Prolog
Programming in Logic

Lecture #4

Ian Lewis, Andrew Rice

Q. What format are the comments?

There's a Prolog convention for comments

```
% take(+L,-H,-T)
% take succeeds if list T is list L with H removed.

take([H|T],H,T).                    % if you take H from a list containing [H|T] you are left with T

take([H|T],E,[H|R]) :- take(T,E,R).  % you can take E from some list with head H and tail T leaving a list with head H and tail R if you can take E from T leaving R.
```

This are called 'modes'

++	The argument is ground (no variables anywhere).
+	Instantiated but not necessarily ground.
-	The argument is an 'output' argument.
?	Anything.

I've missed some of the classes out in the interests of time. See the full list here: 'Type, mode and determinism declaration headers'
http://www.swi-prolog.org/pldoc/man?section=modes

Q. Why use an accumulator? LCO?

Q. Why use an accumulator? LCO?

A. Space Optimisation.

The 'accumulator version' of a relation may permit LCO (len).

Or, working with lists, the accumulator may allow building a list by adding a 'head' rather than appending.

Q. Is Prolog logic 'incomplete' because it lacks functions?

Q. Is Prolog logic 'incomplete' because it lacks functions?

A. Prolog doesn't lack functions. These are deterministic relations implemented by flattening:

```
fun fact(1) = 1;
    fact(N) = N * fact(N-1).
```

```
fact(1,1).
fact(N,FN) :- N > 1, M is N - 1, fact(M,FM), FN is N * FM.
```

Q: Regarding cut, If we have rule
answer :- generate, !, test.
would this evaluate to false (& thus miss solutions) if test is not true for the first generated solution?

A: Yes. But we're not talking about cut today...

Today's discussion

Videos:

- Generate and Test (Dutch Flag, Sudoku)
- Symbolic (Eval)

Course Outline

1. Introduction, terms, facts, unification
2. Unification. Rules. Lists.
3. Arithmetic, Accumulators, Backtracking
4. Generate and Test (Dutch Flag, Sudoku), eval.
5. Extra-logical predicates (cut, negation, assert)
6. Graph Search
7. Difference Lists
8. Wrap Up (..Sudoku..)

Today's discussion

Videos:

- Generate and Test (Dutch Flag, Sudoku)
- Symbolic (Eval)

Dutch Flag

```
?- dutch_national_flag([blue,red,white,blue],L)

L = [red,white,blue,blue]
```

```
dutch_national_flag(In,Out) :-
                    perm(In,Out),        GENERATE
                    checkColours(Out).   TEST

checkColours(List) :- checkRed(List).

checkRed([red|T]) :- checkRed(T).
checkRed([white|T]) :- checkWhite(T).

checkWhite([white|T]) :- checkWhite(T).
checkWhite([blue|T]) :- checkBlue(T).

checkBlue([blue|T]) :- checkBlue(T).
checkBlue([]).
```

brevity: Flag

LIST CONTAINS AT LEAST ONE OF EACH COLOUR

```
?-
flag([b,r,w,b],L)

L = [r,w,b,b]

...
```

Flag

LIST CONTAINS AT LEAST ONE OF EACH COLOUR

```
?-
flag([b,r,w,b],L)

L = [r,w,b,b]

...
```

```
% checkB(L) succeeds if all list L are blue or L empty.
checkB([b|T]) :- checkB(T).
checkB([]).
```

Flag

LIST CONTAINS AT LEAST ONE OF EACH COLOUR

```
?-
flag([b,r,w,b],L)

L = [r,w,b,b]

...
```

```
% checkB(L) succeeds if all list L are blue or L empty.
checkB([b|T]) :- checkB(T).
checkB([]).

% checkWB(L) succeeds if L is white followed by blue.
checkWB([w|T]) :- checkWB(T).
checkWB([b|T]) :- checkB(T).
```

