CS 131 Discussion 4

Winter 2015

Announcements

- **Homework 3**

  Due Tues, Feb 03 at 23:55

- **Midterm**

  Thurs, Feb 05, during lecture

Prolog

http://www.gprolog.org/#download

Simplest Example

test.pl

```
happy(john).
happy(betty).
likes(mary, X) :- happy(X).
```

Queries

```
| ?- consult('test.pl').

| ?- likes(mary, betty).

| ?- likes(mary, susan).

| ?- likes(john, betty).
```

Length of list

```
len([], 0).
len([_|T], N) :-
    len(T, Nt), N is Nt+1.

| ?- len([1,2,3,4,5], N).
N = 5
yes
| ?- len([susan,
          [1,2,john],
          [betty, john]], N).
N = 3
yes
```

But:

```
len([], 0).
len([_|T], N) :-
        len(T, Nt), Nt is N-1.

| ?- len([1,2,3,4], N).
uncaught exception: error
(instantiation_error,(is)/2)
```

This happens because when N-1 is evaluated, the value of N is still unknown.

All elements of list are length N

Example:

```
| ?- allElemSizeN( [[1,2], [1,2],
[1,2]], 2 ).
yes
| ?- allElemSizeN( [[1,2], [1,2],
[1,2,3]], 2 ).
no
| ?- allElemSizeN( M, 2 ).
M = [] ? ;
M = [[_,_]] ? ;
M = [[_,_],[_,_]] ? ;
M = [[_,_],[_,_],[_,_]] ? ;
M = [[_,_],[_,_],[_,_],[_,_]] ?
...
```

Prolog:

```
allElemSizeN([],_).
allElemSizeN([H|T], N) :-
    length(H, N), allElemSizeN(T, N).
```

The list is a NxN matrix

Example:

```
| ?- nxn(2, [[1,2], [3,4]]).
yes
| ?- nxn(N, [[1,2,3],[1,2,3],[1,2,3]]).
N = 3
yes
| ?- nxn(2,M).
M = [[_,_],[_,_]]
yes
```

Prolog

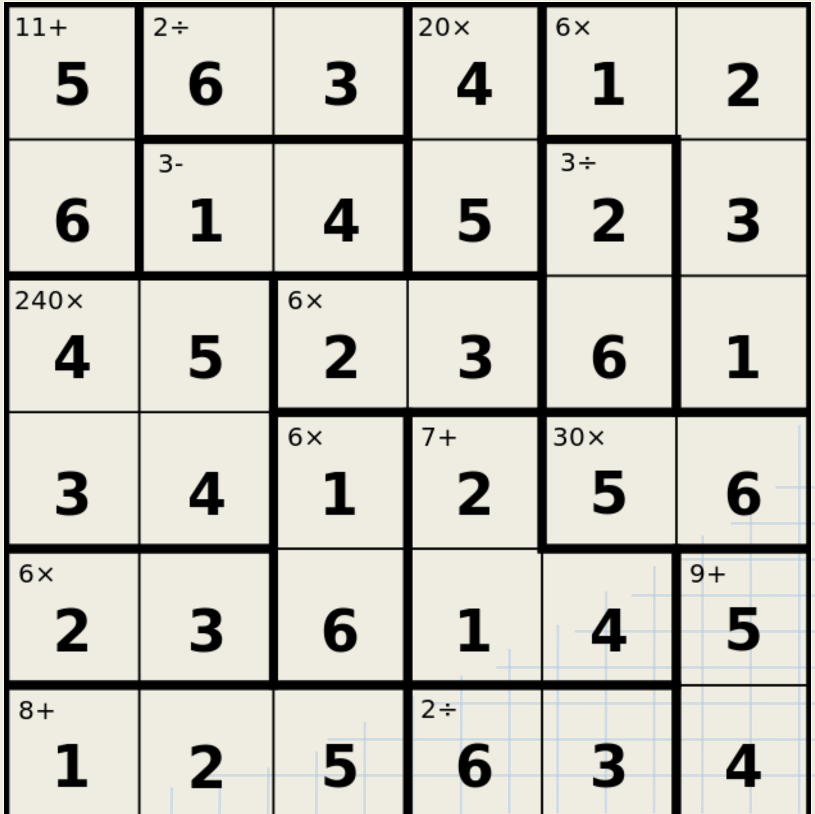
```
nxn(N, S) :-
    length(S,N), allElemSizeN(S,N).
```

Homework 3

KenKen

Logic puzzle:

- NxN matrix (eg. 6x6)
- Each row is a permutation of [1,2,3,...,N]
- Each column is a permutation of [1,2,3,...N]
- Certain cells (1 or more) must add/multiply to a certain value
- Certain pairs of cells must divide or subtract to a certain value



11+ 5	2÷ 6	3	20× 4	6× 1	2
6	3- 1	4	5	3÷ 2	3
240× 4	5	6× 2	3	6	1
3	4	6× 1	7+ 2	30× 5	6
6× 2	3	6	1	4	9+ 5
8+ 1	2	5	2÷ 6	3	4

The Problem

Given the size of the grid, and a set of constraints, solve the puzzle.

```
kenken_testcase(
  6,
  [
   +(11, [1-1, 2-1]), /(2, 1-2, 1-3), *(20, [1-4, 2-4]),  *(6, [1-5, 1-6, 2-6, 3-6]),
   -(3, 2-2, 2-3), /(3, 2-5, 3-5),  *(240, [3-1, 3-2, 4-1, 4-2]), *(6, [3-3, 3-4]),
   *(6, [4-3, 5-3]), +(7, [4-4, 5-4, 5-5]), *(30, [4-5, 4-6]), *(6, [5-1, 5-2]),
   +(9, [5-6, 6-6]), +(8, [6-1, 6-2, 6-3]), /(2, 6-4, 6-5)
  ]
).
```

```
?- fd_set_vector_max(255), kenken_testcase(N,C), kenken(N,C,T).
```

where **T** is the final solution matrix.

Preconditions

You can assume the following without checking:

- N (size of the KenKen matrix) and C (the constraints) are **ground terms**, ie. does not contain variables.

- N is a non-negative integer that is less than **vector_max** in GNU-Prolog's finite domain solver (this should be 127)

Aside: An example of why we have vector_max…

```
| ?- X #\= 256.
X = _#2(0..127@)
yes
| ?- X #\= 256, X = 299.
Warning: Vector too small - maybe lost solutions (FD Var:_2)
no
```

Finite Domain Constraint Solver

Idea: Constrain the variable, before its value is known.

Example:

```
| ?- X > 10, X < 20.
uncaught exception: error(instantiation_error,(>)/2)
| ?- X #> 10, X #< 20.
X = _#2(11..19)
yes
| ?- X #< 10, X #> 2, X = 3.
X = 3
yes
```

BUT:

```
| ?- X #> -10.
X = _#0(0..268435455)
yes
| ?- X #> -10, X = -1.
no
```

Constraining lists:

```
| ?- X #> 0, X #< 10, X = [1,2,3].
no
| ?- fd_domain([1,2,3], 0, 10).
yes
```

Other predicates you can use

see http://www.gprolog.org/manual/html_node/gprolog054.html

Your task

1. Write a KenKen solver using finite domain solvers (this should run faster!)

2. Write a KenKen solver **without** using FD solvers (call `plain_kenken`). This means that you cannot use the predicates described in the previous slide. This should run slow.

3. Measure the performance difference between 1. and 2.