The owne
8) The perso
9) The owne
10) The man
11) The man
12) The man
13) The Germ
14) The red h
15) The Norwegian doesn't live by the red, white, or green houses

Zebra Puzzle

From Wikipedia, the free encyclopedia

The **Zebra Puzzle** is a well-known logic puzzle.

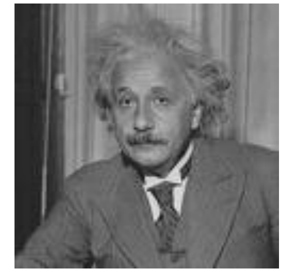
It is often called "Einstein's Puzzle" or "Einstein's Riddle" because it is said to have been invented by Albert Einstein as a boy. Some claim that Einstein said "only 2 percent of the world's population can solve it". It is also sometimes attributed to Lewis Carroll. However, there is no known evidence for Einstein's or Carroll's authorship.

There are several versions of this puzzle. The version below is quoted from the first known publication in Life International magazine on December 17, 1962. The March 25, 1963, issue contained the solution given below and the names of several hundred solvers from around the world.

Who owns the zebra?

There are five houses:

The **nationalities** are **`norwegian, brit, swede, dane, german`**
The **pets** are **`dog, bird, zebra, cat, horse`**
The **cigars** are **`pallmall, winfield, dunhill, rothmans, marlboro`**
The **beverages** are **`tea, coffee, milk, water, beer`**
The **house colors** are **`red, green, yellow, blue, white`**



Einstein's Zebra puzzle:
a logic challenge

We know that

1) The Norwegian lives in the first house.
2) The person living in the center house drinks milk.
3) The Brit lives in a red house.
4) The Swede keeps dogs as pets.
5) The Dane drinks tea.
6) The Green house is next to, and on the left of the White house.
7) The owner of the Green house drinks coffee.
8) The person who smokes Pall Mall rears birds.
9) The owner of the Yellow house smokes Dunhill.
10) The man who smokes Marlboro lives next to the one who keeps cats.
11) The man who keeps horses lives next to the man who smokes Dunhill.
12) The man who smokes Winfields drinks beer.
13) The German smokes Rothmans.
14) The red house is to the right of the blue.
15) The Norwegian doesn't live by the red, white, or green houses

(0) How do we represent the puzzle state?

(1) What relationships do we need?

(2) How do we encode the constraints?

Who owns the zebra?

Zebra-puzzle representation

A list **H** reflecting the houses' spatial order with the five items that each house has…

nation pet cigar beverage house color

```
H = [[norwegian, _, _, _, _], _, [_, _, _, milk, _], _, _],
```

House 1 House 2 House 3 Houses 4+5

Try this constraint: **7) The owner of the Green house drinks coffee.**

Hint: use `member` and `H`

Zebra-puzzle constraints

Creating some helpers:

```
% What does this say about L and R?
lr(L, R, [L, R | _ ]).
lr(L, R, [ _ | Rest]) :- lr(L, R, Rest).

% … and then X and Y?
mystery(X, Y, List) :- lr(X, Y, List).
mystery(X, Y, List) :- lr(Y, X, List).
```

`List` will be **`H`**

What is the best alternate name for the "mystery" predicate?

Zebra-puzzle constraints

Creating some helpers:

```
% left-to-right adjacency
lr(L, R, [L, R | _ ]).
lr(L, R, [ _ | Rest]) :- lr(L, R, Rest).

% adjacency (unordered)
nextTo(X, Y, List) :- lr(X, Y, List).
nextTo(X, Y, List) :- lr(Y, X, List).
```

`List` will be `H`

Try this constraint: **15) The Norwegian does not live by the red house.**

Where should this predicate go?