Flag

LIST CONTAINS AT LEAST ONE OF EACH COLOUR

```
?-
flag([b,r,w,b],L)

L = [r,w,b,b]

...
```

```
% checkB(L) succeeds if all list L are blue or L empty.
checkB([b|T]) :- checkB(T).
checkB([]).

% checkWB(L) succeeds if L is white followed by blue.
checkWB([w|T]) :- checkWB(T).
checkWB([b|T]) :- checkB(T).

% checkRWB succeeds if L is reds.. whites.. blues.
checkRWB([r|T]) :- checkRWB(T).
checkRWB([w|T]) :- checkWB(T).
```

Flag

LIST CONTAINS AT LEAST ONE OF EACH COLOUR

```
?-
flag([b,r,w,b],L)

L = [r,w,b,b]

...
```

```
% checkB(L) succeeds if all list L are blue or L empty.
checkB([b|T]) :- checkB(T).
checkB([]).

% checkWB(L) succeeds if L is white followed by blue.
checkWB([w|T]) :- checkWB(T).
checkWB([b|T]) :- checkB(T).

% checkRWB succeeds if L is reds.. whites.. blues.
checkRWB([r|T]) :- checkRWB(T).
checkRWB([w|T]) :- checkWB(T).

% flag(L1,L2) succeeds if L2 is red-white-blue sorted L1
flag(L1,L2) :-
   perm(L1,L2),          GENERATE
   checkRWB(L2).         TEST
```

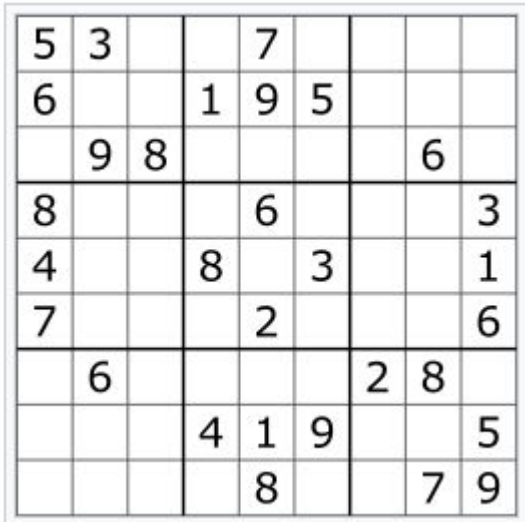
Dutch Flag

```
?- dutch_national_flag([blue,red,white,blue],L)

L = [red,white,blue,blue]
```

```
dutch_national_flag(In,Out) :-
   perm(In,Out),          GENERATE
   checkColours(Out).     TEST

checkColours(List) :- checkRed(List).

checkRed([red|T]) :- checkRed(T).
checkRed([white|T]) :- checkWhite(T).

checkWhite([white|T]) :- checkWhite(T).
checkWhite([blue|T]) :- checkBlue(T).

checkBlue([blue|T]) :- checkBlue(T).
checkBlue([]).
```

SUDOKU 8x8

https://en.wikipedia.org/wiki/Sudoku



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
puzzle1 :- solve([5,3,_,_,7,_,_,_,_],
                 [6,_,_,1,9,5,_,_,_],
                 [_,9,8,_,_,_,_,6,_],
                 [8,_,_,_,6,_,_,_,3],
                 [4,_,_,8,_,3,_,_,1],
                 [7,_,_,_,2,_,_,_,6],
                 [_,6,_,_,_,_,2,8,_],
                 [_,_,_,4,1,9,_,_,5],
                 [_,_,_,_,_,8,_,_,7,9]
                ).
```

perm

```
% take(L1,X,L2) succeeds if output list L2 is input list L1 minus element X
take([H|T],H,T).
take([H|T],R,[H|S]) :- take(T,R,S).

% perm(L1,L2) succeeds if output list L2 is a permutation of input list L1
perm([],[]).
perm(A,[R|S]) :- take(A,R,P), perm(P,S).

