Programming Paradigms: Logic Programming

Modelling, Part 2

Víctor Gutiérrez Basulto

Sum-Free

- Problem: A set X of numbers is called *sum-free* if the sum of two elements of X never belongs to X.

  For instance, the set $\{5, \ldots, 9\}$ is sum-free; the set $\{4, \ldots, 9\}$ is not ($4 + 4 = 8$, $4 + 5 = 9$).

- Can we partition the set $\{1, \ldots, n\}$ into 2 sum-free subsets? This is possible if n = 4: both $\{1, 4\}$ and $\{2,3\}$ are sum-free. But if n = 5 then such a partition does not exist.

Exercises

```
% Partition { 1,..., n } into r sum-free sets
% Input: in/2 representing partitions, pos. integers n, r

1{in(I, 1..r)}1 :- I = 1..n.
% achieved: set { 1,...,n} partitioned into subsets
{I:in(I,1)}, ..., { I:in(I,r)}
```

TO DO

```
% Achieve these subsets are sum-free
```

- Say we save the solution in `solution.pl`

```
clingo -c r= somenumber1  -c n = somenumber2  solution.pl
```

Independent Sets

- Def. A set S of vertices in a graph is independent if no two vertices from S are adjacent.

```
% Find independent sets of vertices of size n
% Input: set node/1 of vertices of a graph G;
% set edge/2 of edges of G, positive integer n.

n {in(X) : node(X)}n.
% achieved : in/1 is a set consisting of n vertices
```

TO DO

```
% achieved: in/1 has no pairs of adjacent vertices
# show in/1.
```

Clique

- Def. A set S of vertices in a graph is called a clique if every two distinct vertices in it are adjacent.

  TO DO Modify the (completed) program for independent sets to describe cliques of size n.

Number of Vertices and Edges

- Def. The degree of a node X is the number of nodes adjacent to X.

```
% Find the number of edges and degrees of vertices
% Input: set of nodes/1 of vertices of a graph G; set
edge/2 of edges of G.

adj(X,Y) :- edge(X,Y).
adj(X,Y) :- edge(Y,X).
% achieved: adjacent (X,Y) iff X,Y are adjacent in the
graph.
```

- HINT: use #count

Largest Independent Sets of Vertices

TO DO Modify the (completed) program for independent sets to find the largest independent set of vertices in a graph.

Classes

- Calculate the number of classes taught on each of the five floors of the CS building.

Classes

```
% input: set where/2 of all pairs (C,I) such that
%        class C is taught on the I-th floor.

--------------------------------------------------
% achieved: howmany(I,N) iff the number of classes
%           taught on the I-th floor is N.

#show howmany/2.
```

Magic Square

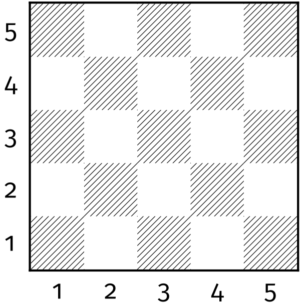
- A magic square is an $n \times n$ square grid filled with distinct integers in the range $1, \cdots, n^2$ so that the sum of numbers in each row, each column, and each of the two diagonals equals the same "magic constant."

Magic Square

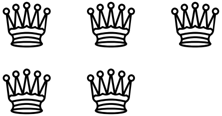
```
% Magic squares of size n

% input: positive integer n.

1 {num(R,C,1..n*n)} 1 :- R=1..n, C=1..n.
% achieved: every square of the grid is filled with
%           a number between 1 and n^2.

R1=R2 :- num(R1,_,X), num(R2,_,X).
C1=C2 :- num(_,C1,X), num(_,C2,X).
% achieved: different squares are filled with different
%           numbers.

% Magic constant: (1+2+...+n^2)/n.
#const magic=(n**3+n)/2.


% achieved: every row sums up to magic.


% achieved: every column sums up to magic.

:-
:-
% achieved: both diagonals sum up to magic.
```

The n-Queens Problem

The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another



Defining the field

queens-listing1.lp

```
row(1..n).
col(1..n).
```

- Create file `queens-listing1.lp`
- Define the field
  - n rows
  - n columns

Defining the field

Running ...

```
$ clingo queens-listing1.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
SATISFIABLE

Models      : 1
Time        : 0.000
```

Placing some queens

queens-listing1.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
```

- Guess a solution candidate
  by placing some queens on the board

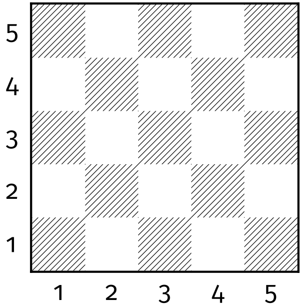
Placing some queens

Running . . .

