Modelling Problems in ASP
Recall:

Modeling and Interpreting

Problem → Modeling → Logic Program → Computation → Answer sets → Interpretation → Solution(s)
General Approach

For solving a problem instance $I$ in problem class $P$, encode

1. the problem instance $I$ as a set of facts $C(I)$ and
2. the problem class $P$ as a set of rules $C(P)$,

such that the solutions to $P$ for $I$ can be extracted from the answer sets of $C(P) \cup C(I)$.
Example: \( n \)-colorability of Graphs

Problem instance

A graph \((V, E)\).

Problem class

Assign each vertex in \( V \) one of \( n \) colors such that no two vertices in \( V \) connected by an edge in \( E \) have the same color.
3–colorability of graphs

### C(I)

- \(\text{vertex}(1) \leftarrow \text{edge}(1,2) \leftarrow\)
- \(\text{vertex}(2) \leftarrow \text{edge}(2,3) \leftarrow\)
- \(\text{vertex}(3) \leftarrow \text{edge}(3,1) \leftarrow\)

### C(P)

- \(\text{colored}(V,r) \leftarrow \neg \text{colored}(V,b), \neg \text{colored}(V,g), \text{vertex}(V)\)
- \(\text{colored}(V,b) \leftarrow \neg \text{colored}(V,r), \neg \text{colored}(V,g), \text{vertex}(V)\)
- \(\text{colored}(V,g) \leftarrow \neg \text{colored}(V,r), \neg \text{colored}(V,b), \text{vertex}(V)\)

- \(\text{colored}(V,g) \leftarrow \neg \text{colored}(V,r), \neg \text{colored}(V,b), \text{vertex}(V)\)
- \(\leftarrow \text{edge}(V,U), \text{colored}(V,C), \text{colored}(U,C), \text{color}(C)\)

### Answer set

\[\{\text{colored}(1,r), \text{colored}(2,b), \text{colored}(3,g), \ldots\}\]

**Aside:** The answer sets will also contain extraneous information such as \(\text{vertex}(1)\), etc.
**n-colorability of graphs with** $n = 3$

<table>
<thead>
<tr>
<th>C(I)</th>
<th>vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(3) ← edge(3,1) ←</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(P)</td>
<td>color(r) ← color(b) ← color(g) ←</td>
</tr>
<tr>
<td></td>
<td>colored(V,C) ← not othercolor(V,C), vertex(V), color(C)</td>
</tr>
<tr>
<td></td>
<td>othercolor(V,C) ← colored(V,C'), C≠C', vertex(V), color(C), color(C')</td>
</tr>
<tr>
<td></td>
<td>edge(V,U), colored(V,C), colored(U,C), color(C)</td>
</tr>
</tbody>
</table>

**Answer set** \{ colored(1,r), colored(2,b), colored(3,g), ... \}

**Mnemonically, hasothercolour may be better than othercolour.**
$n$-colorability of graphs with $n = 3$

C(I)  
vertex(1). vertex(2). vertex(3).
edge(1,2). edge(2,3). edge(3,1).

C(P)  
color(r). color(b). color(g).
colored(V,C) :- not othercolor(V,C),
vertex(V),color(C).
othercolor(V,C) :- colored(V,C1), C != C1,
vertex(V),color(C),color(C1).
:- edge(V,U),color(C),
colored(V,C),colored(U,C).
Running the program

> lparse 3color.lp | smodels 0

smodels version 2.25. Reading...done
Answer: 1
Stable Model: colored(3,g) othercolor(2,g) othercolor(1,g) othercolor(3,b) colored(2,b) othercolor(1,b) othercolor(3,r) othercolor(2,r) colored(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
And the rest!

Answer: 2
Stable Model: colored(3,g) othercolor(2,g) othercolor(1,g) othercolor(3,b) othercolor(2,b) colored(1,b) othercolor(3,r) colored(2,r) othercolor(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
Answer: 3
Stable Model: othercolor(3,g) colored(2,g) othercolor(1,g) colored(3,b) othercolor(2,b) othercolor(1,b) othercolor(3,r) othercolor(2,r) colored(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
Answer: 4
Stable Model: othercolor(3,g) othercolor(2,g) colored(1,g) colored(3,b) othercolor(2,b) othercolor(1,b) othercolor(3,r) colored(2,r) othercolor(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
Answer: 5
Stable Model: othercolor(3,g) colored(2,g) othercolor(1,g) othercolor(3,b) othercolor(2,b) colored(1,b) colored(3,r) othercolor(2,r) othercolor(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
Answer: 6
Stable Model: othercolor(3,g) othercolor(2,g) colored(1,g) othercolor(3,b) colored(2,b) othercolor(1,b) colored(3,r) othercolor(2,r) othercolor(1,r) color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
False
Basic Methodology

Generate and Test (or: Guess and Check) approach:

**Generator:** Generate potential candidates answer sets
  - Typically using non-deterministic constructs

**Tester:** Eliminate non-valid candidates
  - Typically via integrity constraints

**As a slogan:**

Logic program = Data + Generator + Tester
Basic Methodology: Graph Colourability

Recall we had the description:

**Problem instance**

A graph \((V, E)\).