% gen_digits(L) succeeds if L is a permutation of [1,2,3,4,5,6,7,8,9]
gen_digits(L) :- perm([1,2,3,4,5,6,7,8,9],L).
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,_,_,7,_,_,_,_],
           B = [6,_,_,1,9,5,_,_,_],
           C = [_,9,8,_,_,_,_,6,_],
           D = [8,_,_,_,6,_,_,_,3],
           E = [4,_,_,8,_,3,_,_,1],
           F = [7,_,_,_,2,_,_,_,6],
           G = [_,6,_,_,_,_,2,8,_],
           H = [_,_,_,4,1,9,_,_,5],
           I = [_,_,_,_,8,_,_,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test fixed numbers,
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

GENERATE: gen_digits(A), with and without 'seed'

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,_,_,1,9,5,_,_,_],
           C = [_,9,8,_,_,_,_,6,_],
           D = [8,_,_,_,6,_,_,_,3],
           E = [4,_,_,8,_,3,_,_,1],
           F = [7,_,_,_,2,_,_,_,6],
           G = [_,6,_,_,_,_,2,8,_],
           H = [_,_,_,4,1,9,_,_,5],
           I = [_,_,_,_,8,_,_,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test fixed numbers,
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test fixed numbers,
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test fixed numbers,
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

SUDOKU 9x9

```
puzzle1 :- A = [5,3,1,2,7,4,6,8,9],
           B = [6,2,3,1,9,5,4,7,8],
           C = [1,9,8,2,3,4,5,6,7],
           D = [8,1,2,4,6,5,7,9,3],
           E = [4,2,5,8,6,3,7,9,1],
           F = [7,1,3,4,2,5,8,9,6],
           G = [1,6,3,4,5,7,2,8,9],
           H = [2,3,6,4,1,9,7,8,5],
           I = [1,2,3,4,8,5,6,7,9],
```

```
gen_digits(A),
gen_digits(B),
gen_digits(C),
gen_digits(D),
gen_digits(E),
gen_digits(F),
gen_digits(G),
gen_digits(H),
gen_digits(I),
Test the ‘boxes’,
Test the ‘columns’,
Print Solution.
```

```
puzzle1 :-A = [5,3,_,_,7,_,_,_,_],
          B = [6,_,_,1,9,5,_,_,_],
          C = [_,9,8,_,_,_,_,6,_],
          D = [8,_,_,_,6,_,_,_,3],
          E = [4,_,_,8,_,3,_,_,1],
          F = [7,_,_,_,2,_,_,_,6],
          G = [_,6,_,_,_,_,2,8,_],
          H = [_,_,_,4,1,9,_,_,5],
          I = [_,_,_,_,8,_,_,7,9],
          gen_digits(A), A = [5,3,1,2,7,4,6,8,9]
          gen_digits(B),
          gen_digits(C),
          ...
          Test the ‘boxes’
          Test the ‘columns’
```

```
[5,3,1,2,7,4,6,8,9]
[6,_,_,1,9,5,_,_,_]
[_,9,8,_,_,_,_,6,_]
[8,_,_,_,6,_,_,_,3]
[4,_,_,8,_,3,_,_,1]
[7,_,_,_,2,_,_,_,6]
[_,6,_,_,_,_,2,8,_]
[_,_,_,4,1,9,_,_,5]
[_,_,_,_,8,_,_,7,9]
```

```
puzzle1 :-A = [5,3,_,_,7,_,_,_,_],
          B = [6,_,_,1,9,5,_,_,_],
          C = [_,9,8,_,_,_,_,6,_],
          D = [8,_,_,_,6,_,_,_,3],
          E = [4,_,_,8,_,3,_,_,1],
          F = [7,_,_,_,2,_,_,_,6],
          G = [_,6,_,_,_,_,2,8,_],
          H = [_,_,_,4,1,9,_,_,5],
          I = [_,_,_,_,8,_,_,7,9],
          gen_digits(A), A = [5,3,1,2,7,4,6,8,9]
          Test the 'columns'
          gen_digits(B),
          gen_digits(C),
          ...
          Test the 'boxes'
```

