Answer Set Programming

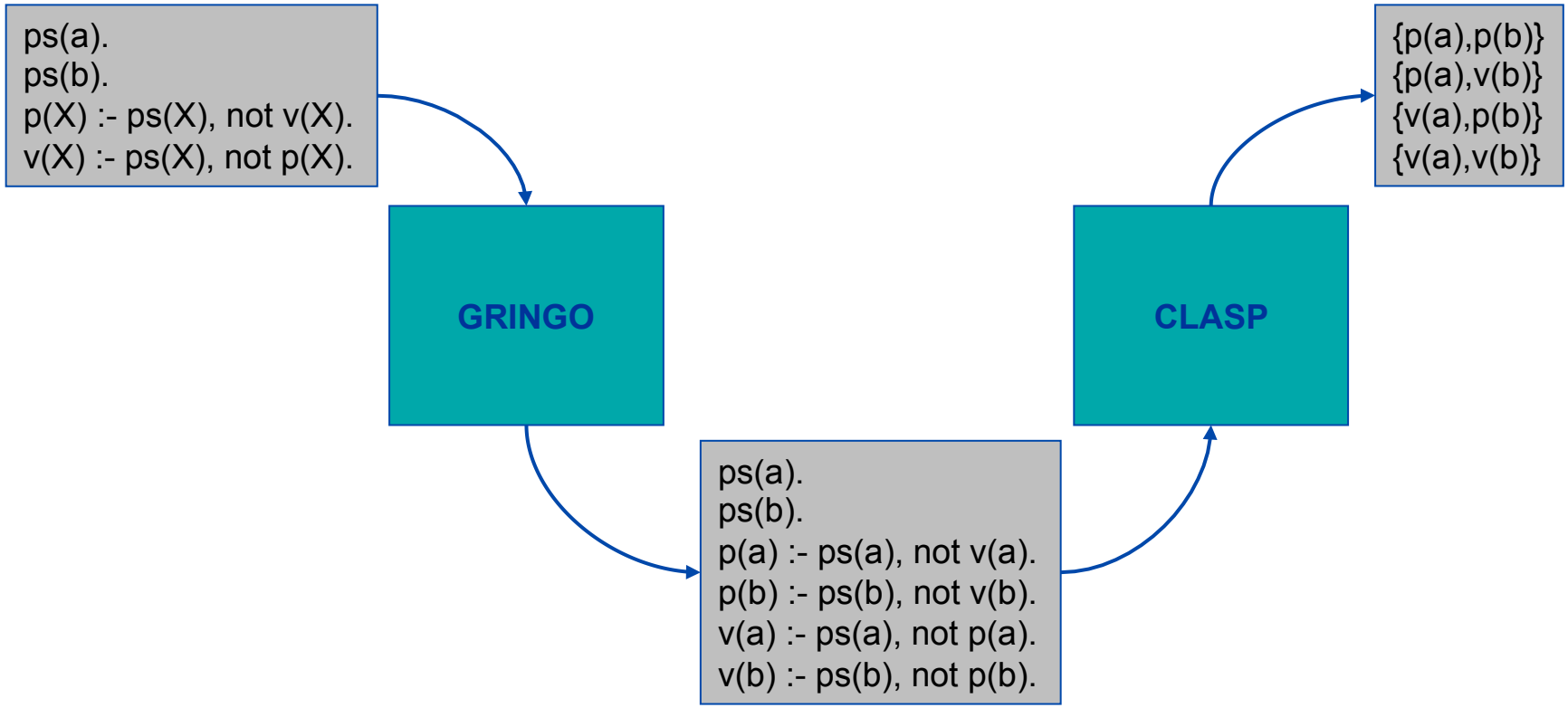
João Leite

Answer Set Programming

- The expressiveness and the declarative character of logic programming under the stable models semantics…
- The existence of software that allows for the fast computation of stable models:
  - CLINGO: https://potassco.org/
  - and many more...
- …led to the application of this paradigm in solving certain problems, leading to the introduction of …

Answer Set Programming

CLINGO = GRINGO + CLASP



- Command line:

  clingo filename 0

- 0 – Returns all answer sets.