Zebra-puzzle output

A zero-argument predicate!

```
solve :-
  einstein( [ H1, H2, H3, H4, H5 ] ),
  write( ' first house: '), write(H1), nl,
  write( 'second house: '), write(H2), nl,
  write( ' third house: '), write(H3), nl,
  write( 'fourth house: '), write(H4), nl,
  write( ' fifth house: '), write(H5), nl.
```

and its output:

```
?- solve.

 first house: [norwegian, cat, dunhill, water, yellow]
second house: [dane, horse, marlboro, tea, blue]
 third house: [brit, bird, pallmall, milk, red]
fourth house: [german, zebra, rothmans, coffee, green]
 fifth house: [swede, dog, winfield, beer, white]

Yes
```

The Mystery of the Spamwarts Express

Five "people" are traveling home in adjacent train seats for summer break…

Seat 1 Seat 2 Seat 3 Seat 4 Seat 5

`Names = [algird, bruno, collette, dino, edwina].` Each has a different name.

`Schools = [pomona, pitzer, hmc, cmc, scripps].` Each is from a different Claremont College.

`Snacks = [jots, chocolate, donuts, pez, spam].` Each has brought a different snack.

9 hints are available

1) Dino and Bruno sat in the end seats.
2) Algird sat next to the student from HMC.
3) Collette sat next to friends with chocolate and donuts.
4) The HMC student brought spam as a snack and sat in the middle seat.
5) Chocolate was immediately to the left of pez.
6) Bruno, Dino, and Algird do not go to Scripps.
7) The Pomona student sat between the one with jots and the one with spam.
8) Dino did not sit next to the person with donuts.
9) The CMC student did not sit next to Edwina.

helper rules?

Sorting out Prolog… by sorting ***in*** Prolog!

- **`removeAll(E,L,NewL)`**
- **`uniq(L,U)`** uniquifies L

- **`perm(L,P)`** generates permutations P of L…
- **`sort(L,S)`** S is L, sorted
- **`split(L,S1,S2)`** where **`L = S1`** ∪ **`S2.`**

Perms can be tough to get right!

Prolog *is* uniq !

uniq(L, U)

removeAll(E, L, NewL)   %% assume E occurs at least once in L

removeOne(E, L, NewL)   %% assume E occurs at least once in L

```
removeOne(X,[X|R],R).
removeOne(X,[F|R],[F|S]) :- removeOne(X,R,S).
```

removeOne(3, X, [h,e,n,r,y]).

Name:______________________________ *"Quiz"*

sorted(L)

Define **`sorted(L)`**. Assume **`L`** contains only integers.

You'll want *2* base cases…

Prolog pandemonium!

---

perm(L, P)

Write **`perm(L,P)`** to generate permutations **`P`** of a list **`L`**.

Use **`removeOne(E,L,NewL)`**

---

sort(L, S)

Write **`sort(L,S)`**, which "sorts" **`L`** into **`S`**.

Use the previous two predicates!

---

split(L, S1, S2)

Write **`split(L,S1,S2)`** that splits a list **`L`** into subsets **`S1`** and **`S2`**.

so that **`L = S1 ∪ S2`**.

not concatenation

Fun and Games!

```
    S E N D
  + M O R E
  ---------
  M O N E Y
```

Goal: Assign a distinct digit (0-9) to each letter to produce a valid sum (M is not 0)