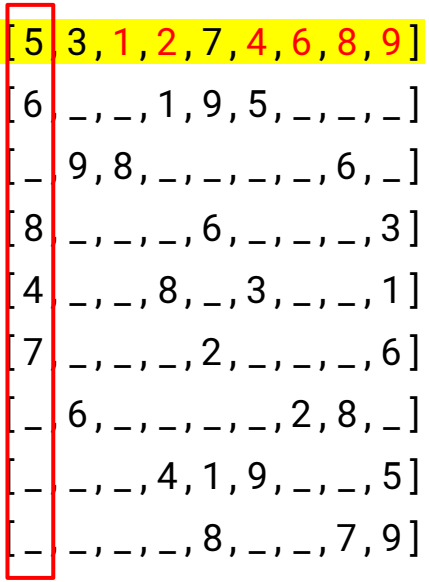
```
[5,3,1,2,7,4,6,8,9]
[6,_,_,1,9,5,_,_,_]
[_,9,8,_,_,_,_,6,_]
[8,_,_,_,6,_,_,_,3]
[4,_,_,8,_,3,_,_,1]
[7,_,_,_,2,_,_,_,6]
[_,6,_,_,_,_,2,8,_]
[_,_,_,4,1,9,_,_,5]
[_,_,_,_,8,_,_,7,9]
```

```
test_digits([5,6,_,8,4,7,_,_,_])
```

Input argument of VARS and DIGITS

Succeed if DIGITS unique in [1..9]

```
heads([[a,b,c],
       [d,e,f],
       [g,h,i]], X)
X = [a,d,g]
```




```
test_digits([5,6,_,8,4,7,_,_,_])

Input argument of VARS and DIGITS

Succeed if DIGITS unique in [1..9]

heads([[a,b,c],
       [d,e,f],
       [g,h,i]], X, Tails)
X = [a,d,g]
Tails = [[b,c],
         [e,f],
         [h,i]]
```

test_digits(L)

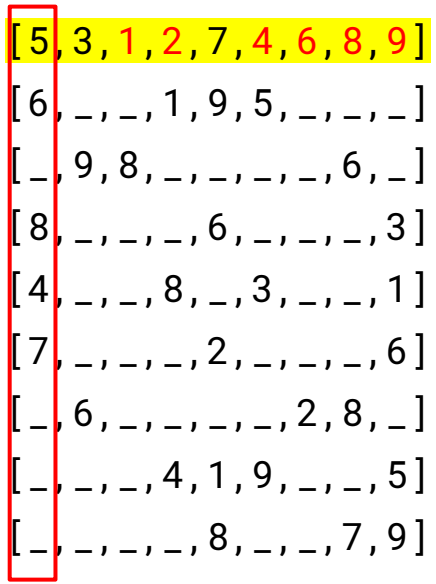
```
% test_digits(L) succeeds if:
%    input list L contains digits and/or variables
%    the digits in L are unique from [1..9]
test_digits(L) :- test_set(L,[1,2,3,4,5,6,7,8,9]).

% test_set(L1,L2) succeeds if:
%    input list L1 contains digits and/or variables
%    the digits in L1 are unique in input list L2
test_set([],_).
test_set([H|T], Digits) :- var(H),
                           test_set(T,Digits).
test_set([H|T], Digits) :- ground(H),
                           take(Digits,H,RemainingDigits),
                           test_set(T,RemainingDigits).
```

heads(LL, LHeads, LTails)

```
% heads(LL, LHeads, LTails) succeeds if:
%   LL is an input list of lists
%   LHeads is a list of the heads of each list in LL
%   LTails is LL with the head of each list removed.
heads([],[],[]).
heads([[H|T1]|T],[H|Hs], [T1|Tails]) :- heads(T,Hs,Tails).

:- heads([[1,2,3],
          [4,5,6],
          [7,8,9]],Heads, Tails).
Heads = [1,4,7]
Tails = [[2,3],[5,6],[8,9]]
```

