Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

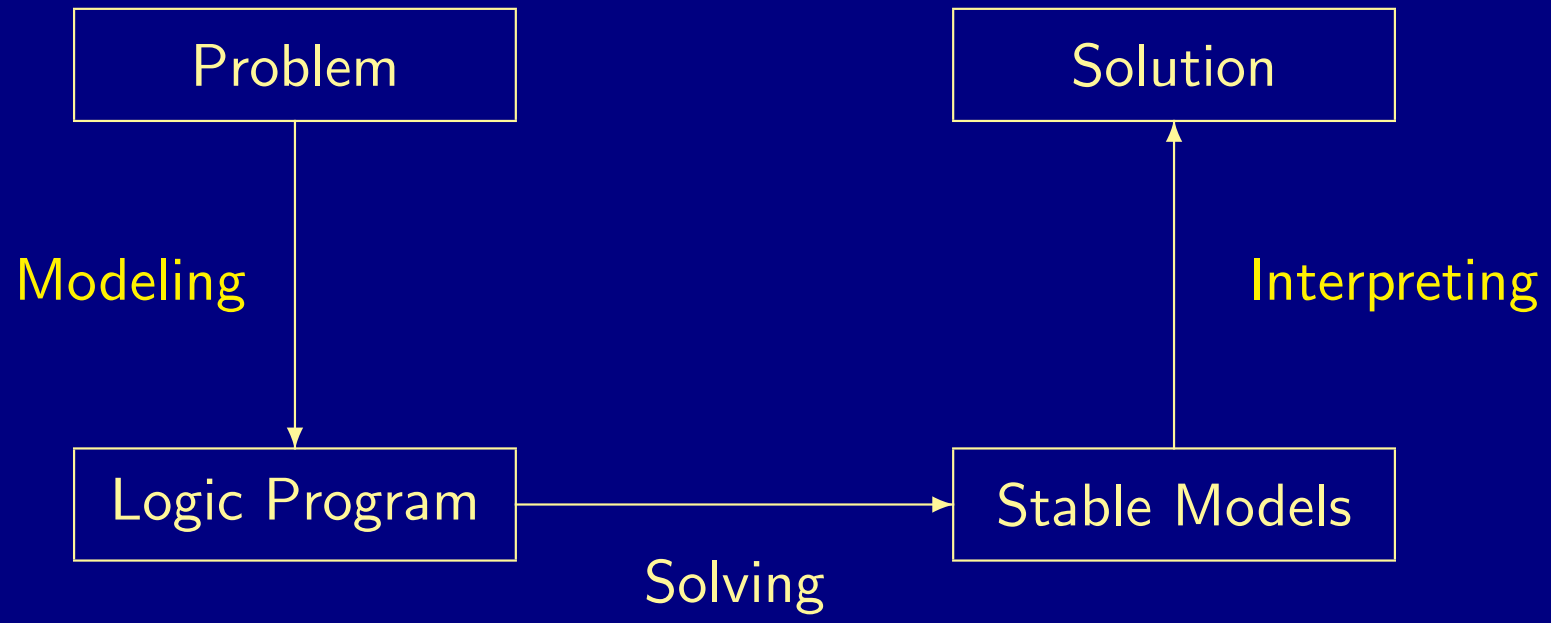
Potassco

Basic Modeling: Overview

1. Elaboration tolerance
2. ASP solving process
3. Methodology
4. Case studies

Potassco

Modeling and Interpreting



Potassco

Outline

Potassco

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $P_{\mathbf{I}}$,
  - **C** is represented as a set of rules $P_{\mathbf{C}}$, and
  - $P_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Potassco

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

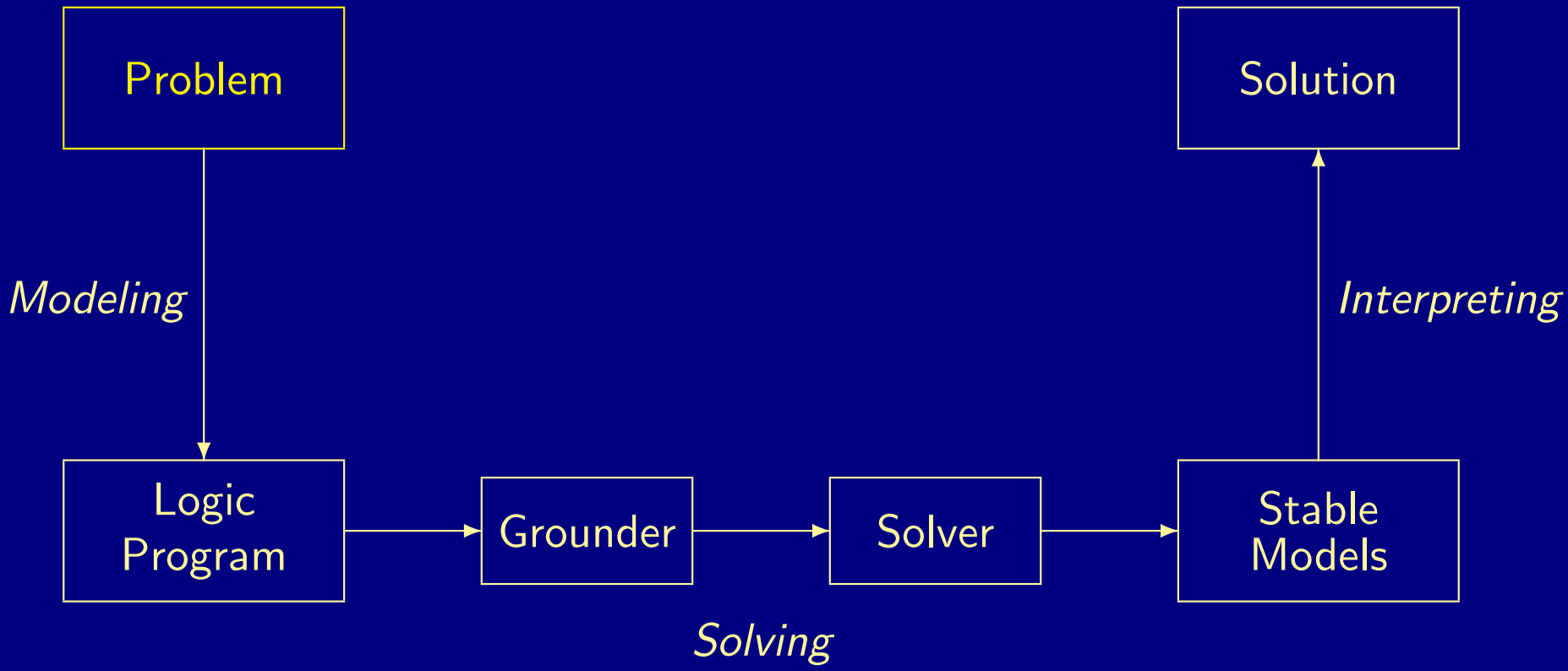
  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $P_{\mathbf{I}}$,
  - **C** is represented as a set of rules $P_{\mathbf{C}}$, and
  - $P_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Potassco