<u>A valid solution?</u>

S =
E =
N =
D =
M =
O =
R =
Y =

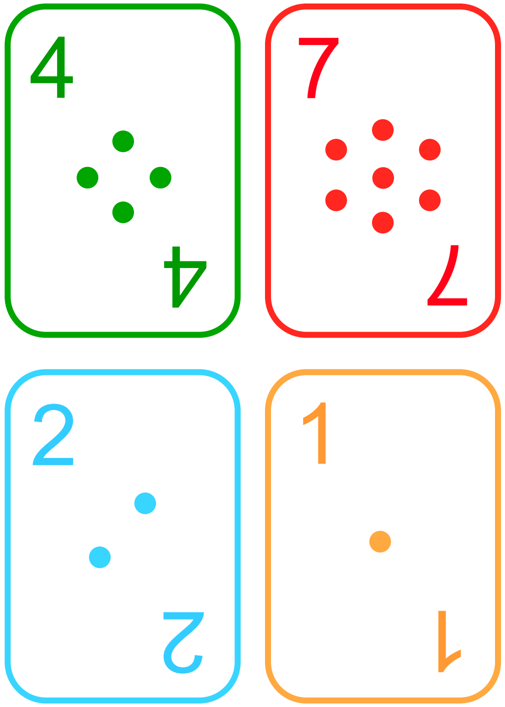
<u>A valid solution?</u>

S =
E =
N =
D =
M =
O =
R =
Y =

```
digits([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).

% element( A, X, Y ) says A is an element of X, residue Y
% i.e., removeone(X, A, Y) !

sum( A, B, C, Sum, Carry ) :-




solution( [S, E, N, D, M, O, R, Y] ) :-
  digits( Digits ),

  element( D, Digits, D1 ),                /* D E Y column */
  element( E, D1, D2 ),
  sum(D, E, 0, Y, C1 ),
  element( Y, D2, D3),


                                           /* N R E column */




                                           /* E O N column */



                                           /* S M O column */



                                           /* M != 0 */
```

money.pl by
Robert Keller

24 (Problem #2)



Can you combine these numbers into 24 with the operations **+ − * ÷** ?

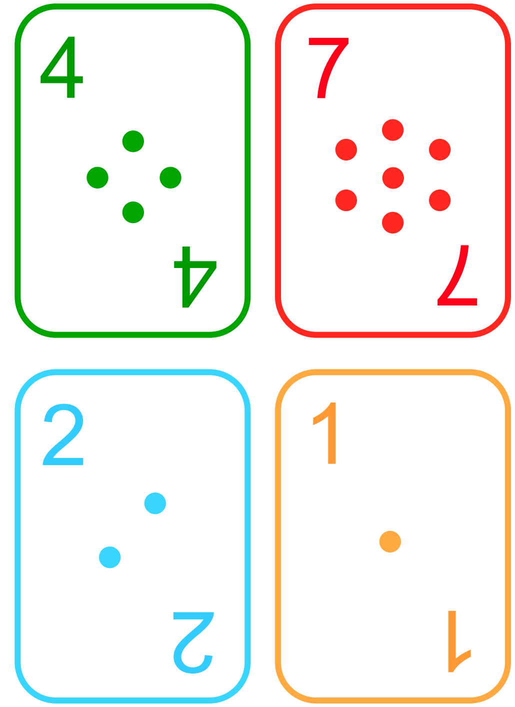
Each operation may be used any number of times.

A two-dot card from the "official" 24 game: **`24game.com`**

24 (Problem #2)

Can you combine these numbers into 24 with the operations **+ − * ÷** ?

Each operation may be used any number of times.




```
solve( ['+','-','*','/'], [1,2,4,7], 24, Tree ).
```

operators available: **(use any number of times)**

values: **(use each exactly once)**

final answer

arithmetic tree

```
Tree = [*, [+, 1, [-,7,2]], 4]
```

Searching to the end…

be sure there *is* an end!

```
setof( Tree, solve( ['+','-','*','/'], [1,2,4,7], 24, Tree ) , S).
```

variable whose values you want to collect into a set

predicate that you would like to satisfy…

the set

```
setof( Tree, solve( ['+','-','*','/'], [5,1,155,11], 24, Tree ) , S).
```

a much smaller example…

notice the splitting!

```
[2,8,22,424] ~ 42
[4,4,4,4] ~ 42
```

```
setof( N, T^solve( ['+','-','*','/'], [4,4,4,4], N, T ) , S).
```

a surprisingly well-studied problem!

T^ ~ there exists a T

For four 4's ...

Martin Gardner MARTIN GAMES

$0 = 4 + 4 - 4 - 4$

$1 = 44 / 44$

$2 = 4^{4/(4+4)}$

...

$8 = \mathrm{sqrt}(\ .\overline{4}\ ) * (\ 4 + 4 + 4\ )$

...