**Problem class**

Assign each vertex in \(V\) one of \(n\) colors such that no two vertices in \(V\) connected by an edge in \(E\) have the same color.

Note the structure of the problem class:

- **Generate:** Assign each vertex in \(V\) one of \(n\) colors . . .
- **Test:** . . . such that no two vertices in \(V\) connected by an edge in \(E\) have the same color.
Satisfiability

Problem instance

A propositional formula $\phi$.

Problem class

Is there an assignment of propositional variables to $true$ and $false$ such that a given formula $\phi$ is true?
Consider the formula \((a \lor \neg b) \land (\neg a \lor b)\).
Consider the formula \((a \lor \lnot b) \land (\lnot a \lor b)\).

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Answer set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Satisfiability
Consider the formula \((a \lor \neg b) \land (\neg a \lor b)\).

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Answer set</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>not a’</td>
<td></td>
</tr>
<tr>
<td>a’</td>
<td>not a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>not b’</td>
<td></td>
</tr>
<tr>
<td>b’</td>
<td>not b</td>
<td></td>
</tr>
<tr>
<td>A_1</td>
<td></td>
<td>{a, b}</td>
</tr>
<tr>
<td>A_2</td>
<td></td>
<td>{a’, b’}</td>
</tr>
</tbody>
</table>
Consider the formula \((a \lor \neg b) \land (\neg a \lor b)\).

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Answer set</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ← not (a')</td>
<td>← not (a, b)</td>
<td>(A_1 = {a, b})</td>
</tr>
<tr>
<td>(a') ← not (a)</td>
<td>← (a, \text{not } b)</td>
<td>(A_2 = {a', b'})</td>
</tr>
<tr>
<td>(b) ← not (b')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b') ← not (b)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the formula \((a \lor \neg b) \land (\neg a \lor b)\).

**Generator**

- \(a \leftarrow \text{not } a'\)
- \(a' \leftarrow \text{not } a\)
- \(b \leftarrow \text{not } b'\)
- \(b' \leftarrow \text{not } b\)

**Tester**

- \(\leftarrow \text{not } a, b\)
- \(\leftarrow a, \text{not } b\)

**Answer set**

- \(A_1 = \{a, b\}\)
- \(A_2 = \{a', b'\}\)
A solution to $n = 4$:

```
  Q
  
  Q
  Q
  Q
```
n-Queens in ASP

- \( q(X, Y) \) gives the legal position of a queen
- \( \neg q(X, Y) \) is an independent auxiliary atom (like \( q'(X, Y) \))
n-Queens in ASP

- \( q(X, Y) \) gives the legal position of a queen
- \( \neg q(X, Y) \) is an independent auxiliary atom (like \( q'(X, Y) \))

\[
q(X, Y) \leftarrow \text{not} \neg q(X, Y)
\]
\[
\neg q(X, Y) \leftarrow \text{not} \quad q(X, Y)
\]
n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom (like $q'(X, Y)$)

\[
q(X, Y) \leftarrow \text{not} \neg q(X, Y)
\]
\[
\neg q(X, Y) \leftarrow \text{not} \; q(X, Y)
\]
\[
\leftarrow q(X, Y), q(X', Y), X \neq X'
\]
\[
\leftarrow q(X, Y), q(X, Y'), Y \neq Y'
\]
\[
\leftarrow q(X, Y), q(X', Y'), |X - X'| = |Y - Y'|,
\]
\[
X \neq X', Y \neq Y'
\]
n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom (like $q'(X, Y)$)

\[
q(X, Y) \leftarrow \text{not} \neg q(X, Y)
\]
\[
\neg q(X, Y) \leftarrow \text{not} \ q(X, Y)
\]
\[
\leftarrow q(X, Y), q(X', Y), X \neq X'
\]
\[
\leftarrow q(X, Y), q(X, Y'), Y \neq Y'
\]
\[
\leftarrow q(X, Y), q(X', Y'), |X - X'| = |Y - Y'|,
\]
\[
\quad X \neq X', Y \neq Y'
\]
\[
\leftarrow \text{not} \ hasq(X)
\]
\[
hasq(X) \leftarrow q(X, Y)
\]
n-Queens (in the smodels language)

d(1..queens).

q(X,Y) :- d(X), d(Y), not negq(X,Y).
negq(X,Y) :- d(X), d(Y), not q(X,Y).