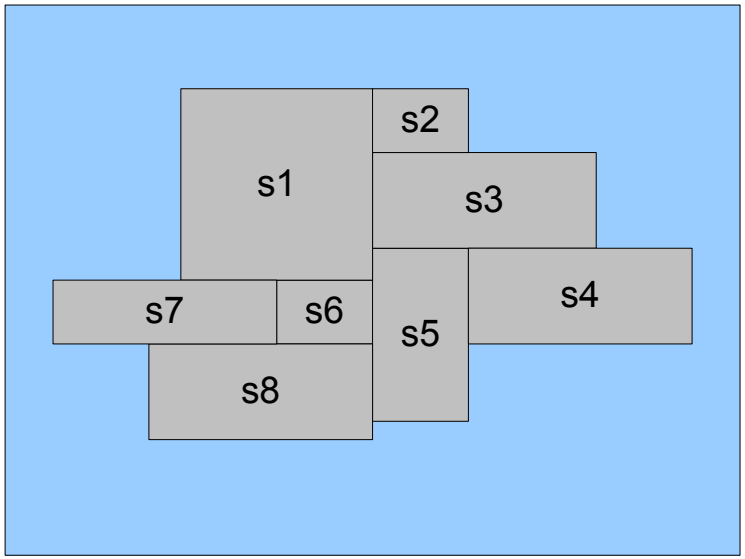
Answer Set Programming

- Concept:
  - Represent the problem as a logic program such that its stable models (answer sets) correspond to the possible solutions (answers).

one stable model = one solution

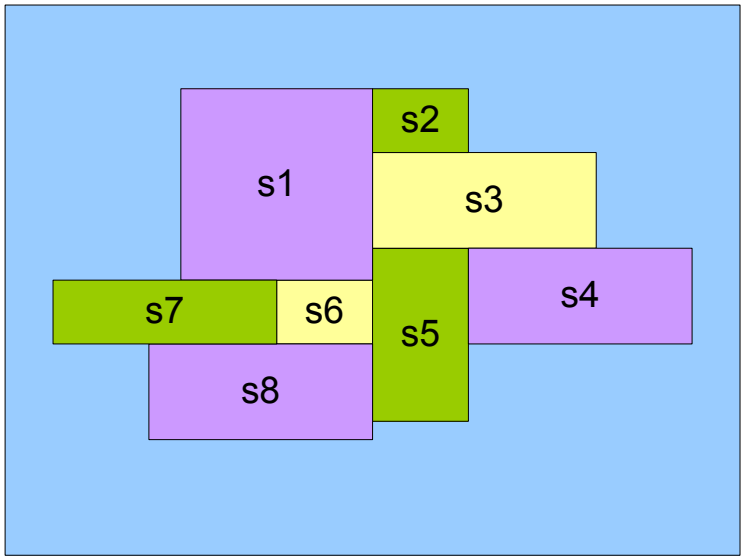
- 3 steps:
  - Determine the format of the solutions to the problem; normally, there will be one (or more) predicates that will act as “containers” to the solution;
  - Generate all possible models, where each model represents one hypothetical solution to the problem;
  - Eliminate the models that do not obey the problem specification.

Map Colouring



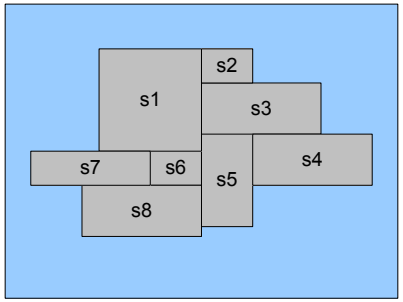
- Problem: find all colourings of a map of countries using not more than 3 colours, such that neighbouring countries are not given the same colour.