19 = ???

$20 = 4! + 4 - 4 - 4$

Allowed Operations

\+ − * /

sqrt . $\overline{}$!

concat power

Ask around…

24 (Hint) (Problem #2)

```
solve( base case ).
solve( Ops, Values, Sol, Tree ) :-
   Tree = [ Op, Left, Right ],
   ... .
```

Why won't this work?

A. What do you mean? It will work!
B. You can't unify a variable with a list
C. You need to use the == operator instead of =
D. All of Tree, Op, Left and Right may be unbound, which will cause a problem
E. You need to instead write [Op, Left, Right] = Tree

solve

- What is the base case?

- What is the recursive case?

```
solve( Ops, Values, Sol, Tree ) :-
```

Set up: **`Ops`** and **`Values`** will have values; **`Sol`** and **`Tree`** may not…

```
Tree = [ Op, Left, Right ].
```

Use **`split`** !

Other constraints, perhaps…

```
solve( [‘+’,‘-’,‘*’,‘/’], [1,2,4,7], 24, Tree ).
```

operators available:
(use any number of times)

values:
(use each exactly once)

final answer

arithmetic tree

```
Tree = [*,[+,1,[-,7,2]],4]
```

Each operation may be used any number of times.

eval

Tree-evaluation code provided:

Notice the base case!

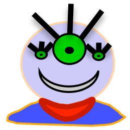
```
eval(R, R) :- number(R).
eval(['+', A, B], R) :- eval(A, AR), eval(B, BR), R is AR + BR.
eval(['*', A, B], R) :- eval(A, AR), eval(B, BR), R is AR * BR.
eval(['-', A, B], R) :- eval(A, AR), eval(B, BR), R is AR - BR.
eval(['/', A, B], R) :- eval(A, AR), eval(B, BR), BR\==0, R is AR // BR.
```

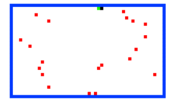
the tree

It also checks `nonvar(A)` and `nonvar(B)`

resulting value

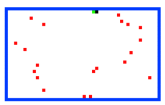
The alien, the spampede, and the spam!

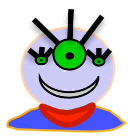




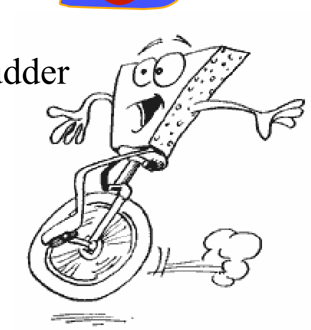
Goal: Get back to the dorms with the spam, the spampede, and the three-eyed alien.

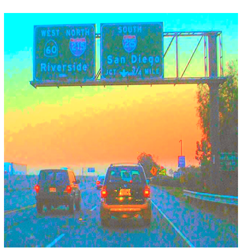
Constraints: Can't carry more than one thing and can't leave two things alone if one would eat the other…





Mudder





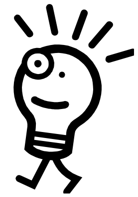
West

East

1. Mudder takes spampede east
2. Mudder goes west
3. Mudder takes alien east
4. Mudder takes spampede west
5. Mudder takes spam east
6. Mudder goes west
7. Mudder takes spampede east

How to formulate in Prolog?

The BIG Idea



initial configuration

`[ [mudder, alien, spampede, spam], [] ]`

`Mudder_takes_spampede_east`

`[ [alien, spam], [mudder, spampede] ] ]`

. . .

`[ [], [mudder, alien, spampede, spam] ]`

final configuration

```
solve( [[mudder,alien,spampede,spam],[]],
        [mudder_takes_spampede_east,
   mudder_goes_west, ...] ).
yes.
```

The alien, the spampede, and the spam…

Configuration ~ a list of two ***sorted*** lists

$$\underbrace{\texttt{[[mudder, alien, spampede, spam], [ ]]}}_{\text{one possible configuration}}$$

$$\texttt{[}\underbrace{\texttt{[mudder, alien, spampede, spam]}}_{\text{West}}\texttt{,}\ \underbrace{\texttt{[ ]}}_{\text{East}}\texttt{]}$$

What constraints do we have?