```
$ clingo -n 3 queens-listing1.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) queen(1,1)
Answer: 3
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) queen(2,1)
SATISFIABLE

Models       : 3+
```

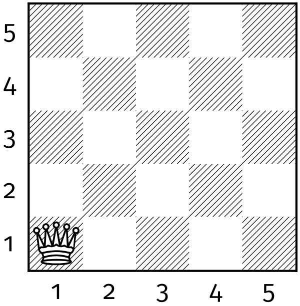
Placing some queens

`Answer: 1`



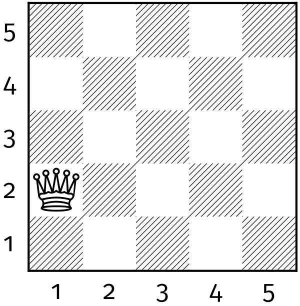
Placing some queens

`Answer: 2`



Placing some queens

`Answer: 3`



Placing n queens

queens-listing1.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n {queen(I,J)} n.
```

- Place exactly n queens on the board

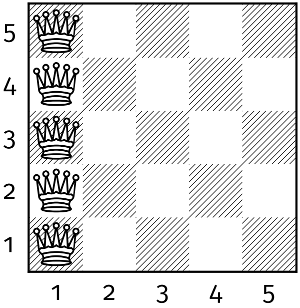
Placing *n* queens

Running ...

```
$ clingo -n 2 queens-listing1.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,1) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(1,2) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
```

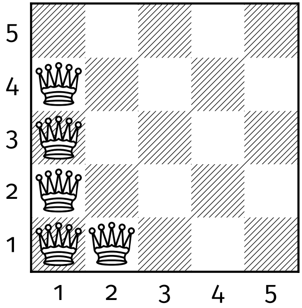
Placing *n* queens

`Answer: 1`



Placing *n* queens

```
Answer: 2
```



Horizontal and vertical attack

queens-listing1.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : col(I), row(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J != JJ.
```

- Forbid horizontal attacks

Horizontal and vertical attack

queens-listing1.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : col(I), row(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
```

- Forbid horizontal attacks
- Forbid vertical attacks

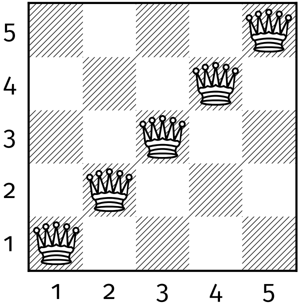
Horizontal and vertical attack

Running ...

```
$ clingo queens-listing1.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,5) queen(4,4) queen(3,3) queen(2,2) queen(1,1)
```

Horizontal and vertical attack

```
Answer: 1
```



Diagonal attack

queens-listing1.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : col(I), row(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

- Forbid diagonal attacks

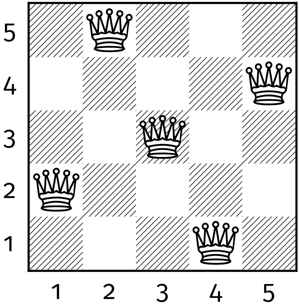
Diagonal attack

Running . . .

```
$ clingo queens-listing1.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(4,5) queen(1,4) queen(3,3) queen(5,2) queen(2,1)
SATISFIABLE

Models       : 1+
Time         : 0.000
```

Diagonal attack

`Answer: 1`



```
clingo --stats --const n=14 queens-listing1.lp

...

Time           : 18.356s (Solving: 18.34s 1st Model: 18.34s Unsat: 0.00s)
CPU Time       : 18.340s

...
```

```
clingo --mode=gringo --text queens-listing1.lp
```

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

Grounded:

```
:- queen(3,1), queen(3,2).
:- queen(3,2), queen(3,1).
```

Remove redundancy with symmetric breaking: listing 2

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J < JJ.
:- queen(I,J), queen(II,J), I < II.
:- queen(I,J), queen(II,JJ), I < II, I-J == II-JJ.
:- queen(I,J), queen(II,JJ), I < II, I+J == II+JJ.
```

```
clingo --stats --const n=14 queens-listing2.lp

...

