Logical Modeling using Answer Set Programming

Research Seminar, SoSe 2020

Lukas Schweizer
mailto:lukas.schweizer@tu-dresden.de

https://iccl.inf.tu-dresden.de/web/Seminar:_Logical_Modelling_(SS2020)

April 20, 2020

Answer Set Programming
What is ASP?

- ASP is a *declarative problem solving approach*.
  $\rightsquigarrow$ Syntactically similar to `Prolog`, but truely declarative.
- For a given ASP "program", we compute *models* representing a solution for the encoded problem.
- Very efficient implementations are available, for computing models.
- We will use the `Potassco` tools `Gringo`, `Clasp` and `Clingo`.
  $\rightsquigarrow$ `http://potassco.sourceforge.net/`

Sudoku

The famous number riddle Sudoku represents a constraint problem, typically defined on a 9×9 *board, where numbers* $1 \dots 9$ *are placed on each cell. The goal is to complete a given board such that in each row, column, and square the numbers* $1 \dots 9$ *occur exactly once.*

Sudoku

The famous number riddle Sudoku represents a constraint problem, typically defined on a 9×9 *board, where numbers* $1 \ldots 9$ *are placed on each cell. The goal is to complete a given board such that in each row, column, and square the numbers* $1 \ldots 9$ *occur exactly once.*

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

Unsolved Sudoku

Sudoku

The famous number riddle Sudoku represents a constraint problem, typically defined on a 9×9 *board, where numbers* $1 \ldots 9$ *are placed on each cell. The goal is to complete a given board such that in each row, column, and square the numbers* $1 \ldots 9$ *occur exactly once.*

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

Unsolved Sudoku

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

Solved Sudoku

Sudoku
How to start modeling?

Declarative means we *declare* and *describe* the things we know;

Sudoku

How to start modeling?

Declarative means we *declare* and *describe* the things we know; i.e. the notion of

- The game board, cells, rows, columns and squares.

Sudoku
How to start modeling?

Declarative means we *declare* and *describe* the things we know; i.e. the notion of

- The game board, cells, rows, columns and squares.
- The rules, in detail
  - On each cell there is exactly one number from $\{1 \dots 9\}$.
  - In each row every number occurs exactly once.
  - In each column every number occurs exactly once.
  - In each square every number occurs exactly once.

Sudoku

How to start modeling?

Declarative means we *declare* and *describe* the things we know; i.e. the notion of

- The game board, cells, rows, columns and squares.
- The rules, in detail
  - On each cell there is exactly one number from $\{1 \dots 9\}$.
  - In each row every number occurs exactly once.
  - In each column every number occurs exactly once.
  - In each square every number occurs exactly once.

And there is no need to specify an algorithm on how to solve Sudokus.

Sudoku
Board Encoding

- We encode the board via facts:

$$number(1..9).\ row(0..8).\ column(0..8).$$

$$square(0,0..2,0..2).\ square(1,0..2,3..5).\ square(2,0..2,6..8).$$
$$square(3,3..5,0..2).\ square(4,3..5,3..5).\ square(5,3..5,6..8).$$
$$square(6,6..8,0..2).\ square(7,6..8,3..5).\ square(8,6..8,6..8).$$

Sudoku

Board Encoding

- Alternatively, via rules:

$$
\begin{array}{l}
square(0,X,Y) \text{ :- } row(X),\ column(Y),\ X<3,\ Y<3.\\
square(1,X,Y) \text{ :- } row(X),\ column(Y),\ X<3,\ Y>2,\ Y<6.\\
square(2,X,Y) \text{ :- } row(X),\ column(Y),\ X<3,\ Y>5.\\
square(3,X,Y) \text{ :- } row(X),\ column(Y),\ X>2,\ X<6,\ Y<3.\\
square(4,X,Y) \text{ :- } row(X),\ column(Y),\ X>2,\ X<6,\ Y>2,\ Y<6.\\
square(5,X,Y) \text{ :- } row(X),\ column(Y),\ X>2,\ X<6,\ Y>5.\\
square(6,X,Y) \text{ :- } row(X),\ column(Y),\ X>5,\ Y<3.\\
square(7,X,Y) \text{ :- } row(X),\ column(Y),\ X>5,\ Y>2,\ Y<6.\\
square(8,X,Y) \text{ :- } row(X),\ column(Y),\ X>5,\ Y>5.
\end{array}
$$