Map Colouring



- Problem: find all colourings of a map of countries using not more than 3 colours, such that neighbouring countries are not given the same colour.

ASP – Map Colouring



- Problem: find all colourings of a map using not more than 3 colours, such that neighbouring states are not given the same colour. **Solution 1**:

state(s1). … state(s8).

border(s1,s2). border(s1,s7). … border(s5,s4).

these facts encode the given map (graph).

colour(red). colour(green). colour(blue).

these facts encode the three colours.

painted(S,C) :- state(S), colour(C), not n_painted(S,C).

n_painted(S,C) :- state(S), colour(C), not painted(S,C).

these two rules generate all models resulting from the possible combinations where each state (S) either has a colour (C) (painted(S,C) is true) or doesn't have a colour (painted(S,C) is false, thus n_painted(S,C) is true).

is_painted(S) :- state(S), colour(C), painted(S,C).

:- state(S), not is_painted(S).

predicate is_painted(X) is defined such that it belongs to an answer set every time state S has at least one colour. Therefore, the two rules eliminate all answer sets where there is a state without a colour.

:- state(S), colour(C1), colour(C2), neq(C1,C2), painted(S,C1), painted(S,C2).

this rule eliminates all answer sets with one state painted with two different colours.

:- state(S1), state(S2), colour(C), border(S1,S2), painted(S1,C), painted(S2,C).

this rule eliminates all answer sets with two neighbor states painted with the same colour.

Stable Models - Restrictions

- To prevent the existence of stable models where CONDITION is true, we add the rule:

  :- CONDITION.

  - For example, to prevent the existence of stable models where, simultaneously, atoms a and b are true, and c and d are false, we add the rule :

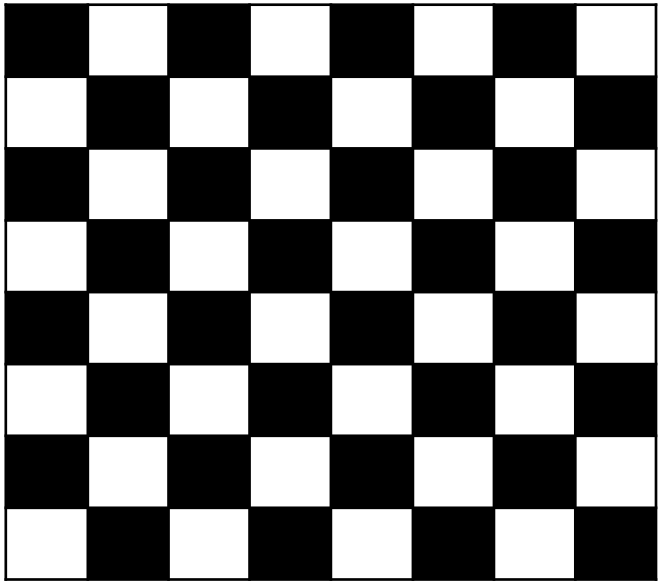
    :- a, b, not c, not d.

  - For example, for only allowing the existence of stable models where, simultaneously, atoms a and b are true, and c and d are false, we can add the pair of rules (where β is a new atom):

    β :- a, b, not c, not d.

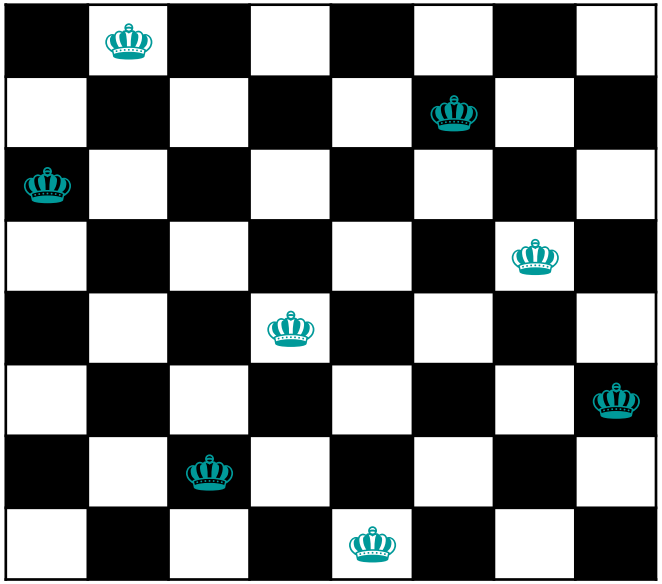
    :- not β.

