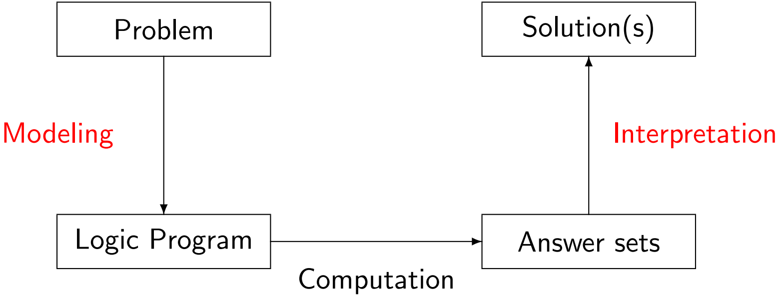
Modelling Problems in ASP

Modeling and Interpreting

Recall:



Problem $\longmapsto$ Logic Program

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)	vertex(1) $\leftarrow$		edge(1,2) $\leftarrow$	
	vertex(2) $\leftarrow$		edge(2,3) $\leftarrow$	
	vertex(3) $\leftarrow$		edge(3,1) $\leftarrow$	
C(P)	colored(V,r)	$\leftarrow$	not colored(V,b), not colored(V,g), vertex(V)	
	colored(V,b)	$\leftarrow$	not colored(V,r), not colored(V,g), vertex(V)	
	colored(V,g)	$\leftarrow$	not colored(V,r), not colored(V,b), vertex(V)	
		$\leftarrow$	edge(V,U), colored(V,C), colored(U,C), color(C)	
Answer set	{ colored(1,r), colored(2,b), colored(3,g), ...}			

Aside: The answer sets will also contain extraneous information such as vertex(1), etc.

n-colorability of graphs with $n = 3$

C(I)	vertex(1) ←	edge(1,2) ←	
	vertex(2) ←	edge(2,3) ←	
	vertex(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,C'), C≠C', vertex(V), color(C), color(C')
		←	edge(V,U), colored(V,C), colored(U,C), color(C)
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:

$$\text{Logic program} \quad = \quad \text{Data} + \text{Generator} + \text{Tester}$$

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 ϕ.

Problem class

Is there an assignment of propositional variables to *true* and *false* such that a given formula ϕ is true?

Satisfiability

Consider the formula $(a \vee \neg b) \wedge (\neg a \vee b)$.

Satisfiability

Consider the formula $(a \vee \neg b) \wedge (\neg a \vee b)$.

Generator Tester Answer set

Satisfiability

Consider the formula $(a \vee \neg b) \wedge (\neg a \vee b)$.

Generator	Tester	Answer set
$a \leftarrow not\ a'$		
$a' \leftarrow not\ a$		
$b \leftarrow not\ b'$		
$b' \leftarrow not\ b$		

Satisfiability

Consider the formula $(a \vee \neg b) \wedge (\neg a \vee b)$.

Generator	Tester	Answer set
$a \leftarrow not\ a'$	$\leftarrow not\ a, b$	
$a' \leftarrow not\ a$	$\leftarrow a, not\ b$	
$b \leftarrow not\ b'$		
$b' \leftarrow not\ b$		

Satisfiability

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

Generator

$$\begin{array}{lcl} a & \leftarrow & \text{not } a' \\ a' & \leftarrow & \text{not } a \\ b & \leftarrow & \text{not } b' \\ b' & \leftarrow & \text{not } b \end{array}$$

Tester

$$\begin{array}{ll} \leftarrow & \text{not } a,\ b \\ \leftarrow & a,\ \text{not } b \end{array}$$

Answer set

$$\begin{array}{lcl} A_1 & = & \{a,b\} \\ A_2 & = & \{a',b'\} \end{array}$$

n-Queens Problem

A solution to $n = 4$:

	Q		
			Q
Q			
		Q	

n-Queens in ASP

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

n-Queens in ASP

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

$$\begin{aligned} q(X,Y) &\leftarrow not\ negq(X,Y) \\ negq(X,Y) &\leftarrow not\ \ q(X,Y) \end{aligned}$$

n-Queens in ASP

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

$$
\begin{aligned}
q(X, Y) &\leftarrow not\ negq(X, Y) \\
negq(X, Y) &\leftarrow not\ \ q(X, Y) \\
\\
&\leftarrow q(X, Y), q(X', Y), X \neq X' \\
&\leftarrow q(X, Y), q(X, Y'), Y \neq Y' \\
&\leftarrow q(X, Y), q(X', Y'), |X - X'| = |Y - Y'|, \\
&\qquad X \neq X', Y \neq Y'
\end{aligned}
$$

n-Queens in ASP

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

$$
\begin{array}{rcl}
q(X, Y) & \leftarrow & not\ negq(X, Y) \\
negq(X, Y) & \leftarrow & not\ \ q(X, Y)
\end{array}
$$

$$
\begin{array}{rcl}
 & \leftarrow & q(X, Y), q(X', Y), X \neq X' \\
 & \leftarrow & q(X, Y), q(X, Y'), Y \neq Y' \\
 & \leftarrow & q(X, Y), q(X', Y'), |X - X'| = |Y - Y'|, \\
 & & X \neq X', Y \neq Y'
\end{array}
$$

$$
\begin{array}{rcl}
 & \leftarrow & not\ hasq(X) \\
hasq(X) & \leftarrow & q(X, Y)
\end{array}
$$

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: *vertex/1*, *arc/2*, *start/1*

Strategy

- Generate candidate paths
- Eliminate candidates having vertices visited more than once
- Eliminate candidates having vertices never visited

Generator (for candidate paths)

$$
\begin{aligned}
\mathit{inPath}(X, Y) &\leftarrow \mathit{arc}(X, Y),\ \mathit{not}\ \mathit{outPath}(X, Y) \\
\mathit{outPath}(X, Y) &\leftarrow \mathit{arc}(X, Y),\ \mathit{not}\ \mathit{inPath}(X, Y)
\end{aligned}
$$

Tester (to eliminate invalid paths)

- Eliminate candidates having vertices visited more than once

$$\leftarrow \quad inPath(X, Y),\ inPath(X, Z),\ Y \neq Z$$
$$\leftarrow \quad inPath(X, Y),\ inPath(Z, Y),\ X \neq Z$$

Tester (to eliminate invalid paths)

- Eliminate candidates having vertices visited more than once

$$\leftarrow \quad inPath(X,Y),\ inPath(X,Z),\ Y \neq Z$$
$$\leftarrow \quad inPath(X,Y),\ inPath(Z,Y),\ X \neq Z$$

- Eliminate candidates having vertices never visited

$$\begin{aligned} reached(X) &\leftarrow start(X) \\ reached(X) &\leftarrow reached(Y),\ inPath(Y,X) \\ &\leftarrow vertex(X),\ not\ reached(X) \end{aligned}$$

Classical Negation: Syntax

Normal logic programs

- In logic programs *not* (or $\sim$) denotes default negation.
- Default negation refers to the *absence of information*

Classical Negation: Syntax

Normal logic programs

- In logic programs *not* (or $\sim$) denotes default negation.
- Default negation refers to the *absence of information*

Generalization

- We allow classical negation for atoms (only!).
- "classical" negation stipulates the *presence of the negated information*

Classical Negation: Syntax

Normal logic programs

- In logic programs *not* (or $\sim$) denotes default negation.
- Default negation refers to the *absence of information*

Generalization

- We allow classical negation for atoms (only!).
- "classical" negation stipulates the *presence of the negated information*
- Given an alphabet $\mathcal{A}$ of atoms, let
  $$\overline{\mathcal{A}} = \{\neg A \mid A \in \mathcal{A}\} \qquad \text{(and so } \mathcal{A} \cap \overline{\mathcal{A}} = \emptyset\text{)}$$
- The atoms A and $\neg A$ are complementary.
  - ☞ $\neg A$ is the classical negation of A, and vice versa.