Sudoku
Board Encoding

- Grounding the program via

```
    gringo sudoku.lp ---text
```

- Yields instantiated $square$ atoms (beside the known facts):

```
square(0,0,0). square(0,0,1). square(0,0,2). ...
square(1,0,3). square(1,0,4). square(1,0,5). ...
square(2,0,6). square(2,0,7). square(2,0,8). ...
...
```

Sudoku
Rules Encoding

- On each cell there is exactly one number from $\{1 \ldots 9\}$.

$$
\begin{array}{lll}
cell(X,Y,1) & \text{:-} & row(X),\ column(Y), \\
 & & \texttt{not}\ cell(X,Y,2),\ \texttt{not}\ cell(X,Y,3),\ \texttt{not}\ cell(X,Y,4), \\
 & & \texttt{not}\ cell(X,Y,5),\ \texttt{not}\ cell(X,Y,6),\ \texttt{not}\ cell(X,Y,7), \\
 & & \texttt{not}\ cell(X,Y,8),\ \texttt{not}\ cell(X,Y,9). \\
cell(X,Y,2) & \text{:-} & row(X),\ column(Y), \\
 & & \texttt{not}\ cell(X,Y,1),\ \texttt{not}\ cell(X,Y,3),\ \texttt{not}\ cell(X,Y,4), \\
 & & \texttt{not}\ cell(X,Y,5),\ \texttt{not}\ cell(X,Y,6),\ \texttt{not}\ cell(X,Y,7), \\
 & & \texttt{not}\ cell(X,Y,8),\ \texttt{not}\ cell(X,Y,9). \\
 & & \ldots \\
cell(X,Y,9) & \text{:-} & row(X),\ column(Y), \\
 & & \texttt{not}\ cell(X,Y,1),\ \texttt{not}\ cell(X,Y,2),\ \texttt{not}\ cell(X,Y,3), \\
 & & \texttt{not}\ cell(X,Y,4),\ \texttt{not}\ cell(X,Y,5),\ \texttt{not}\ cell(X,Y,6), \\
 & & \texttt{not}\ cell(X,Y,7),\ \texttt{not}\ cell(X,Y,8).
\end{array}
$$

Sudoku

Board Encoding

- What is the meaning of these rules?

$$
\begin{aligned}
cell(X,Y,1) \quad \text{:-} \quad & row(X),\ column(Y),\\
& \texttt{not}\ cell(X,Y,2),\ \texttt{not}\ cell(X,Y,3),\ \texttt{not}\ cell(X,Y,4),\\
& \texttt{not}\ cell(X,Y,5),\ \texttt{not}\ cell(X,Y,6),\ \texttt{not}\ cell(X,Y,7),\\
& \texttt{not}\ cell(X,Y,8),\ \texttt{not}\ cell(X,Y,9).
\end{aligned}
$$

Sudoku
Board Encoding

- What is the meaning of these rules?

$$\begin{array}{lll} cell(X,Y,1) & \text{:-} & row(X),\ column(Y), \\ & & \texttt{not}\ cell(X,Y,2),\ \texttt{not}\ cell(X,Y,3),\ \texttt{not}\ cell(X,Y,4), \\ & & \texttt{not}\ cell(X,Y,5),\ \texttt{not}\ cell(X,Y,6),\ \texttt{not}\ cell(X,Y,7), \\ & & \texttt{not}\ cell(X,Y,8),\ \texttt{not}\ cell(X,Y,9). \end{array}$$

For the cell at position (X, Y) we can place 1, if X corresponds to some row and Y to some column, and we failed to demonstrate that there is already placed one of the other numbers $2 \ldots 9$.

Sudoku

Board Encoding

Since we have one such rule for every number, we can *non-deterministically* choose / guess which number to place on some cell.

We defined the search space with these rules. In fact, without further knowledge, these rules generate all 9^{81} combinations of cell numberings; therefore they are called *guessing rules*.

Sudoku
Board Encoding

Since we have one such rule for every number, we can *non-deterministically* choose / guess which number to place on some cell.

We defined the search space with these rules. In fact, without further knowledge, these rules generate all 9^{81} combinations of cell numberings; therefore they are called *guessing rules*.

We need to restrict the search space such that only proper numberings are generated.

Sudoku
Constraints Encoding

- Each number occurs only once in each row and column.

  :- $cell(X, Y1, N),\ cell(X, Y2, N), Y1$!= $Y2.$
  :- $cell(X1, Y, N),\ cell(X2, Y, N), X1$!= $X2.$

Sudoku
Constraints Encoding

- Each number occurs only once in each row and column.