8 Queens



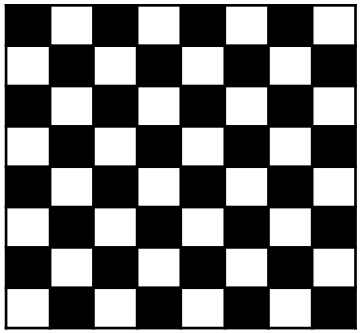
- The 8 queens puzzle is the problem of placing 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal.

8 Queens



- The 8 queens puzzle is the problem of placing 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal.

ASP – 8 Queens



The 8 queens puzzle is the problem of placing 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal. **Solution 1**:

```
column(1). … column(8).      row(1). … row(8).
```

these facts define the domain of column and row.

```
in(X,Y) :- column(X), row(Y), not n_in(X,Y).
n_in(X,Y) :- column(X), row(Y), not in(X,Y).
```

these two rules generate all models resulting from the possible combinations where each cell (X,Y) either has a queen (in(X,Y) is true) or is empty (in(X,Y) is false, thus n_in(X,Y) is true).

```
has_queen(X) :- column(X), row(Y), in(X,Y).
:- column(X), not has_queen(X).
```

predicate has_queen(X) is defined such that it belongs to an answer set every time column X has at least one queen. Therefore, the two rules eliminate all answer sets with a column without a queen.

```
:- column(X), row(Y), column(XX), not eq(X,XX), in(X,Y), in(XX,Y).
```

this rule eliminates all answer sets with more than one queen in the same row (Y).

```
:- column(X), row(Y), row(YY), not eq(Y,YY), in(X,Y), in(X,YY).
```

this rule eliminates all answer sets with more than one queen in the same column (X).

```
:- column(X), row(Y), column(XX), row(YY), not eq(X,XX), not eq(Y,YY),
   in(X,Y), in(XX,YY), eq(abs(X-XX),abs(Y-YY)).
```

this rule eliminates all answer sets with more than one queen in the same diagonal.

CLINGO = GRINGO + CLASP

- n(a;b;c).
  - is equivalent to n(a), n(b), n(c).
- const max = 10
  - defines the value of the constant max as being equal to 10.
- s(1..max).
  - is equivalent to s(1), s(2), …, s(10).
- eq(X,Y) ⇔ X=Y; lt(X,Y) ⇔ X<Y; ge(X,Y) ⇔ X>=Y; …
- hide p(_,_).
  - hides atoms of the form p(_,_) from the answer-sets generated by CLINGO;
- hide.
  - hides all atoms from the answer-sets generated by CLINGO;
- show q(_,_).
  - invalidates a previous hide instruction for atoms of the form q(_,_);

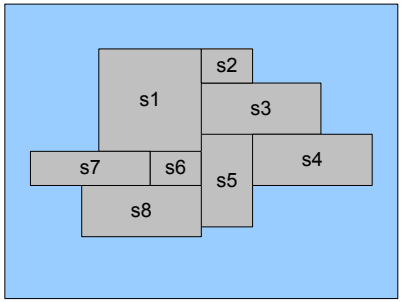
CLINGO = GRINGO + CLASP

- CLINGO offers a special syntax.
- This syntax allows the selective generation of models, eliminating the need to use:
  - auxiliary atoms,
  - cycles to generate models,
  - some integrity constraints.
- Its use increases efficiency of CLINGO, and is strongly advised.
- The general syntax is:

$$\mathbf{min\{p(X_1,\ldots,X_n,Y_1,\ldots,Y_m):q(X_1,\ldots,X_n)\}max :- s(Y_1,\ldots,Y_m).}$$

- Meaning: for each $s(Y_1,\ldots,Y_m)$, generate all possible models with a minimum min and a maximum max of atoms $p(X_1,\ldots,X_n,Y_1,\ldots,Y_m)$, where the domain of $X_1,\ldots,X_n$ is given by $q(X_1,\ldots,X_n)$.
- More details can be found in CLINGO manual.