test_cols(Rows)

```
% test_cols(Rows) succeeds if:
%    input Rows is a list of lists of digits and variables
%    the 'columns' of digits and variables contain unique digits from [1..9]
test_cols([[]|_]).
test_cols(Rows) :- heads(Rows,Heads,Tails),
                   test_digits(Heads),
                   test_cols(Tails).
```



```
[5,3,1,2,7,4,6,9,8]
[6,2,3,1,9,5,4,8,7]
[_,9,8,_,_,_,_,6,_]
[8,_,_,_,6,_,_,_,3]
[4,_,_,8,_,3,_,_,1]
[7,_,_,_,2,_,_,_,6]
[_,6,_,_,_,_,2,8,_]
[_,_,_,4,1,9,_,_,5]
[_,_,_,_,8,_,_,7,9]
```

```
puzzle1 :-A = [5,3,_,_,7,_,_,_,_],
          B = [6,_,_,1,9,5,_,_,_],
          C = [_,9,8,_,_,_,_,6,_],
          D = [8,_,_,_,6,_,_,_,3],
          E = [4,_,_,8,_,3,_,_,1],
          F = [7,_,_,_,2,_,_,_,6],
          G = [_,6,_,_,_,_,2,8,_],
          H = [_,_,_,4,1,9,_,_,5],
          I = [_,_,_,_,8,_,_,7,9],
          gen_digits(A), A = [5,3,1,2,7,4,6,9,8]
          Test the ‘columns’
          gen_digits(B), B = [6,2,3,1,9,5,4,8,7]
          Test the ‘columns’
          gen_digits(C),
          ...
          Test the ‘boxes’
```

```
[5,3,1,2,7,4,6,9,8]
[6,2,3,1,9,5,4,8,7]
[1,9,8,2,3,4,5,6,7]
[8,_,_,_,6,_,_,_,3]
[4,_,_,8,_,3,_,_,1]
[7,_,_,_,2,_,_,_,6]
[_,6,_,_,_,_,2,8,_]
[_,_,_,4,1,9,_,_,5]
[_,_,_,_,8,_,_,7,9]
```

```
puzzle1 :-A = [5,3,_,_,7,_,_,_,_],
          B = [6,_,_,1,9,5,_,_,_],
          C = [_,9,8,_,_,_,_,6,_],
          D = [8,_,_,_,6,_,_,_,3],
          E = [4,_,_,8,_,3,_,_,1],
          F = [7,_,_,_,2,_,_,_,6],
          G = [_,6,_,_,_,_,2,8,_],
          H = [_,_,_,4,1,9,_,_,5],
          I = [_,_,_,_,8,_,_,7,9],
          gen_digits(A), A = [5,3,1,2,7,4,6,9,8]
          Test the ‘columns’
          gen_digits(B), B = [6,2,3,1,9,5,4,8,7]
          Test the ‘columns’
          gen_digits(C),
          ...
          Test the ‘boxes’
```

```
[5,3,1,2,7,4,6,9,8]
[6,2,3,1,9,5,4,8,7]
[1,9,8,2,3,4,5,6,7]
[8,_,_,_,6,_,_,_,3]
[4,_,_,8,_,3,_,_,1]
[7,_,_,_,2,_,_,_,6]
[_,6,_,_,_,_,2,8,_]
[_,_,_,4,1,9,_,_,5]
[_,_,_,_,8,_,_,7,9]
```

```
puzzle1 :-A = [5,3,_,_,7,_,_,_,_],
          B = [6,_,_,1,9,5,_,_,_],
          C = [_,9,8,_,_,_,_,6,_],
          D = [8,_,_,_,6,_,_,_,3],
          E = [4,_,_,8,_,3,_,_,1],
          F = [7,_,_,_,2,_,_,_,6],
          G = [_,6,_,_,_,_,2,8,_],
          H = [_,_,_,4,1,9,_,_,5],
          I = [_,_,_,_,8,_,_,7,9],
          gen_digits(A), A = [5,3,1,2,7,4,6,9,8]
          Test the 'columns'
          gen_digits(B), B = [6,2,3,1,9,5,4,8,7]
          Test the 'columns'
          gen_digits(C),
          Test the 'boxes'
```