  :- $cell(X, Y1, N),\ cell(X, Y2, N), Y1 \mathrel{!=} Y2.$
  :- $cell(X1, Y, N),\ cell(X2, Y, N), X1 \mathrel{!=} X2.$

- Each number occurs only once in each square.

$$\begin{array}{lll} in\_square(S,N) & \text{:-} & cell(X,Y,N),\ square(S,X,Y). \\ & \text{:-} & number(N),\ \texttt{not}\ in\_square(S,N), \\ & & square(S,\_,\_). \end{array}$$

Sudoku
Grounding and Solving

- We ground the program and call `clasp`

```
gringo sudoku.lp | clasp
```

Sudoku
Grounding and Solving

```
Solving...
Answer:  1
cell(8,8,1) cell(8,7,4) cell(8,6,7) cell(8,5,2) cell(8,4,9)
cell(8,3,8) cell(8,2,3) cell(8,1,6) cell(8,0,5) cell(7,8,4)
cell(7,7,2) cell(7,6,9) cell(7,5,1) cell(7,4,8) cell(7,3,7)
cell(7,2,6) cell(7,1,5) cell(7,0,3) cell(6,8,8) cell(6,7,9)
cell(6,6,3) cell(6,5,7) cell(6,4,6) cell(6,3,5) cell(6,2,1)
cell(6,1,4) cell(6,0,2) cell(5,8,2) cell(5,7,1) cell(5,6,8)
cell(5,5,4) cell(5,4,7) cell(5,3,9) cell(5,2,5) cell(5,1,3)
cell(5,0,6) cell(4,8,9) cell(4,7,7) cell(4,6,6) cell(4,5,8)
cell(4,4,5) cell(4,3,3) cell(4,2,4) cell(4,1,2) cell(4,0,1)
cell(3,8,7) cell(3,7,8) cell(3,6,5) cell(3,5,9) cell(3,4,3)
cell(3,3,6) cell(3,2,2) cell(3,1,1) cell(3,0,4) cell(2,8,3)
SATISFIABLE
Models :  1+
Time :  0.043s (Solving:  0.01s 1st Model:  0.01s Unsat:  0.00s)
CPU Time :  0.040s
```

Sudoku
Grounding and Solving

- We ground the program and call `clasp`

```
gringo sudoku.lp | clasp
```

- When providing no options `clasp` will compute one answer set in case of satisfiability. We can request more, e.g. 5, via

```
gringo sudoku.lp | clasp ---number 5
```

Sudoku
Grounding and Solving

- We ground the program and call `clasp`

```
gringo sudoku.lp | clasp
```

- When providing no options `clasp` will compute one answer set in case of satisfiability. We can request more, e.g. 5, via

```
gringo sudoku.lp | clasp ---number 5
```

Up to now, what are we generating?

Sudoku
Problem Instance

- We add a concrete Sudoku, given as $cell$ facts.

  $cell(0,0,3).$ $cell(0,4,8).$ $cell(0,6,6).$ $cell(0,8,7).$
  $cell(1,1,1).$ $cell(1,6,4).$ $cell(1,8,9).$
  $cell(2,0,8).$ $cell(2,1,9).$ $cell(2,4,6).$ $cell(2,5,7).$
  $cell(3,1,6).$ $cell(3,3,1).$ $cell(3,4,9).$ $cell(3,6,7).$
  $cell(4,2,9).$ $cell(4,3,6).$ $cell(4,4,5).$ $cell(4,8,2).$
  $cell(5,2,2).$ $cell(5,7,1).$
  $cell(6,1,5).$ $cell(6,4,4).$ $cell(6,8,3).$
  $cell(7,1,4).$ $cell(7,3,2).$ $cell(7,7,9).$ $cell(7,8,8).$
  $cell(8,1,8).$ $cell(8,2,6).$ $cell(8,4,3).$ $cell(8,6,1).$

- There is only one solution, which we can verify by requesting all answer sets

  ```
  gringo sudoku.lp sudoku-instance.lp | clasp ---number 0
  ```

Modeling Paradigm
Guess and Check Programs [?]

For the Sudoku example, we developed two main parts.

- *Problem description*
  Encoding the underlying problem, i.e. the board, rules and constraints.
- *Problem instance*
  Encoding of a concrete instance of the problem; i.e. a partially filled Sudoku.

We can solve any Sudoku with our encoding, the problem description is therefore said to be *uniform*.