The alien, the spampede, and the spam…

Configuration ~ a list of two ***sorted*** lists

$$\underbrace{[\ \underbrace{[\ \texttt{mudder, alien, spampede, spam}\ ]}_{\text{West}},\ \underbrace{[\ \ ]}_{\text{East}}\ ]}_{\mathbf{C}}$$

Constraints ~ describing the puzzle

```
%% This checks each List (side) of a configuration. How?!
ok(L) :- sorted(L), (member(mudder, L) ; \+predatorprey(L)).

predatorprey(L) :- member(alien, L), member(spampede, L).
predatorprey(L) :- member(spampede, L), member(spam, L).

%% The final configuration is defined here.
final([ [], [mudder, alien, spampede, spam] ]).
```

`L` can be the West ***or*** the East side of a configuration

The alien, the spampede, and the spam…

Constraints ~ describe and ***select*** the moves:

describes a valid move

from this "before" configuration

via this action

to this "after" configuration

```
valid([Wb, Eb], mudder_goes_east, [Wa, Ea]) :-
```

How do we generate **`Wa`** and **`Ea`** ?
What constraints do we need to check?

The alien, the spampede, and the spam…

Constraints ~ describe and ***select*** the moves:

describes a valid move

from this "before" configuration

via this action

to this "after" configuration

```
valid([Wb, Eb], mudder_goes_east, [Wa, Ea]) :-
    perm([mudder|Eb],Ea),
    perm(Wb,[mudder|Wa]),
    ok(Wa),
    ok(Ea).
```

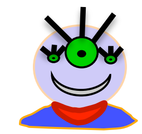
uses `perm` to generate the resulting configuration

makes sure that they're sorted and safe from predators!

How Muddery different possible moves are there?

The alien, the spampede, and the spam…

Wasn't there 42-dimensional configuration-space search in CS 5?



`solve` ~ general-purpose puzzle-solving!

configuration-space search!

need to specify a starting configuration

Ask Prolog to generate a list of moves that results in the final configuration…

```
solve( StartConfig, ListOfMoves )
```

base case?

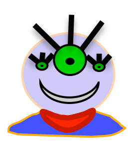
The alien, the spampede, and the spam…

solve ~ general-purpose puzzle-solving!

Prolog's configuration-space search:

```
%% what it means for a list of moves to solve the puzzle...
%% C is a configuration.
solve(C, []) :- final(C).    %% Done! (base case)

solve(C, [Move | RoM]) :-    %% Move ~ next move to make
    [W, E] = C,              %% C is the [ West, East ] pair
    ok(W), ok(E),            %% is everything in order?
    valid(C, Move, NewC),    %% selects a valid Move !
    solve(NewC, RoM).        %% solve the rest of the puzzle...
```

Doesn't this solve
ALL problems !?!



How does it do?

The alien, the spampede, and the spam…

solve ~

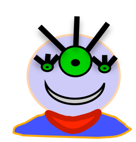
Prolog's config

```
%% what it                         puzzle...
%% C is a c
solve(C, []

solve(C, [M                         make
     [W, E]                        st ] pair
     ok(W),                        der?
     valid(C                       !
     solve(N                       ne puzzle...
```

Aargh!

It doesn't work!!

Doesn't this solve
ALL problems !?!



How does it do?

`dynamic` predicates

```
:- dynamic marked/1.
```

declares **`marked`** to be a ***dynamic*** rule ~ iy can be changed an the fly…

add

```
assert( marked(42) ).
```

places **`marked(42)`** into the program's current database of facts.

```
marked( X )
```

will now produce at least one binding: **`X = 42`**

remove

```
retract( marked(42) ).
```

removes **`marked(42)`** from the program's current database of facts.

```
retractall( marked(_) ).
```

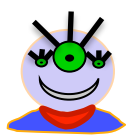
removes all of **`marked`**'s rules from the program's current facts.