ASP – Map Colouring



- Problem: find all colourings of a map using not more than 3 colours, such that neighbouring states are not given the same colour. **Solution 2**:

```
state(s1). … state(s8).
border(s1,s2). border(s1,s7). … border(s5,s4).
```

these facts encode the given map (graph).

```
colour(red). colour(green). colour(blue).
```

these facts encode the three colours.

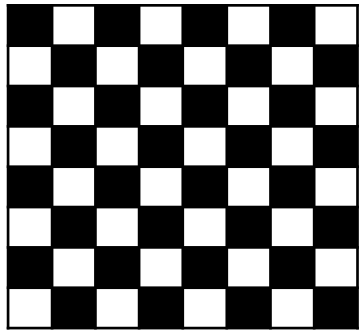
```
1{painted(S,C):colour(C)}1 :- state(S).
```

this rule generates all models where each state has exactly one colour.

```
:- state(S1), state(S2), colour(C), border(S1,S2),
   painted(S1,C), painted(S2,C).
```

this rule eliminates all answer sets with two neighbor states painted with the same colour.

ASP – Rainhas



The 8 queens puzzle is the problem of placing 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal. **Solution 2**:

column(1..8).

defines the domain of column. Equivalent to the facts column(1), column(2),…, column(8).

row(1..8).

defines the domain of row. Equivalent to the facts row(1), row(2),…, row(8).

1{in(X,Y):column(X)}1 :- row(Y).

this rule generates all models where each row Y has a queen in exactly one column X, eliminating all other models.

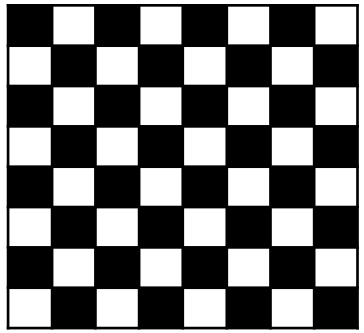
:- column(X), row(Y), row(YY), not eq(Y,YY), in(X,Y), in(X,YY).

this rule eliminates all answer sets with more than one queen in the same column (X).

:- column(X), row(Y), column(XX), row(YY), not eq(X,XX), not eq(Y,YY), in(X,Y), in(XX,YY), eq(abs(X-XX),abs(Y-YY)).

this rule eliminates all answer sets with more than one queen in the same diagonal.

ASP – Rainhas



The 8 queens puzzle is the problem of placing 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal. **Solution 3**:

```
column(1..8).
```

defines the domain of column. Equivalent to the facts column(1), column(2),…, column(8).

```
row(1..8).
```

defines the domain of row. Equivalent to the facts row(1), row(2),…, row(8).

```
1{in(X,Y):column(X)}1 :- row(Y).
```

this rule generates all models where each row Y has a queen in exactly one column X, eliminating all other models.