:- d(X), d(Y), d(X1), q(X,Y), q(X1,Y), X1 != X.
:- d(X), d(Y), d(Y1), q(X,Y), q(X,Y1), Y1 != Y.
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1),
    X != X1, Y != Y1, abs(X - X1) == abs(Y - Y1).

:- d(X), not hasq(X).
hasq(X) :- d(X), d(Y), q(X,Y).
Hamiltonian Path

Problem instance
A directed graph \((V, E)\) and a starting vertex \(v \in V\).

Problem class
Find a path in \((V, E)\) starting at \(v\) and visiting all other vertices in \(V\) exactly once.

- Predicates: \(\text{vertex}/1\), \(\text{arc}/2\), \(\text{start}/1\)
Strategy

- Generate candidate paths
- Eliminate candidates having vertices visited more than once
- Eliminate candidates having vertices never visited
Generator (for candidate paths)

\[
\begin{align*}
\text{inPath}(X, Y) & \leftarrow \text{arc}(X, Y), \ not \ \text{outPath}(X, Y) \\
\text{outPath}(X, Y) & \leftarrow \text{arc}(X, Y), \ not \ \text{inPath}(X, Y)
\end{align*}
\]
Tester (to eliminate invalid paths)

- Eliminate candidates having vertices visited more than once
  \[ \leftarrow \text{inPath}(X, Y), \text{inPath}(X, Z), \ Y \neq Z \]
  \[ \leftarrow \text{inPath}(X, Y), \text{inPath}(Z, Y), \ X \neq Z \]
Tester (to eliminate invalid paths)

- Eliminate candidates having vertices visited more than once
  \[\text{inPath}(X, Y), \text{inPath}(X, Z), Y \neq Z\]
  \[\text{inPath}(X, Y), \text{inPath}(Z, Y), X \neq Z\]

- Eliminate candidates having vertices never visited
  \[\text{reached}(X) \leftarrow \text{start}(X)\]
  \[\text{reached}(X) \leftarrow \text{reached}(Y), \text{inPath}(Y, X)\]
  \[\text{vertex}(X), \text{not reached}(X)\]
Planning in the Blocks World

Initial situation

1
2

3
4

5
6

Goal situation

3
2

6
5

1
4
const grippers=2.
const lasttime=3.

block(1..6).

% DEFINE
on(1,2,0). % block 1 is on 2 in time 0
on(2,table,0).
on(3,4,0).
on(4,table,0).
on(5,6,0).
on(6,table,0).
Goal Situation

% TEST
:- not on(3,2,lasttime).
:- not on(2,1,lasttime).
:- not on(1,table,lasttime).
:- not on(6,5,lasttime).
:- not on(5,4,lasttime).
:- not on(4,table,lasttime).

I.e. exclude answer sets where the goal conditions do not hold.
time(0..lasttime).

% Possible locations are on top of blocks or on the table.

location(B) :- block(B).
location(table).

% GENERATE (using a choice rule)
\{ move(B,L,T) : block(B) : location(L) \} grippers :-
  time(T), T<lasttime.

\[\]
% I.e. For a legal time point, can make as many moves as there
are grippers.
% effect of moving a block
on(B,L,T+1) :- move(B,L,T),
    block(B), location(L),
    time(T), T<lasttime.

% inertia
on(B,L,T+1) :- on(B,L,T), not neg_on(B,L,T+1),
    location(L), block(B),
    time(T), T<lasttime.

% uniqueness of location
neg_on(B,L1,T) :- on(B,L,T), L!=L1,
    block(B), location(L), location(L1),
    time(T).
% neg_on is the negation of on
:- on(B,L,T), neg_on(B,L,T),
   block(B), location(L), time(T).

% two blocks cannot be on top of the same block
:- on(B1,B,T), on(B2,B,T),
   block(B1), block(B2), time(T), B1!=B2.

% a block can’t be moved unless it is clear
:- move(B,L,T), on(B1,B,T),
   block(B), block(B1), location(L), time(T), T<lasttime.

% a block can’t be moved onto a block that is being moved also
:- move(B,B1,T), move(B1,L,T),
   block(B), block(B1), location(L), time(T), T<lasttime.
The Plan

> lparse blocks.lp | smodels

smodels version 2.25. Reading...done
Answer: 1
Stable Model: move(1,table,0) move(3,table,0)
    move(2,1,1) move(5,4,1)
    move(3,2,2) move(6,5,2)
Duration: 0.050
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 507
Number of rules: 3026
Number of picked atoms: 24
Number of forced atoms: 13
Number of truth assignments: 944
Size of searchspace (removed): 0 (0)