The alien, the spampede, and the spam…

Prolog's configuration-space search, with ***dynamic*** marking of configurations along the way…

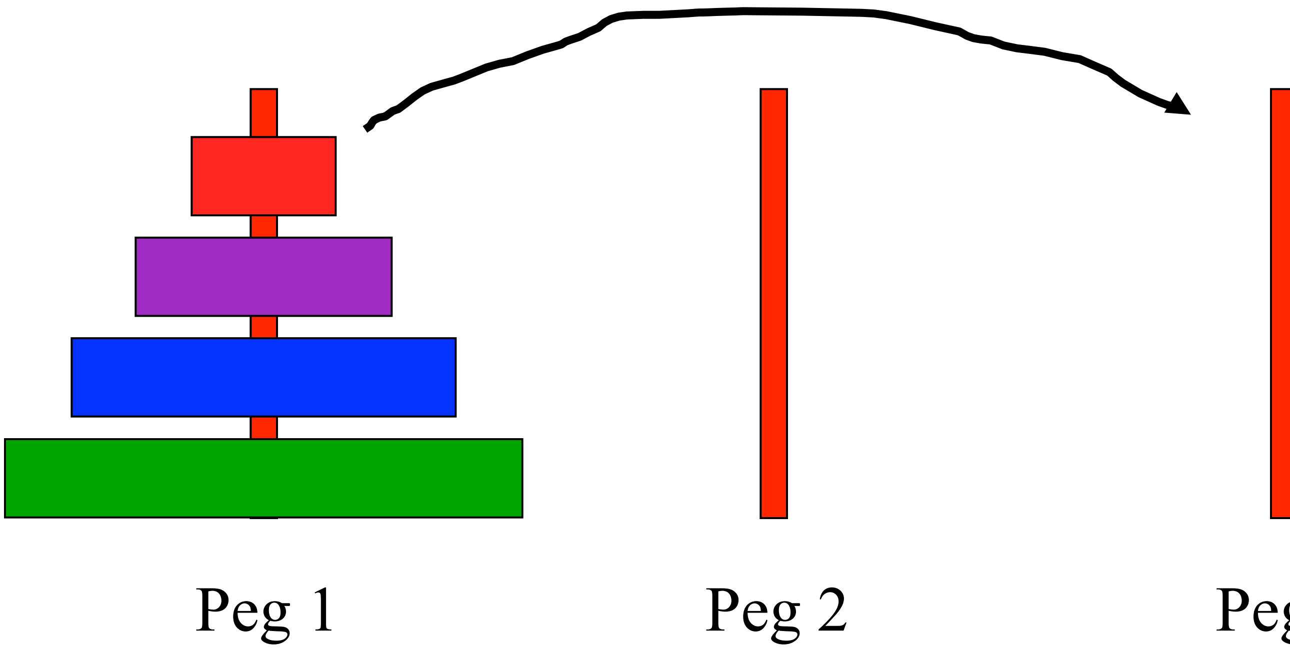
```
:- dynamic marked/1.

solve(C, []) :- final(C),                      %% Done!
                retractall(marked(_)).          %% clean up marks
solve(C, [Move | RoM]) :-   %% Move ~ next move to make
     [W, E] = C,            %% C is the [ West, East ] pair
     ok(W), ok(E),          %% is everything in order?
     \+marked(C),           %% "if" we haven't visited this C
     assert(marked(C)),     %% we mark it
     valid(C, Move, NewC),  %% select a valid Move !
     solve(NewC, RoM).      %% solve the rest of the puzzle...
```

Let's try it (twice)!



Towers of Hanoi

Goal: Get all of the disks from Peg 1 to Peg 3. **Constraints:** You may not place a smaller disk onto a larger one.




```
solve( [[1,2,3,4],[],[]], M ).
```

This could get Hanoi'ing!

Use spam.pl as a starting point…