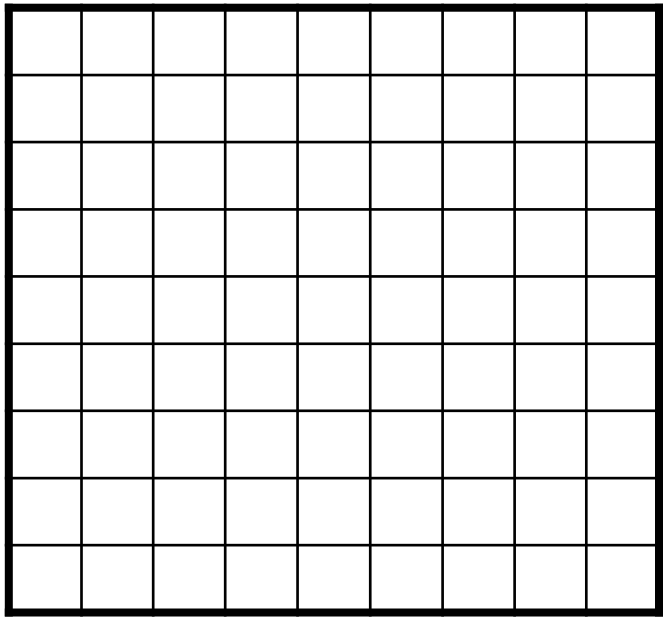
```
1{in(X,Y):row(Y)}1 :- column(X).
```

this rule generates all models where each column X has a queen in exactly one row Y, eliminating all other models.

```
:- column(X), row(Y), column(XX), row(YY), not eq(X,XX), not
      eq(Y,YY), in(X,Y), in(XX,YY), eq(abs(X-XX),abs(Y-YY)).
```

this rule eliminates all answer sets with more than one queen in the same diagonal.

Latin Squares



- The Latin square puzzle is the problem of placing n different symbols (e.g. numbers), is an n × n table, in such a way that each symbol occurs exactly once in each row and exactly once in each column.

Latin Squares

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

- The Latin square puzzle is the problem of placing n different symbols (e.g. numbers), is an n × n table, in such a way that each symbol occurs exactly once in each row and exactly once in each column.

ASP – Latin Squares

- The Latin square puzzle is the problem of placing n different symbols (e.g. numbers), is an n × n table, in such a way that each symbol occurs exactly once in each row and exactly once in each column.
- Solution 1:

```
const size = 10.
```

  - defines the constant size = 10

```
n(1.. size).
```

  - defines the domain of n. Equivalent to the facts n(1),n(2),…,n(size).

```
1{in(X,Y,N):n(N)}1:- n(Y),n(X).
```

  - this rule generates all models where each cell (X,Y) is filled with exactly one number N, eliminating all other models.

```
:- n(X;XX;Y;N), in(X,Y,N), in(XX,Y,N), not eq(X,XX).
```

  - this rule eliminates all models where the same number N is placed in more than one column (X and XX) in the same row Y.

```
:- n(X;YY;Y;N), in(X,Y,N), in(X,YY,N), not eq(Y,YY).
```

  - this rule eliminates all models where the same number N is placed in more than one row (Y and YY) in the same column Y.

ASP – Latin Squares

- The Latin square puzzle is the problem of placing n different symbols (e.g. numbers), is an n × n table, in such a way that each symbol occurs exactly once in each row and exactly once in each column.
- Solution 2:

```
const size = 10.
```

  - defines the constant size = 10

```
n(1.. size).
```

  - defines the domain of n. Equivalent to the facts n(1),n(2),…,n(size).

```
1{in(X,Y,N):n(N)}1:- n(Y),n(X).
```

  - this rule generates all models where each cell (X,Y) is filled with exactly one number N, eliminating all other models.

```
1{in(X,Y,N):n(Y)}1:- n(X),n(N).
```

  - this rule generates all models where in each column (X), each number N is placed in exactly in one row (Y), eliminating all other models.

```
1{in(X,Y,N):n(X)}1:- n(Y),n(N).
```

  - this rule generates all models where in each row (Y), each number N is placed in exactly in one column (X), eliminating all other models.

Sudoku

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

- The Sudoku puzzle is the problem of filling a partially completed 9x9 grid so that each column, each row, and each of the nine 3x3 boxes (also called blocks or regions) contains the digits from 1 to 9.

Sudoku

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

- The Sudoku puzzle is the problem of filling a partially completed 9x9 grid so that each column, each row, and each of the nine 3x3 boxes (also called blocks or regions) contains the digits from 1 to 9.