Modeling Paradigm
Guess and Check Programs

Two important aspects in the problem description:

- We defined so-called *guessing rules* to generate prospective solutions.
- We then *constrained* guesses in order to rule out those not representing a solution.

These *guess & check* parts are characteristic for answer set programs.

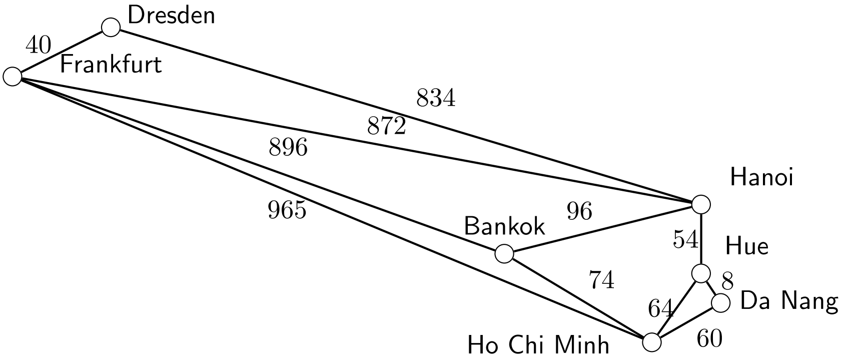
- Motivated by NP-problems.
- Can be seen as "Design-Pattern‘

Traveling Salesman Problem

A salesman is requested to visit some pre-defined cities. In order to be as efficient as possible, he wants to visit every city only once, as well as to travel the shortest roundtrip visiting all cities starting and ending in the same city.

Traveling Salesman Problem

A salesman is requested to visit some pre-defined cities. In order to be as efficient as possible, he wants to visit every city only once, as well as to travel the shortest roundtrip visiting all cities starting and ending in the same city.



Traveling Salesman Problem
Problem Seperation

- We can split the problem into:
  (a) finding roundtrips beginning from and ending in the same city visiting all other cities only once, and

Traveling Salesman Problem
Problem Seperation

- We can split the problem into:
  (a) finding roundtrips beginning from and ending in the same city visiting all other cities only once, and
  (b) computing the length of each roundtrip in order to find the shortest.

Traveling Salesman Problem
Problem Seperation

- We can split the problem into:
  - (a) finding roundtrips beginning from and ending in the same city visiting all other cities only once, and
  - (b) computing the length of each roundtrip in order to find the shortest.
- In fact, the first is the very well-known NP-*complete* problem of finding *Hamiltonian* cycles.

Traveling Salesman Problem
Problem Seperation

- We can split the problem into:
  - (a) finding roundtrips beginning from and ending in the same city visiting all other cities only once, and
  - (b) computing the length of each roundtrip in order to find the shortest.
- In fact, the first is the very well-known NP-*complete* problem of finding *Hamiltonian* cycles.
- For the encoding of the Hamiltonian cycle problem we stick to the guess and check paradigm.

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- Every node in a Hamiltonian cycle has exactly one incoming and one outgoing edge.

$$
\begin{array}{lll}
1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 & \text{:-} & node(X). \\
1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 & \text{:-} & node(Y).
\end{array}
$$

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- Every node in a Hamiltonian cycle has exactly one incoming and one outgoing edge.

$$\begin{array}{lll} 1\ \{\mathit{cycle}(X,Y) : \mathit{edge}(X,Y)\}\ 1 & \text{:-} & \mathit{node}(X). \\ 1\ \{\mathit{cycle}(X,Y) : \mathit{edge}(X,Y)\}\ 1 & \text{:-} & \mathit{node}(Y). \end{array}$$

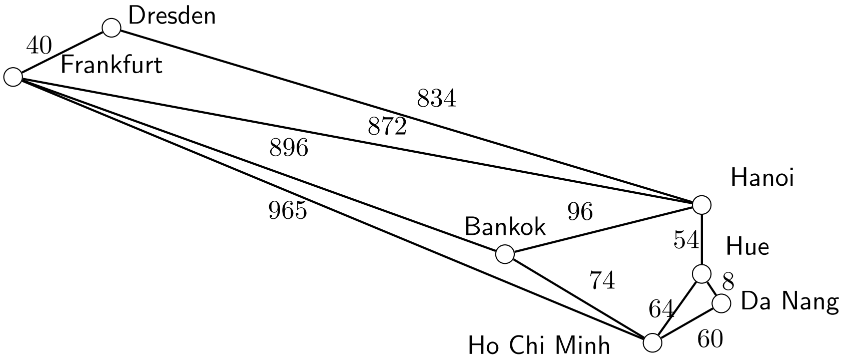
- In a Hamiltonian cycle, every node is reachable.