Outline

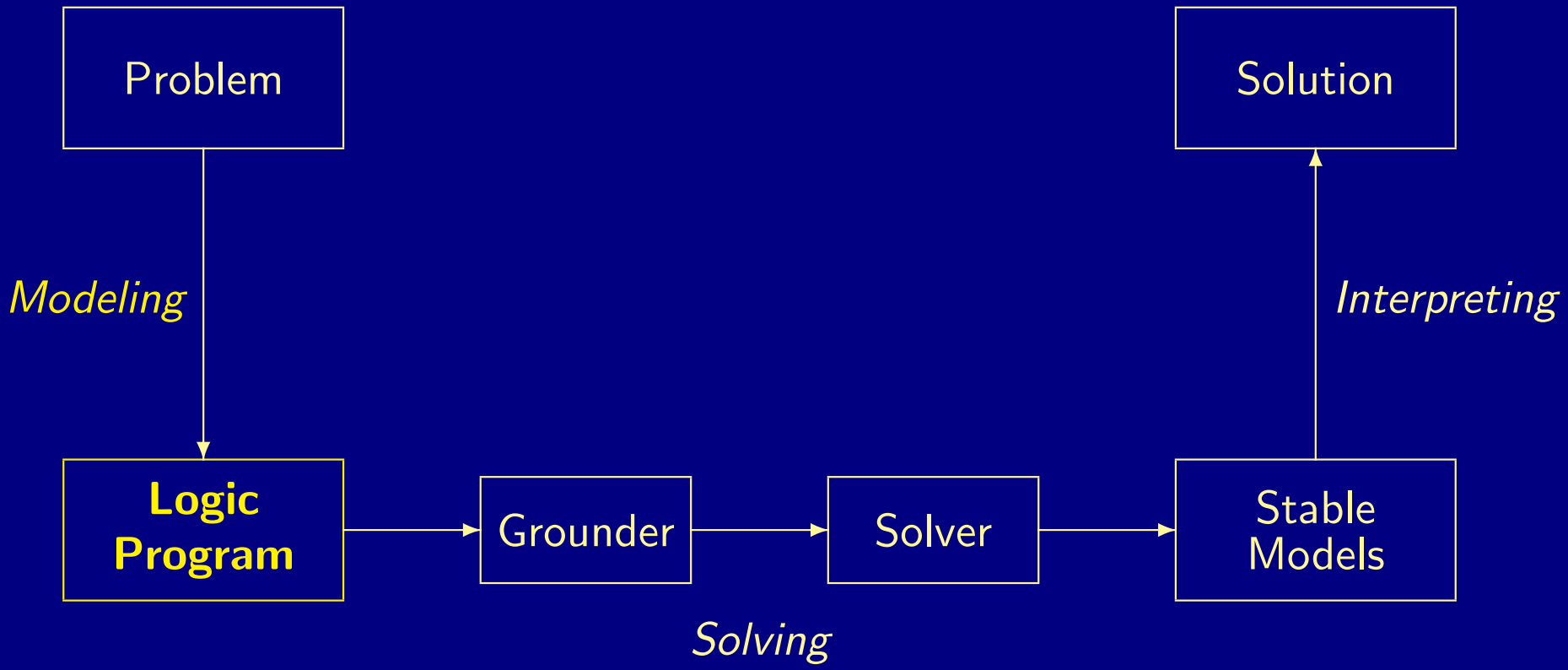
Potassco

ASP solving process



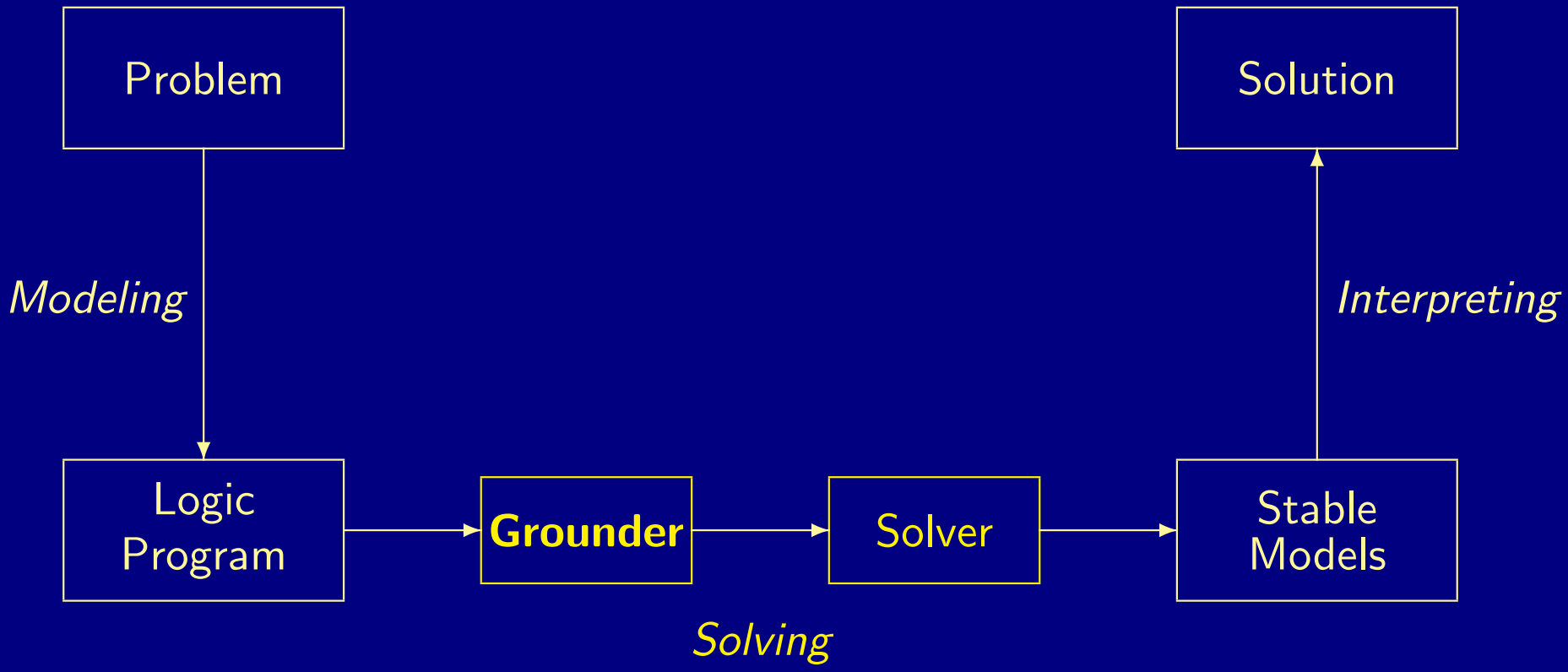
Potassco

ASP solving process



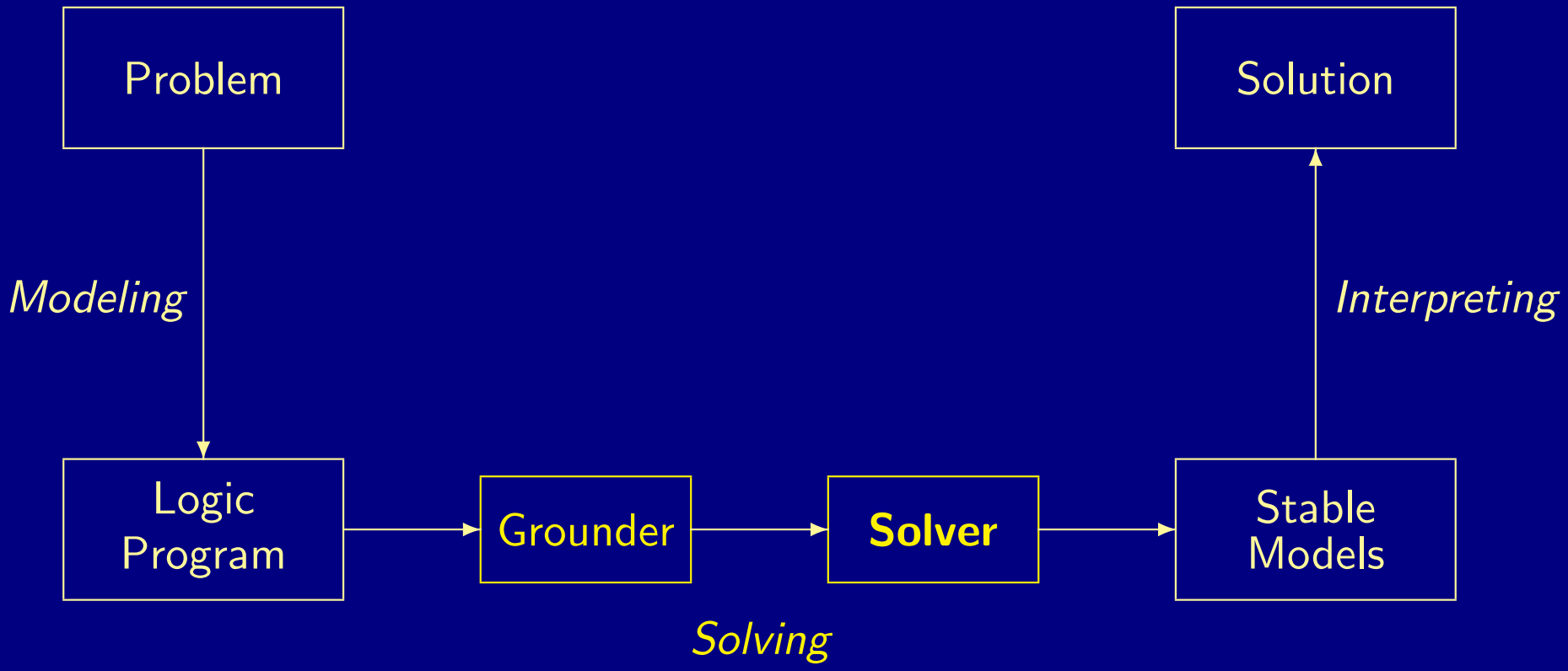
Potassco

ASP solving process



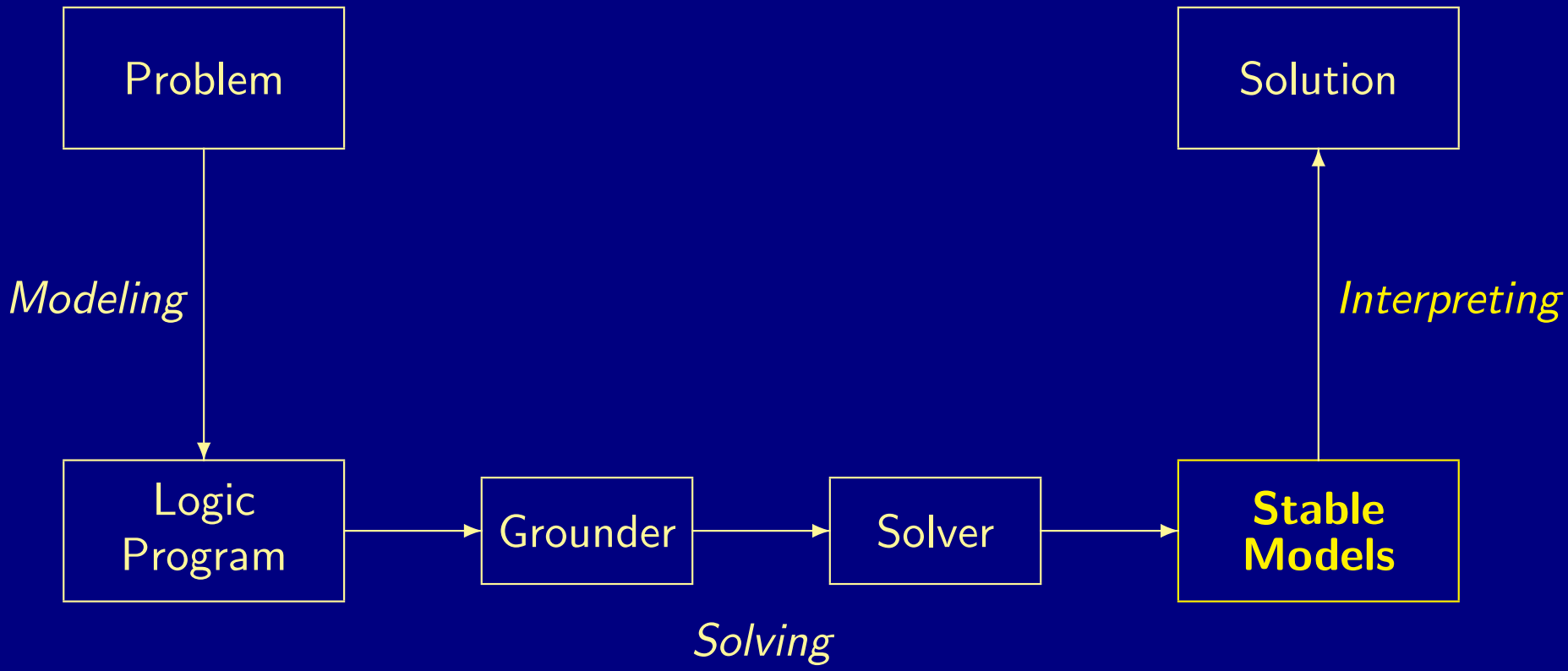
Potassco

ASP solving process



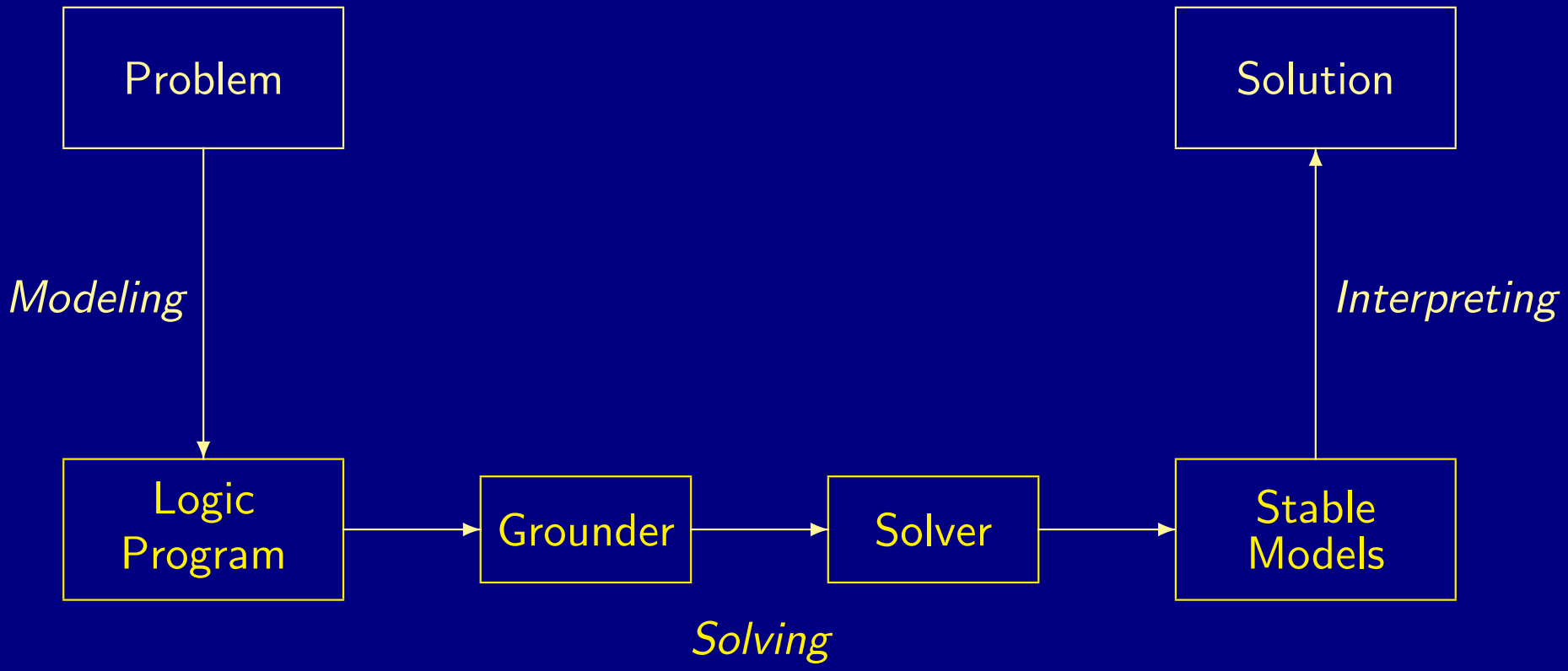
Potassco

ASP solving process