test_boxes(RowA,RowB,RowC)

```
% test_boxes(RowA,RowB,RowC) succeeds if:
%    RowA, RowB, RowC are input lists of 9 unique digits [1..9]
%    each aligned 3x3 'box' contains 9 unique digits 1..9
test_boxes([A1,A2,A3,A4,A5,A6,A7,A8,A9],
           [B1,B2,B3,B4,B5,B6,B7,B8,B9],
           [C1,C2,C3,C4,C5,C6,C7,C8,C9]) :- test_digits([A1,A2,A3,B1,B2,B3,C1,C2,C3]),
                                            test_digits([A4,A5,A6,B4,B5,B6,C4,C5,C6]),
                                            test_digits([A7,A8,A9,B7,B8,B9,C7,C8,C9]).
```

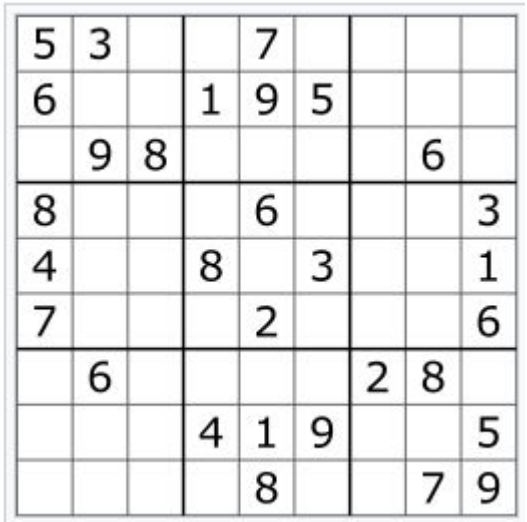
Note by time of test, boxes are fully instantiated, i.e. ground.

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

```
solve(A,B,C,D,E,F,G,H,I) :- gen_digits(A),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(B),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(C),
                            test_boxes(A,B,C),
                            gen_digits(D),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(E),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(F),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            test_boxes(D,E,F),
                            gen_digits(G),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(H),
                            test_cols([A,B,C,D,E,F,G,H,I]),
                            gen_digits(I),
                            test_boxes(G,H,I),
                            test_cols([A,B,C,D,E,F,G,H,I]).
```

SUDOKU 8x8

https://en.wikipedia.org/wiki/Sudoku



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
puzzle1 :- solve([5,3,_,_,7,_,_,_,_],
                 [6,_,_,1,9,5,_,_,_],
                 [_,9,8,_,_,_,_,6,_],
                 [8,_,_,_,6,_,_,_,3],
                 [4,_,_,8,_,3,_,_,1],
                 [7,_,_,_,2,_,_,_,6],
                 [_,6,_,_,_,_,2,8,_],
                 [_,_,_,4,1,9,_,_,5],
                 [_,_,_,_,_,8,_,_,7,9]
                ).
```

SUDOKU 8x8

https://en.wikipedia.org/wiki/Sudoku

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Symbolic Evaluation: eval, reduce, flatten

```
eval(add(A,B),C) :- eval(A,A1), eval(B,B1), C is A1+B1.
reduce(+(A,B),C) :- reduce(A,A1), reduce(B,B1), add(A1,B1,C).
```

Function support:

```
    fun fact(1) = 1;
        fact(N) = N * fact(N-1).

    fun(;(=(fact(1),1), =(fact(N),*(N,fact(-(N,1)))))).
```

Flattening:

```
    foo(X,Y) :- moo(fact(X+3),Y).
```

becomes:

```
    foo(X,Y) :- add(X,3,A1), fact(A1,A2), moo(A2,Y).
```

NOTE THAT FUNCTIONAL REDUCTION IS DETERMINISTIC

Next time

Videos

- Cut
- Negation
- Databases

Course Outline

1. Introduction, terms, facts, unification
2. Unification. Rules. Lists.
3. Arithmetic, Accumulators, Backtracking
4. Generate and Test (Dutch Flag, Sudoku), eval.
5. Extra-logical predicates (cut, negation, assert)
6. Graph Search
7. Difference Lists
8. Wrap Up (..Sudoku..)