$$\begin{array}{lll} \mathit{reachable}(Y) & \text{:-} & \mathit{cycle}(s,Y). \\ \mathit{reachable}(Y) & \text{:-} & \mathit{cycle}(X,Y),\ \mathit{reachable}(X). \\ & \text{:-} & \mathit{node}(X),\ \texttt{not}\ \mathit{reachable}(X). \end{array}$$

Traveling Salesman Problem

Hamiltonian Cycle Encoding

- Facts for the input graph.



Traveling Salesman Problem
Hamiltonian Cycle Encoding

- Facts for the input graph.

$node(dresden).$
$node(bankok).$
$node(hue).$
$node(danang).$
$edge(dresden, hanoi).$
$edge(frankfurt, bankok).$
$edge(bankok, hanoi).$
$edge(hochi, hue).$
$edge(danang, hue).$
$edge(Y, X)$:- $edge(X, Y).$

$node(frankfurt).$
$node(hanoi).$
$node(hochi).$
$edge(dresden, frankfurt).$
$edge(frankfurt, hanoi).$
$edge(frankfurt, hochi).$
$edge(bankok, hochi).$
$edge(hochi, danang).$
$edge(hanoi, hue).$

Traveling Salesman Problem

Solving - Computing Hamiltonian Cycles

```
gringo -c s=dresden hamiltonian.lp map.lp | clasp ---n 0
```

```
Solving...
Answer:  1
cycle(dresden,hanoi) cycle(hanoi,bankok)
cycle(bankok,hue) cycle(hue,danang)
cycle(danang,hochi) cycle(hochi, frankfurt)
cycle(frankfurt,dresden)
...
SATISFIABLE
Models :  2
Time :  0.002s (Solving:  0.00s 1st Model:  0.00s Unsat:  0.00s)
CPU Time :  0.000s
```

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- In order to calculate the cycle length, we need the distance information.

$distance(dresden, frankfurt, 40).$
$distance(frankfurt, bankok, 896).$
$distance(bankok, hochi, 74).$
$distance(frankfurt, hanoi, 872).$
$distance(hochi, danang, 60).$
$distance(danang, hue, 8).$
$distance(X, Y, C)$:- $distance(Y, X, C).$

$distance(dresden, hanoi, 834).$
$distance(bankok, hanoi, 96).$
$distance(frankfurt, hochi, 965).$
$distance(hanoi, hue, 54).$
$distance(hochi, hue, 64).$

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- In order to calculate the cycle length, we need the distance information.

$$
\begin{array}{ll}
distance(dresden, frankfurt, 40). & distance(dresden, hanoi, 834). \\
distance(frankfurt, bankok, 896). & distance(bankok, hanoi, 96). \\
distance(bankok, hochi, 74). & distance(frankfurt, hochi, 965). \\
distance(frankfurt, hanoi, 872). & distance(hanoi, hue, 54). \\
distance(hochi, danang, 60). & distance(hochi, hue, 64). \\
distance(danang, hue, 8). & \\
distance(X, Y, C) \text{ :- } distance(Y, X, C). &
\end{array}
$$

- With a so-called *aggregate function*, we can compute the sum of the edge in a cycle.

$$circumference(N) \text{ :- } N = \#sum\ [cycle(X, Y) : distance(X, Y, C) = C].$$

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- In example, for $cycle(dresden, frankfurt)$ and $cycle(frankfurt, hanoi)$, we obtain the grounded rule

$$\begin{aligned} circumference(912) \text{ :- } 912 = \#sum[cycle(dresden, frankfurt) &= 40, \\ cycle(frankfurt, hanoi) &= 872] \end{aligned}$$

Traveling Salesman Problem
Hamiltonian Cycle Encoding

- In example, for $cycle(dresden, frankfurt)$ and $cycle(frankfurt, hanoi)$, we obtain the grounded rule

$$\begin{aligned} circumference(912) \text{ :- } 912 = \#sum[cycle(dresden, frankfurt) &= 40, \\ cycle(frankfurt, hanoi) &= 872]. \end{aligned}$$

- We can compute answer sets including one $circumference$ atom, via

```
gringo -c s=dresden  hamiltonian.lp
                     map.lp
                     distances.lp | clasp ---n 0
```

Traveling Salesman Problem

Minimization

- Answer sets can be computed and enumerated with respect to some optimization value; i.e. the circumference of our cycles.