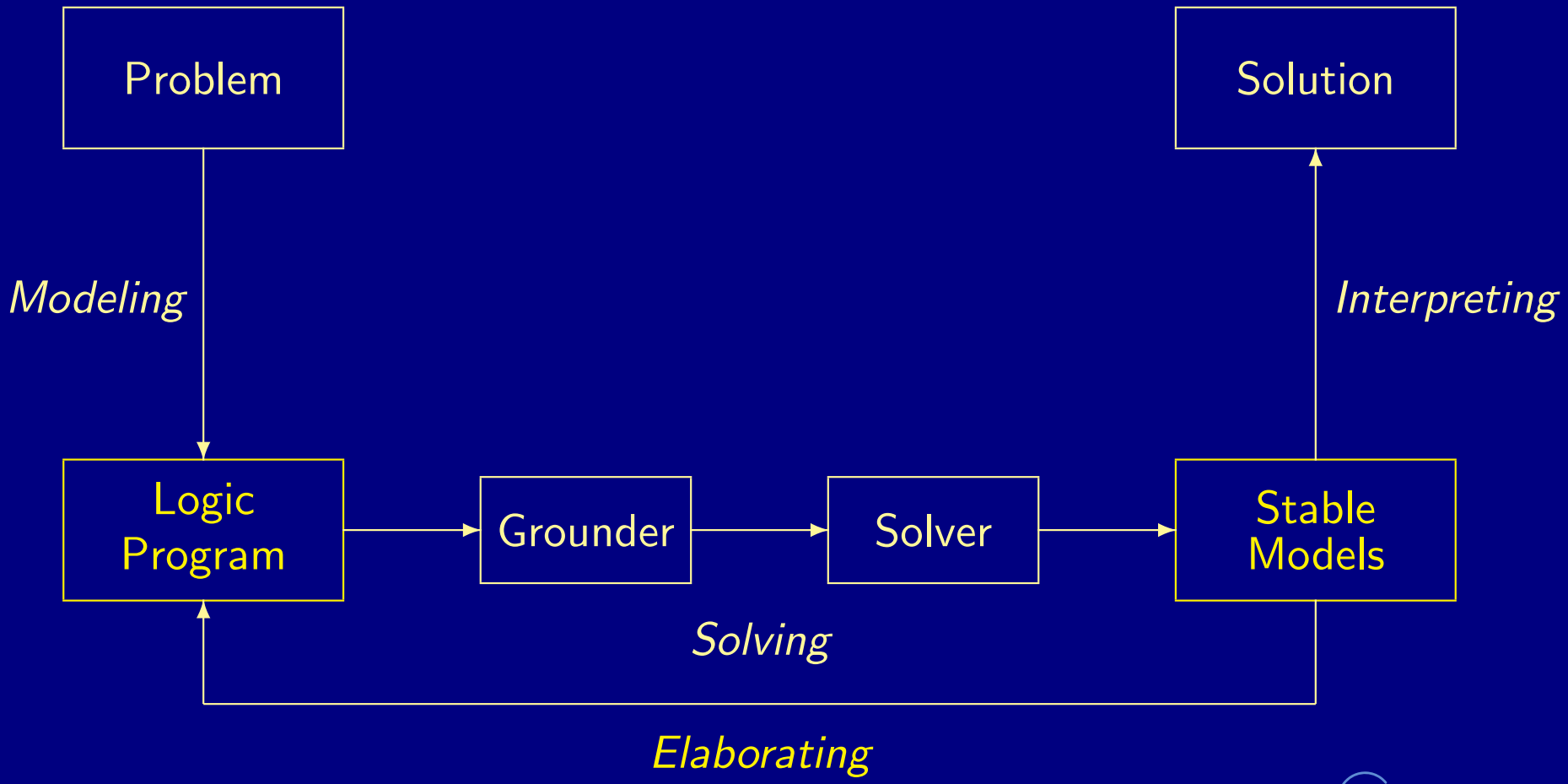
Potassco

ASP solving process



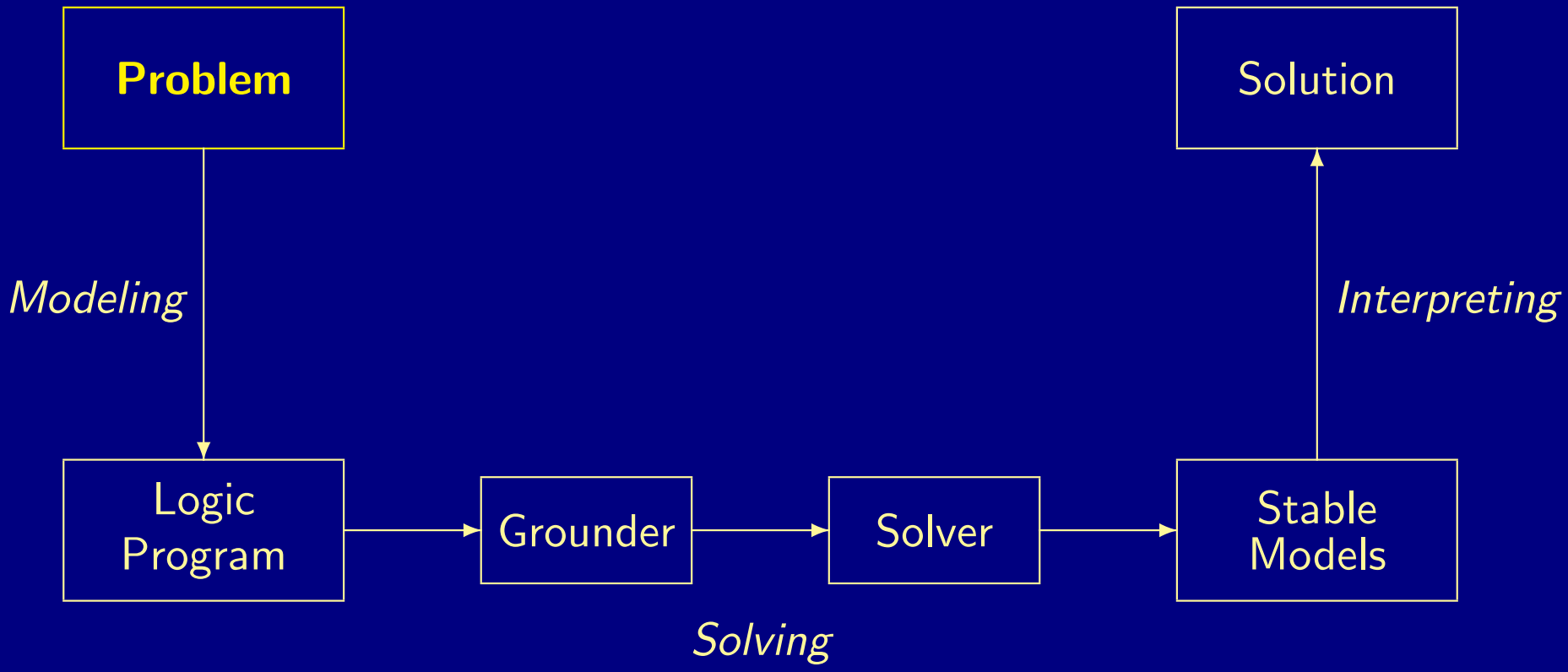
Potassco

ASP solving process



Potassco

A case-study: Graph coloring



Graph coloring

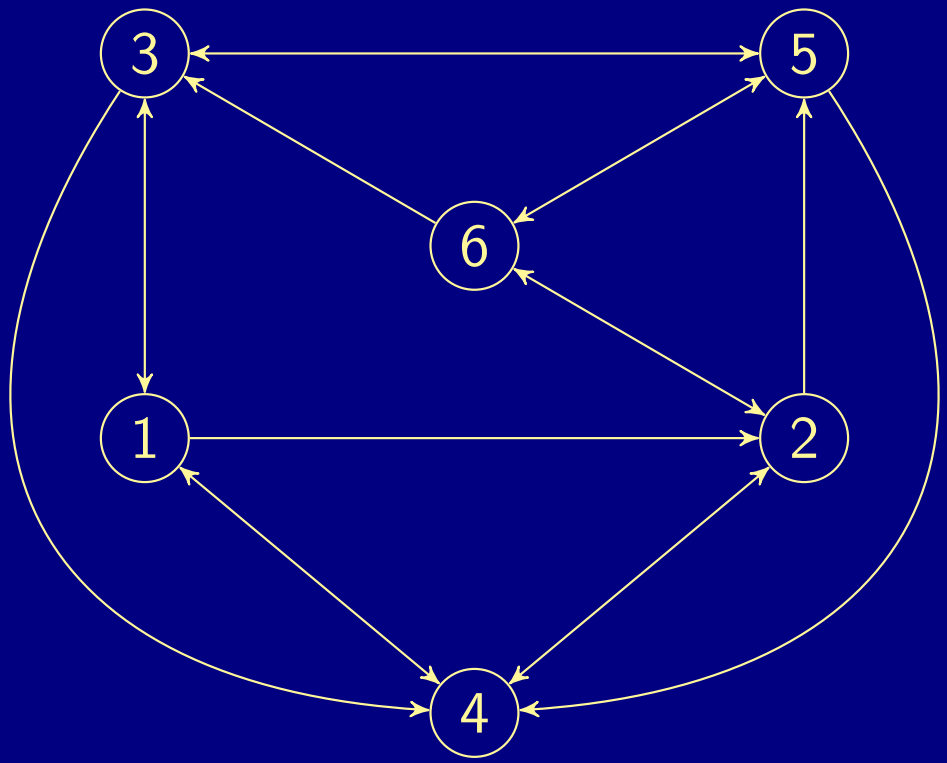
- Problem instance A graph consisting of nodes and edges

Graph coloring

- Problem instance A graph consisting of nodes and edges

Graph coloring