Time             : 18.269s (Solving: 18.26s 1st Model: 18.26s Unsat: 0.00s)
CPU Time         : 18.256s

...
```

```
clingo --mode=gringo --text queens-listing2.lp
```

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,JJ), J < JJ.
:- queen(I,J), queen(II,J), I < II.
:- queen(I,J), queen(II,JJ), I < II, I-J == II-JJ.
:- queen(I,J), queen(II,JJ), I < II, I+J == II+JJ.
```

For each column, there must be a single queen.
For each row, there must be a single queen.

Reduce the number of grounded instances: listing 3

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
% :- not n { queen(I,J) } n.   Redundant

:- col(J), not 1 { queen(I,J) } 1.
:- row(I), not 1 { queen(I,J) } 1.
:- queen(I,J), queen(II,JJ), I < II, I-J == II-JJ.
:- queen(I,J), queen(II,JJ), I < II, I+J == II+JJ.
```

```
clingo --stats --const n=14 queens-listing3.lp

...

Time         : 0.010s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.008s

...
```

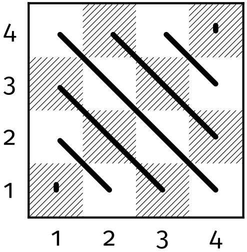
```
clingo --mode=gringo --text queens-listing3.lp
```

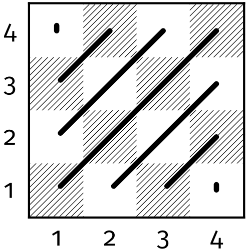
Exercise: last improvement in Chapter 8, Section 8.1

- See slides below.

Enumerating Diagonals

$N = 4$





Enumerating Diagonals

$N = 4$

4	**4**	**5**	**6**	**7**
3	**3**	**4**	**5**	**6**
2	**2**	**3**	**4**	**5**
1	**1**	**2**	**3**	**4**
	1	2	3	4

4	**7**	**6**	**5**	**4**
3	**6**	**5**	**4**	**3**
2	**5**	**4**	**3**	**2**
1	**4**	**3**	**2**	**1**
	1	2	3	4

Enumerating Diagonals

$N = 4$

4	4	5	6	7
3	3	4	5	6
2	2	3	4	5
1	1	2	3	4
	1	2	3	4

$\#\text{diagonal}_1 =$
$(\#\text{row} + \#\text{column}) - 1$

4	7	6	5	4
3	6	5	4	3
2	5	4	3	2
1	4	3	2	1
	1	2	3	4

$\#\text{diagonal}_2 =$
$(\#\text{row} - \#\text{column}) + N$

Enumerating Diagonals

$N = 4$

4	4	5	6	7
3	3	4	5	6
2	2	3	4	5
1	1	2	3	4
	1	2	3	4

$\#\text{diagonal}_1 = (\#\text{row} + \#\text{column}) - 1$

4	7	6	5	4
3	6	5	4	3
2	5	4	3	2
1	4	3	2	1
	1	2	3	4

$\#\text{diagonal}_2 = (\#\text{row} - \#\text{column}) + N$

Reduce the number of grounded instances: listing 4

```
row(1..n).
col(1..n).

{ queen(I,J) : row(I), col(J) }.

:- col(J), not 1 { queen(I,J) } 1.
:- row(I), not 1 { queen(I,J) } 1.
:- D= 1..n*2-1, not { queen(I,J): D==I-J+n} 1.
:- D= 1..n*2-1, not { queen(I,J): D==I+J-1} 1.
```

Performance

1. Monitor the time spent (e.g. `--stats`) and the output size (e.g. `clingo --mode=gringo --text`)
2. Once identified, reformulate "critical" logic program parts.

Mandatory reading: Sections 3.2, 8.1, and 8.4 of of Answer Set Solving in Practice, by Gebser, Kaminski, Kaufmann, Schaub