Traveling Salesman Problem
Minimization

- Answer sets can be computed and enumerated with respect to some optimization value; i.e. the circumference of our cycles.
- In `clasp` objective optimization functions are offered, in our case we use $minimize$.

$$\#minimize\ [circumference(N) = N].$$

Traveling Salesman Problem
Minimization

- Answer sets can be computed and enumerated with respect to some optimization value; i.e. the circumference of our cycles.
- In `clasp` objective optimization functions are offered, in our case we use $minimize$.

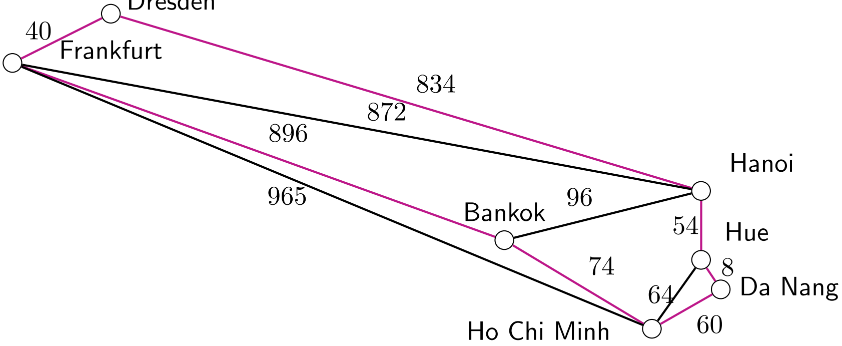
$$\#minimize\ [circumference(N) = N].$$

- Adding the statement to the program yields a minimal hamiltonian cycle of length 1966.

Traveling Salesman Problem
Minimization

```
Answer: 1
cycle(dresden,hanoi) cycle(hanoi,bankok)
cycle(bankok,hue) cycle(hue,danang)
cycle(danang,hochi) cycle(hochi, frankfurt)
cycle(frankfurt,dresden)
Optimization: 1966
...
OPTIMUM FOUND
Models : 1
Optimization: 1966
Time : 1.358s (Solving: 0.40s 1st Model: 0.01s Unsat: 0.39s)
```

Traveling Salesman Problem
Minimization



Literature

Refer to the literature given on the course page:
`https://iccl.inf.tu-dresden.de/web/Seminar:_Logical_Modelling_(SS2020)`

Download `clingo` (and maybe `gringo` and `clasp`) here:

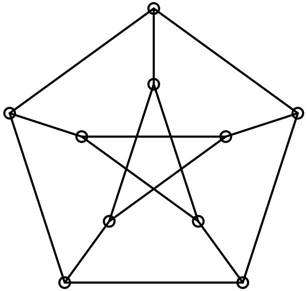
`http://potassco.org/`

BACKUP

Graph Coloring

N-Coloring Problem

Is there some coloring of the vertices of a given graph using n colors, such that no two adjacent vdertices share the same color?



Graph Coloring

$color(green).\ color(red).\ color(blue).$

$$\begin{array}{lcl}
coloring(X, green) & \text{:-} & node(X),\ \texttt{not}\ coloring(X, red), \\
 & & \texttt{not}\ coloring(X, blue). \\
coloring(X, red) & \text{:-} & node(X),\ \texttt{not}\ coloring(X, green), \\
 & & \texttt{not}\ coloring(X, blue). \\
coloring(X, blue) & \text{:-} & node(X),\ \texttt{not}\ coloring(X, green), \\
 & & \texttt{not}\ coloring(X, red).
\end{array}$$

Graph Coloring

$color(green).\ color(red).\ color(blue).$

$$\begin{array}{lll}
coloring(X, green) & \texttt{:-} & node(X),\ \texttt{not}\ coloring(X, red), \\
 & & \texttt{not}\ coloring(X, blue). \\
coloring(X, red) & \texttt{:-} & node(X),\ \texttt{not}\ coloring(X, green), \\
 & & \texttt{not}\ coloring(X, blue). \\
coloring(X, blue) & \texttt{:-} & node(X),\ \texttt{not}\ coloring(X, green), \\
 & & \texttt{not}\ coloring(X, red). \\
 & \texttt{:-} & coloring(X1, C),\ coloring(X2, C), \\
 & & edge(X1, X2).
\end{array}$$

Cannibals and Missionaries

Three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.