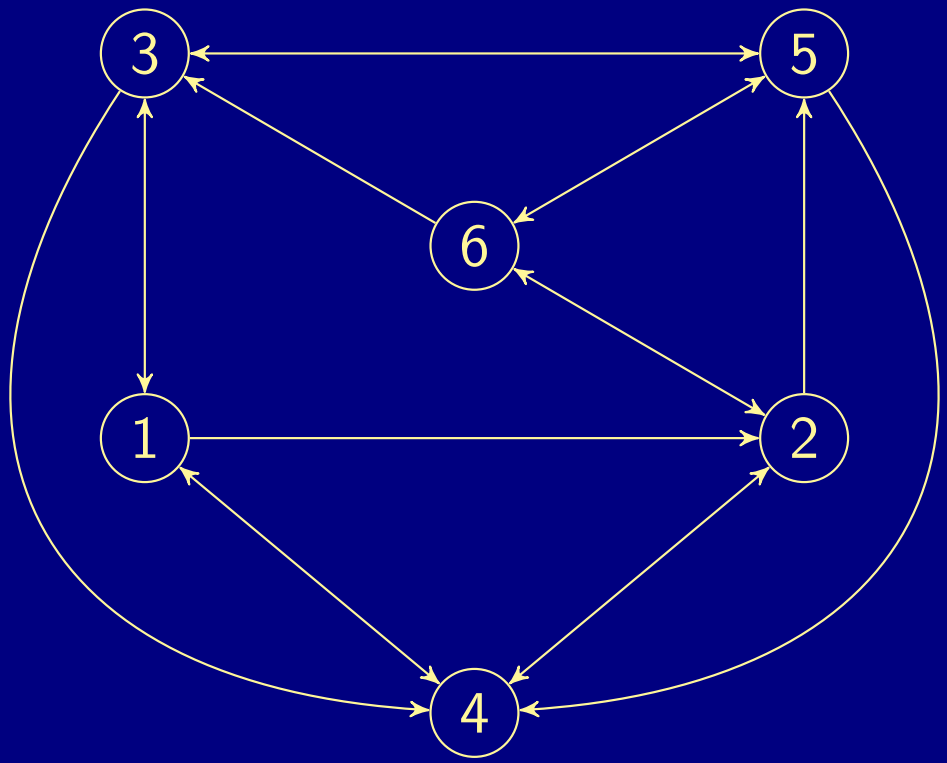
- Problem instance A graph consisting of nodes and edges



Potassco

Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`



Potassco

Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`

Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`
- Problem class Assign each node one color such that no two nodes connected by an edge have the same color