ASP – Sudoku

- The Sudoku puzzle is the problem of filling a partially completed 9x9 grid so that each column, each row, and each of the nine 3x3 boxes (also called blocks or regions) contains the digits from 1 to 9.
- Sudoku is a particular case of the Latin Squares with n=9, to which we add the regions constraint. We can start with the Latin Square specification and simply add the new constraint. Solution:

```
n(1.. 9).
1{in(X,Y,N):n(N)}1:- n(Y),n(X).
1{in(X,Y,N):n(Y)}1:- n(X),n(N).
1{in(X,Y,N):n(X)}1:- n(Y),n(N).
v(1;4;7).
```

this fact defines the coordinates of the lower left cell of each 3x3 region. Equivalent to the facts v(1),v(4), e v(7).

```
:- n(X;Y;X1;Y1;N), v(VX;VY), neq(X,X1), neq(Y,Y1), in(X,Y,N), in(X1,Y1,N),
   X-VX < 3, X1-VX < 3, Y-VY < 3, Y1-VY < 3,
   X >= VX, X1 >= VX, Y >= VY, Y1 >= VY.
```

This rule eliminates all models where the same number N is placed in more than one cell in the same region.

Note that this rule only verifies pairs of cells where both the column and row are different, since other cases are already treated by the rules above.

ASP – Sudoku

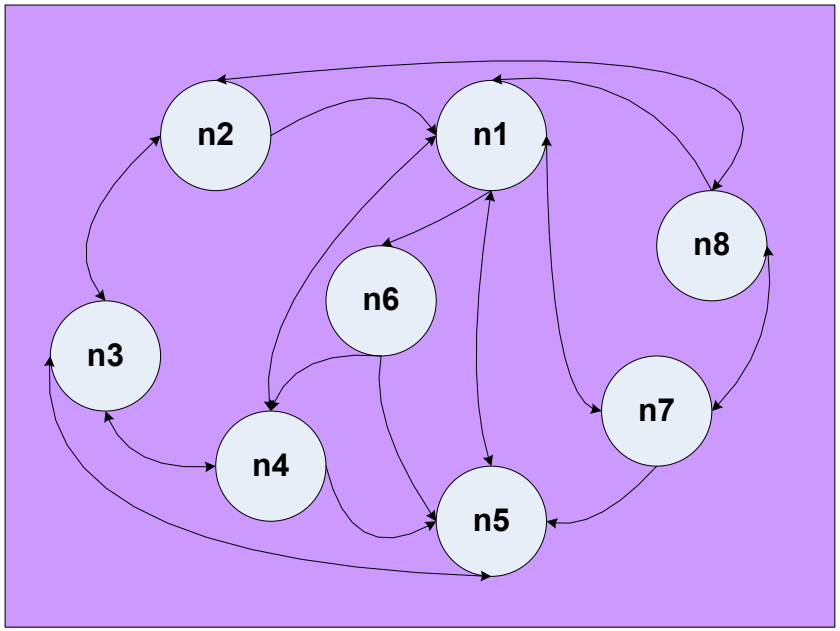
		6					9	
			5		1	7		
2			9			3		
	7			3			5	
	2			9			6	
	4			8			2	
		1			3			4
		5	2		7			
	3					8		

- Now, we only have to add facts for the cells that are already filled.

```
n(1.. 9).
1{in(X,Y,N):n(N)}1:- n(Y),n(X).
1{in(X,Y,N):n(Y)}1:- n(X),n(N).
1{in(X,Y,N):n(X)}1:- n(Y),n(N).
v(1;4;7).
:- n(X;Y;X1;Y1;N), v(VX;VY), neq(X,X1), neq(Y,Y1), in(X,Y,N), in(X1,Y1,N),
   X-VX < 3, X1-VX < 3, Y-VY < 3, Y1-VY < 3,
   X >= VX, X1 >= VX, Y >= VY, Y1 >= VY.
```