Syntax (ctd)

- Given set X, the difference between *not a* and $\neg a$ amounts to:

$$a \notin X \quad \text{versus} \quad \neg a \in X$$

Syntax (ctd)

- Given set X, the difference between *not* a and $\neg a$ amounts to:

$$a \notin X \quad \text{versus} \quad \neg a \in X$$

- Example:

$$a \leftarrow \ not\ b \qquad\qquad a \leftarrow \ \neg\ b$$

Syntax (ctd)

- Given set X, the difference between *not* a and $\neg a$ amounts to:

$$a \notin X \quad \text{versus} \quad \neg a \in X$$

- Example:

$$\begin{array}{ll} a \leftarrow \; not \; b & \qquad a \leftarrow \; \neg \; b \\ X = \{a\} & \qquad X = \emptyset \end{array}$$

Syntax (ctd)

- Given set X, the difference between *not a* and $\neg a$ amounts to:

$$a \notin X \quad \text{versus} \quad \neg a \in X$$

- Example:

$$a \leftarrow \ not\ b \qquad\qquad a \leftarrow \ \neg\ b$$
$$X = \{a\} \qquad\qquad X = \emptyset$$

- Again:
  - default negation refers to the absence of information
  - "classical" negation is the presence of the negated information

Semantics

- A set X of atoms is an answer set of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ if X is an answer set of $\Pi \cup \Pi'$ where

$$\Pi' = \{\leftarrow A, \neg A \mid A \in \mathcal{A}\}$$

☞ The text has a more general definition, which we won't bother with

- We've already seen "encoded" classical negation used in earlier examples
  - E.g.
    - in satisfiability: a vs. a', and
    - in n-queens: q(X,Y) vs. negq(X,Y)
  - Here the definition is given by adding, for every $A \in \mathcal{A}$:

$$A \leftarrow \mathit{not}\, \neg A \qquad \text{and} \qquad \neg A \leftarrow \mathit{not}\, A$$

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ \ train\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$

To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$

To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ \ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$

To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ \ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$

- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \ \neg cross \leftarrow\}$

To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ \ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
  - No answer set
- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ \ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
  - No answer set
- $\Pi_5 = \{cross \leftarrow \neg train, not\ \neg cross, \quad \neg train \leftarrow, \quad \neg cross \leftarrow\}$
  - Answer set: $\{\neg cross, \neg train\}$

Planning

- The following is included as an example, but isn't covered in class.
- It uses an advanced construct, called a *choice* construct, which we won't be going over
- The statement:

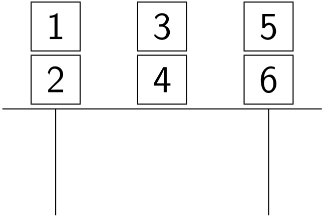
```
{ move(B,L,T) : block(B) : location(L) } grippers :-
        time(T), T<lasttime.
```

  says that for a time point T, one can make as many moves as there are grippers.
- More precisely:
  - "grippers" is a constant, here 2
  - For a given value of T,
    `{ move(B,L,T) : block(B) : location(L) }`
    stands for 0, 1, or 2 distinct instances of move(B,L,T)

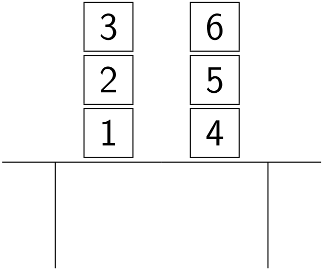
Planning

in the Blocks World

Initial situation



Goal situation



Initial Situation

```
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.

Planning in the Blocks World I

GENERATE

```
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.
```

- The above uses is *choice* construct, which we won't cover
- Idea: for a time point T, can make as many moves as there are grippers.

Planning in the Blocks World II

DEFINE

```
% 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).
```

Planning in the Blocks World III

TEST

```
% 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)
```