Graph coloring

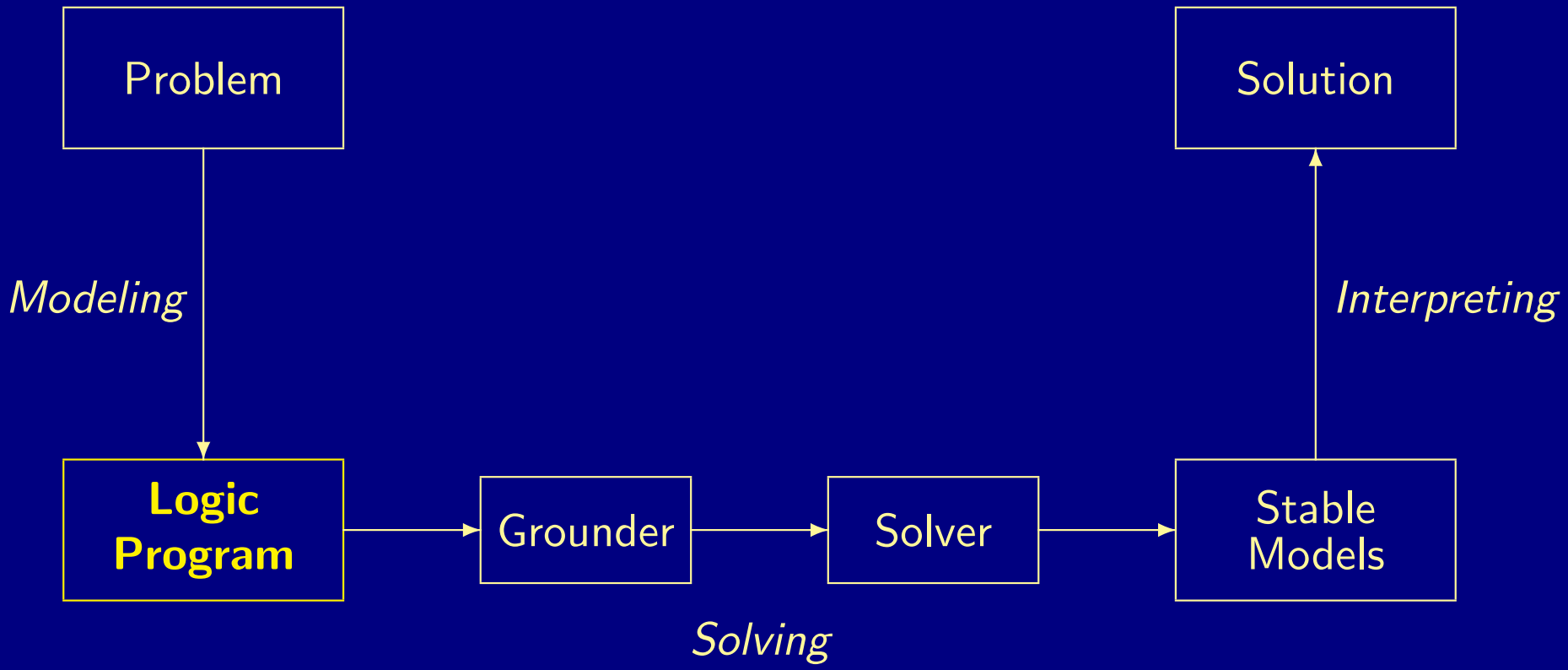
- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`
- Problem class Assign each node one color such that no two nodes connected by an edge have the same color

  In other words,

  1. Each node has one color
  2. Two connected nodes must not have the same color

Potassco

ASP solving process



Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

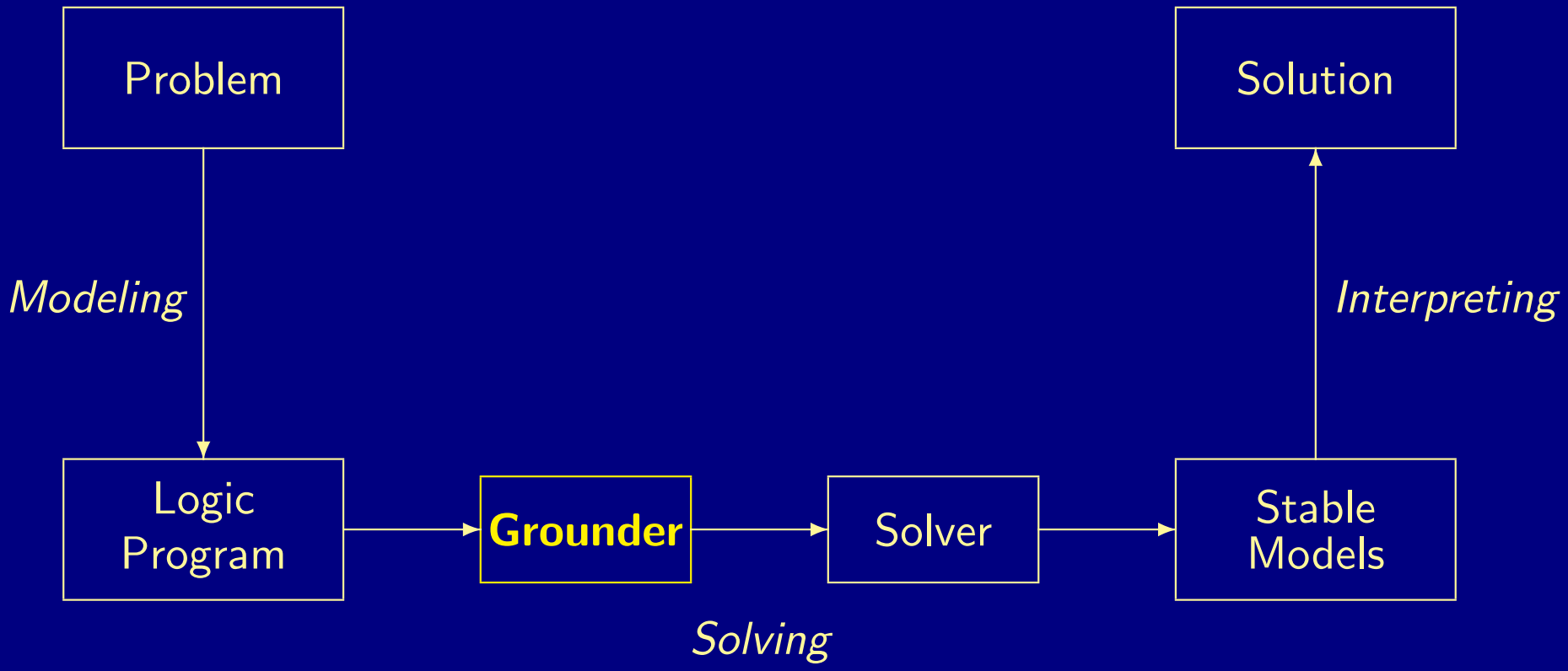
graph.lp

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

color.lp

Potassco

ASP solving process



Potassco

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp
```

```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).            edge(5,6). edge(6,5).

color(r). color(b). color(g).

{assign(1,r),assign(1,b),assign(1,g)} = 1. {assign(4,r),assign(4,b),assign(4,g)} = 1.
{assign(2,r),assign(2,b),assign(2,g)} = 1. {assign(5,r),assign(5,b),assign(5,g)} = 1.
{assign(3,r),assign(3,b),assign(3,g)} = 1. {assign(6,r),assign(6,b),assign(6,g)} = 1.

:- assign(1,r),assign(2,r). :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b). :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g). :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r). :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b). :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g). :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r). :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b). :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g). :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Potassco

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp
```

```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).            edge(5,6). edge(6,5).

color(r). color(b). color(g).

{assign(1,r),assign(1,b),assign(1,g)} = 1. {assign(4,r),assign(4,b),assign(4,g)} = 1.
{assign(2,r),assign(2,b),assign(2,g)} = 1. {assign(5,r),assign(5,b),assign(5,g)} = 1.
{assign(3,r),assign(3,b),assign(3,g)} = 1. {assign(6,r),assign(6,b),assign(6,g)} = 1.

:- assign(1,r),assign(2,r). :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b). :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g). :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r). :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b). :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g). :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r). :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b). :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g). :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Potassco

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp
```

```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).            edge(5,6). edge(6,5).

color(r). color(b). color(g).

{assign(1,r),assign(1,b),assign(1,g)} = 1. {assign(4,r),assign(4,b),assign(4,g)} = 1.
{assign(2,r),assign(2,b),assign(2,g)} = 1. {assign(5,r),assign(5,b),assign(5,g)} = 1.
{assign(3,r),assign(3,b),assign(3,g)} = 1. {assign(6,r),assign(6,b),assign(6,g)} = 1.

:- assign(1,r),assign(2,r). :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b). :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g). :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r). :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b). :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g). :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r). :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b). :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g). :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Potassco

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp
```

```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).            edge(5,6). edge(6,5).

color(r). color(b). color(g).