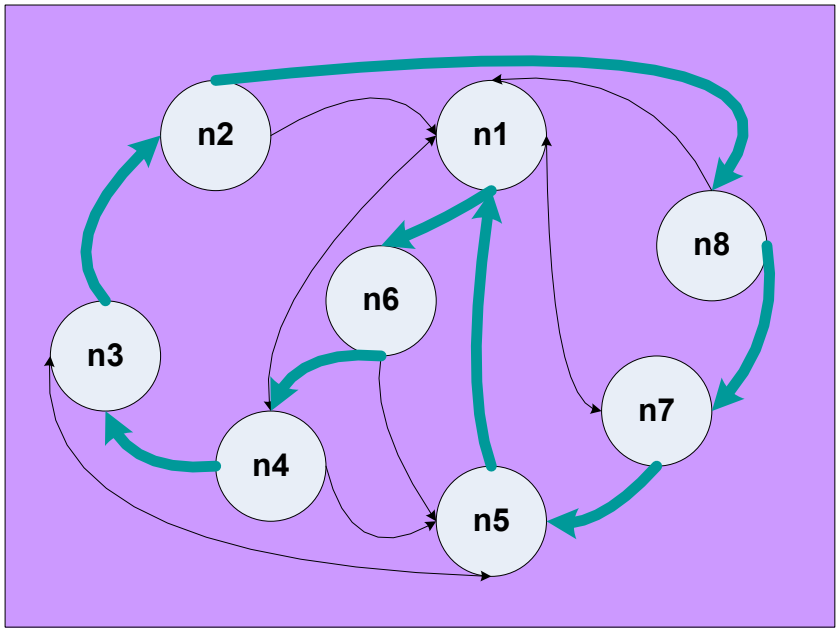
```
in(1,7,2).  in(2,1,3).  in(2,4,4).  in(2,5,2).  in(2,6,7).
in(3,2,5).  in(3,3,1).  in(3,9,6).  in(4,2,2).  in(4,7,9).
in(4,8,5).  in(5,4,8).  in(5,5,9).  in(5,6,3).  in(6,2,7).
in(6,3,3).  in(6,8,1).  in(7,1,8).  in(7,7,3).  in(7,8,7).
in(8,4,2).  in(8,5,6).  in(8,6,5).  in(8,9,9).  in(9,3,4).
```

Hamiltonian Cycles



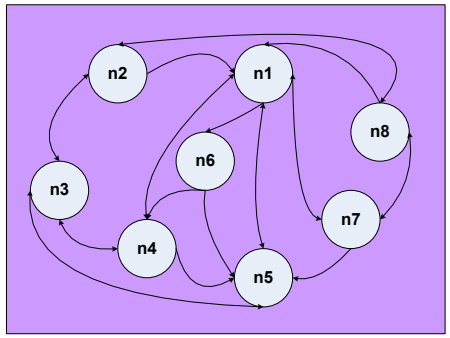
- Problem: given a directed graph, find all Hamiltonian Cycles i.e. all cycles that touch each node exactly once.

Hamiltonian Cycles



- Problem: given a directed graph, find all Hamiltonian Cycles i.e. all cycles that touch each node exactly once.

ASP – Hamiltonian Cycles



- Problem: given a directed graph, find all Hamiltonian Cycles i.e. all cycles that touch each node exactly once.

node(n1). … node(n8).

edge(n1,n5). edge(n4,n3). … edge(n7,n8).

these facts encode the given graph.

1 {in(X,Y) : edge(X,Y)} 1 :- node(X).

this rule generates all models where the cycle contains, for each node, exactly one edge going out.

1 {in(X,Y) : edge(X,Y)} 1 :- node(Y).

this rule generates all models where the cycle contains, for each node, exactly one edge going in.

reachable(X) :- node(X), in(n1,X).

reachable(Y) :- node(X), node(Y), reachable(X), in(X,Y).

these rules define the notion of reachability, starting from node n1.

:- not reachable(X), node(X).

this rule eliminates all answer sets where some node is not reachable.