{assign(1,r),assign(1,b),assign(1,g)} = 1. {assign(4,r),assign(4,b),assign(4,g)} = 1.
{assign(2,r),assign(2,b),assign(2,g)} = 1. {assign(5,r),assign(5,b),assign(5,g)} = 1.
{assign(3,r),assign(3,b),assign(3,g)} = 1. {assign(6,r),assign(6,b),assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Potassco

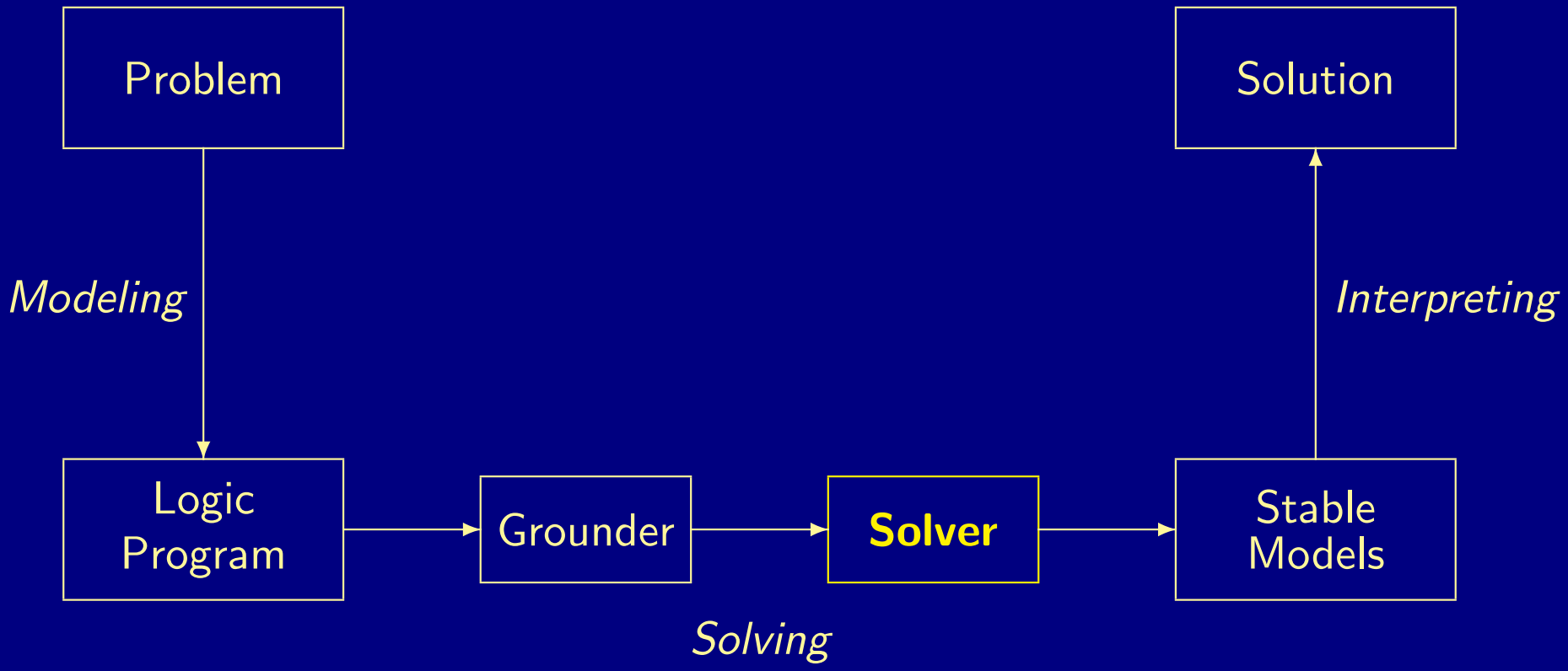
Graph coloring: Grounding

```
$ clingo --text graph.lp color.lp
```

```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).            edge(5,6). edge(6,5).

color(r). color(b). color(g).

{assign(1,r),assign(1,b),assign(1,g)} = 1. {assign(4,r),assign(4,b),assign(4,g)} = 1.
{assign(2,r),assign(2,b),assign(2,g)} = 1. {assign(5,r),assign(5,b),assign(5,g)} = 1.
{assign(3,r),assign(3,b),assign(3,g)} = 1. {assign(6,r),assign(6,b),assign(6,g)} = 1.

:- assign(1,r),assign(2,r). :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b). :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g). :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r). :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b). :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g). :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r). :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b). :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g). :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Potassco

ASP solving process



Potassco

Graph coloring: Solving

```
$ gringo graph.lp color.lp | clasp 0
clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models       : 6
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Potassco

Graph coloring: Solving

```
$ gringo graph.lp color.lp | clasp 0
```

```
clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models       : 6
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Potassco

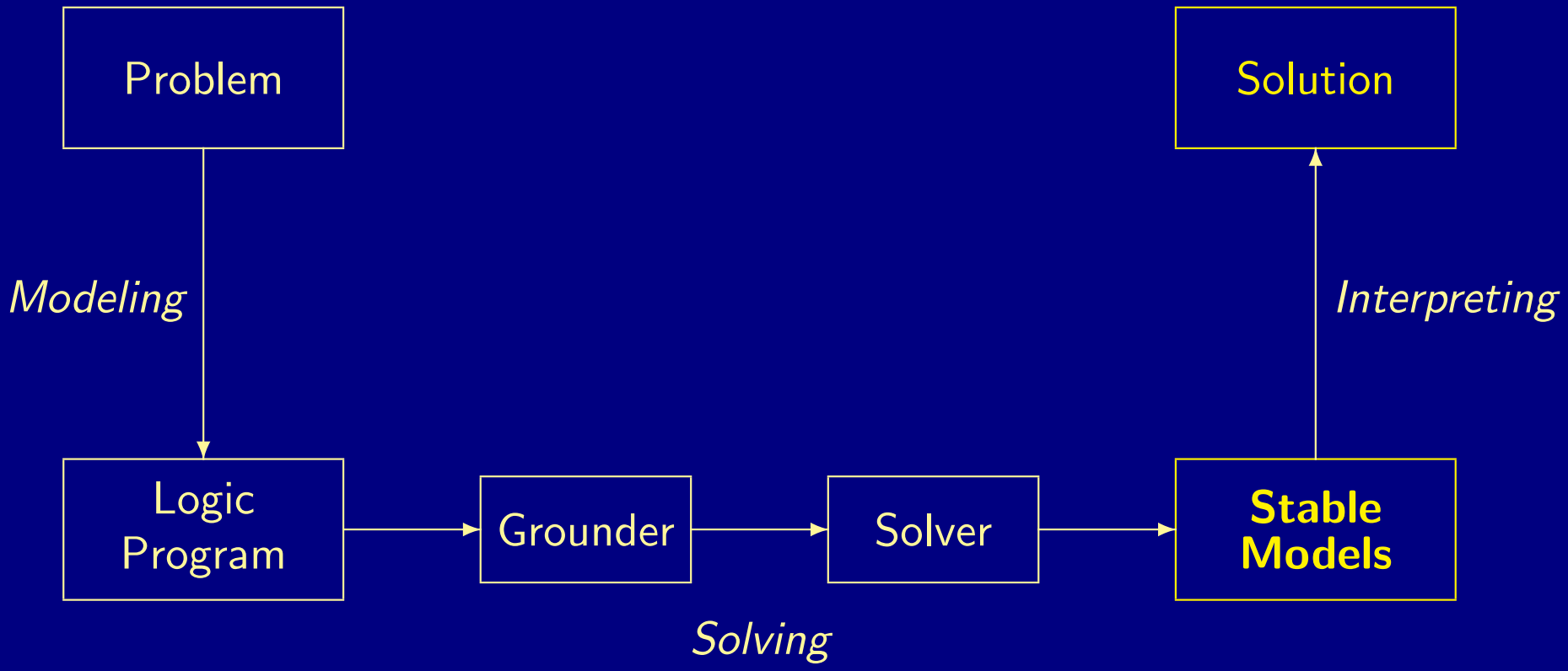
Graph coloring: Solving

```
$ clingo graph.lp color.lp 0
```

```
clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models       : 6
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Potassco

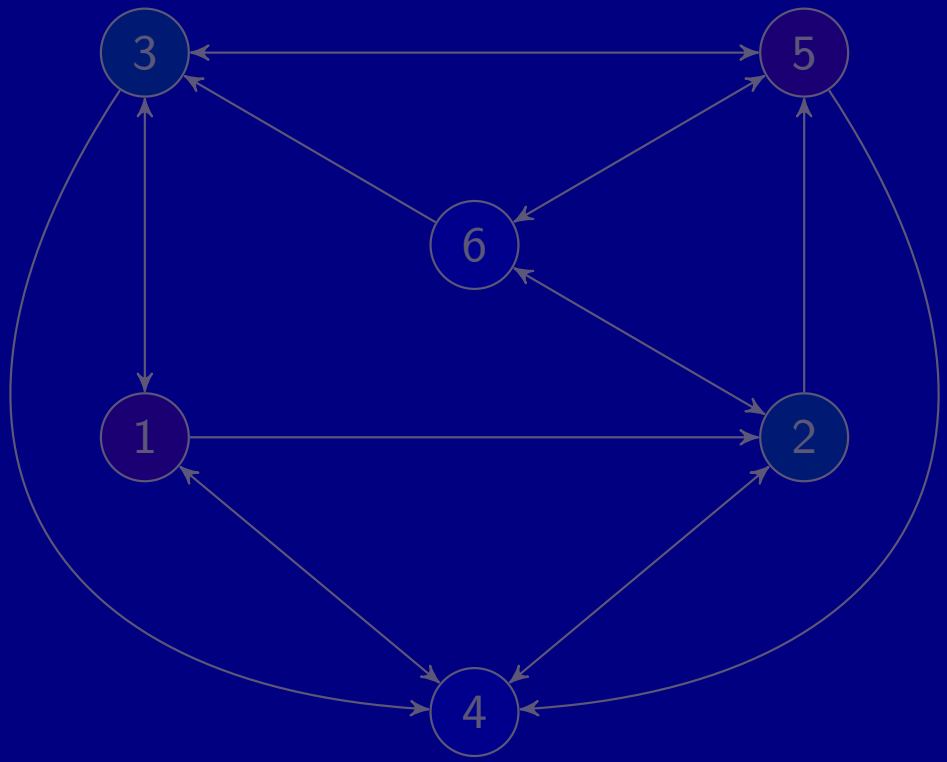
ASP solving process



Potassco

A coloring

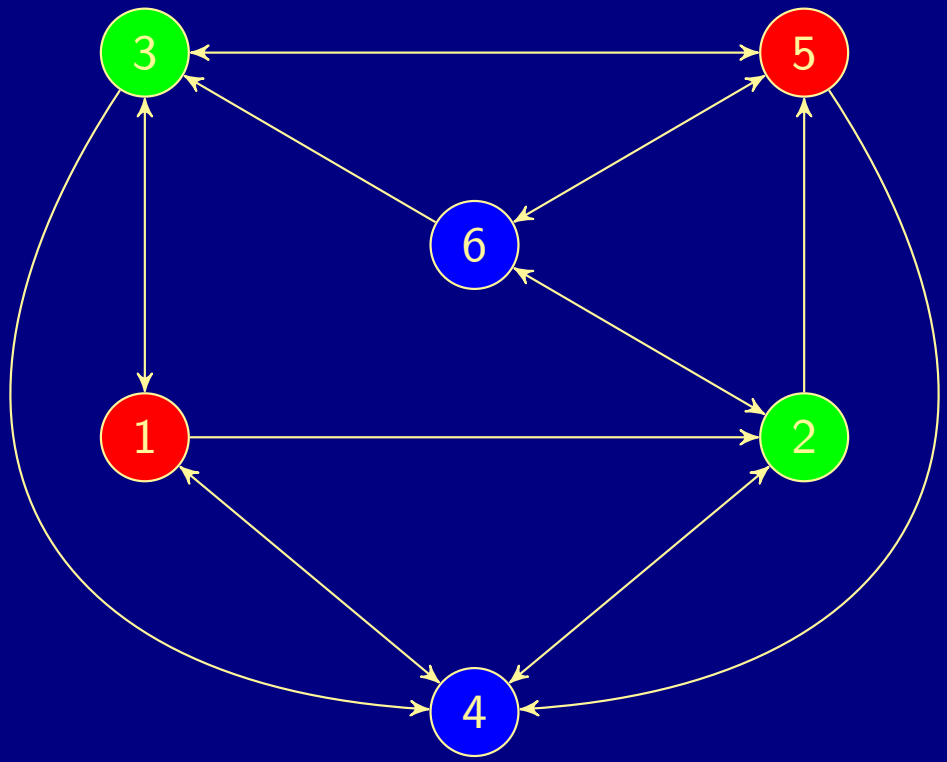
```
Answer: 6
node(1)   [...]    \
assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
```



Potassco

A coloring

```
Answer: 6
node(1)   [...]    \
assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
```



Outline

Potassco

Basic methodology

Methodology

Generate and Test (or: Guess and Check)

Generator: Generate potential stable model candidates (typically through non-deterministic constructs)

Tester: Eliminate invalid candidates (typically through integrity constraints)

Nutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Potassco

Basic methodology

Methodology

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

Generator: Generate potential stable model candidates (typically through non-deterministic constructs)

Tester: Eliminate invalid candidates (typically through integrity constraints)

Nutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Data

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Data

```
{ assign(N,C) : color(C) } = 1 :- node(N).
```

Generator

```
:- edge(N,M), assign(N,C), assign(M,C).
```

Tester

Potassco

Outline

Potassco

Outline

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true

- Example: Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator		**Tester**		**Stable models**		
$\{a\}$	$\leftarrow$	$\leftarrow$	$\sim a, b$	X_1	$=$	$\{a, b\}$
$\{b\}$	$\leftarrow$	$\leftarrow$	$a, \sim b$	X_2	$=$	$\{\}$

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true

- Example: Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator		**Tester**		**Stable models**		
$\{a\}$	$\leftarrow$	$\leftarrow$	$\sim a, b$	X_1	$=$	$\{a, b\}$
$\{b\}$	$\leftarrow$	$\leftarrow$	$a, \sim b$	X_2	$=$	$\{\}$

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true

- Example: Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator		**Tester**		**Stable models**		
$\{a\}$	$\leftarrow$	$\leftarrow$	$\sim a, b$	X_1	$=$	$\{a, b\}$
$\{b\}$	$\leftarrow$	$\leftarrow$	$a, \sim b$	X_2	$=$	$\{\}$

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true

- Example: Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator		**Tester**		**Stable models**		
$\{a\}$	$\leftarrow$	$\leftarrow$	$\sim a, b$	X_1	$=$	$\{a, b\}$
$\{b\}$	$\leftarrow$	$\leftarrow$	$a, \sim b$	X_2	$=$	$\{\}$

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true

- Example: Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

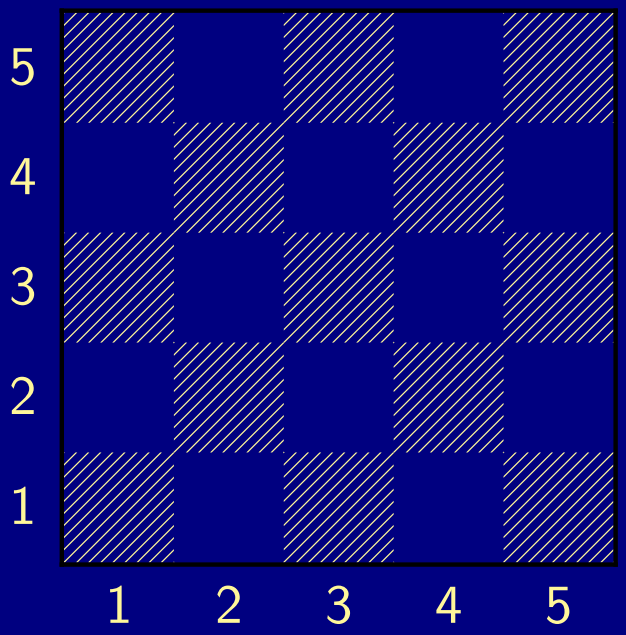
Generator		**Tester**		**Stable models**		
$\{a\}$	$\leftarrow$	$\leftarrow$	$\sim a, b$	X_1	$=$	$\{a, b\}$
$\{b\}$	$\leftarrow$	$\leftarrow$	$a, \sim b$	X_2	$=$	$\{\}$

Potassco

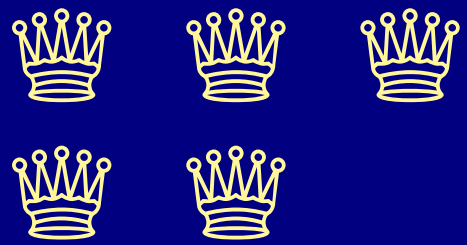
Outline

Potassco

The n-queens problem



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



Defining the field

queens.lp

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

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

Defining the field

Running ...

```
$ clingo queens.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.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

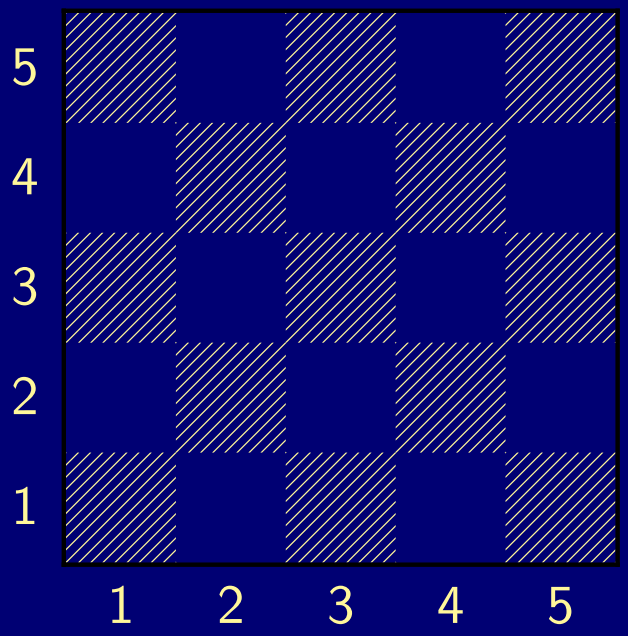
Running ...

```
$ clingo queens.lp --const n=5 3
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+
```

Potassco

Placing some queens

```
Answer: 1
```

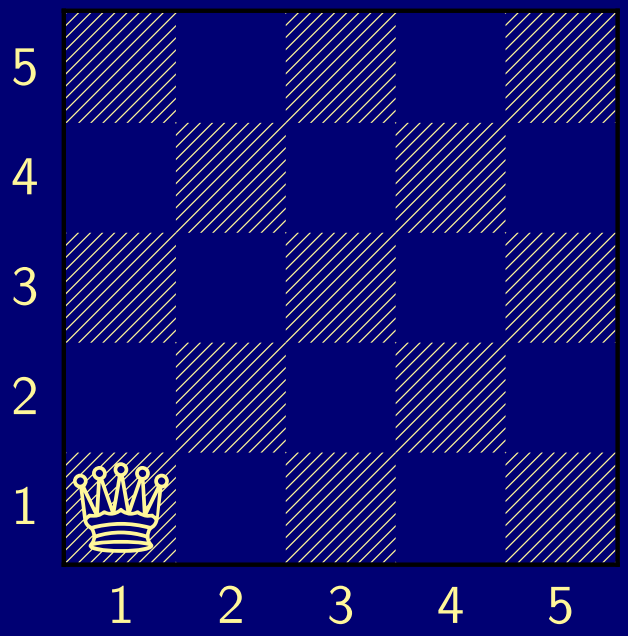



```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
```

Potassco

Placing some queens

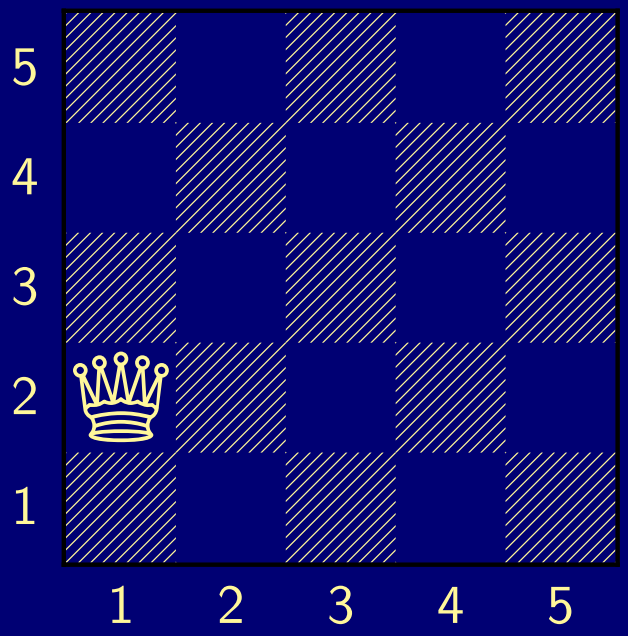
```
Answer: 2
```




```
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)
```

Placing some queens

```
Answer: 3
```




```
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)
```

Placing n queens

queens.lp

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

- Place exactly n queens on the board

Placing n queens

queens.lp

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

- Place exactly n queens on the board

Placing n queens

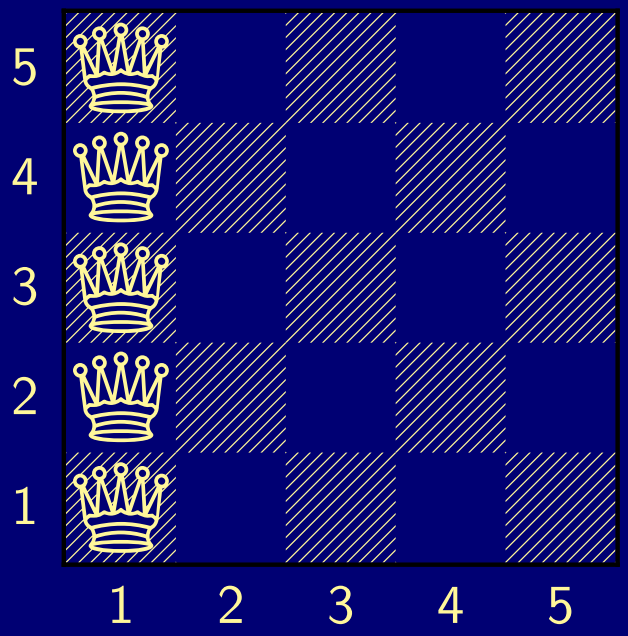
Running ...

```
$ clingo queens.lp --const n=5 2
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)
```

Potassco

Placing n queens

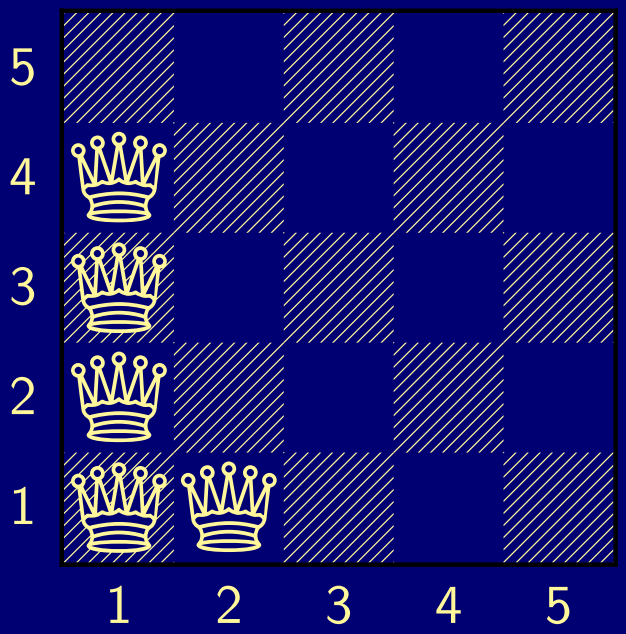
```
Answer: 1
```




```
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)
```

Placing n queens

```
Answer: 2
```




```
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)
```

Potassco

Horizontal and vertical attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

- Forbid horizontal attacks
- Forbid vertical attacks

Potassco

Horizontal and vertical attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

- Forbid horizontal attacks
- Forbid vertical attacks

Horizontal and vertical attack

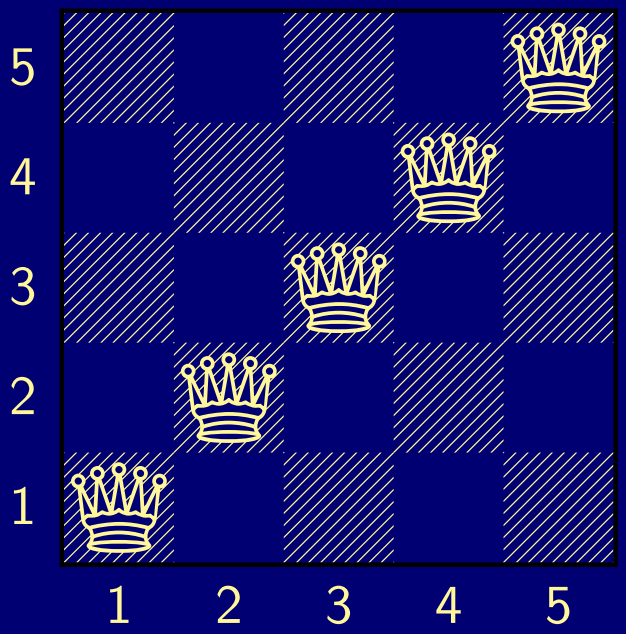
Running ...

```
$ clingo queens.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)
```

Potassco

Horizontal and vertical attack

```
Answer: 1
```




```
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)
```

Potassco

Diagonal attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

- Forbid diagonal attacks

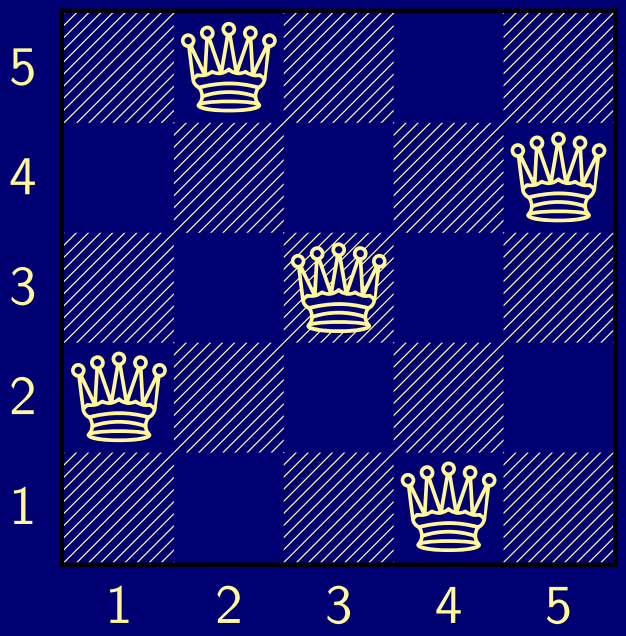
Diagonal attack

Running ...

```
$ clingo queens.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




```
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)
```

Potassco

Optimizing

queens-opt.lp

```
{ queen(I,1..n) } = 1 :- I = 1..n.
{ queen(1..n,J) } = 1 :- J = 1..n.
 :- { queen(D-J,J) } > 1, D = 2..2*n.
 :- { queen(D+J,J) } > 1, D = 1-n..n-1.
```

- Encoding can be optimized
- Much faster to solve

Potassco

And sometimes it rocks

```
$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2
clingo version 4.1.0
Solving...
SATISFIABLE

Models       : 1+
Time         : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time     : 3758.320s

Choices      : 288594554
Conflicts    : 3442   (Analyzed: 3442)
Restarts     : 17     (Average: 202.47 Last: 3442)
Model-Level  : 7594728.0
Problems     : 1      (Average Length: 0.00 Splits: 0)
Lemmas       : 3442   (Deleted: 0)
  Binary     : 0      (Ratio:   0.00%)
  Ternary    : 0      (Ratio:   0.00%)
  Conflict   : 3442   (Average Length: 229056.5 Ratio: 100.00%)
  Loop       : 0      (Average Length:    0.0 Ratio:   0.00%)
  Other      : 0      (Average Length:    0.0 Ratio:   0.00%)

Atoms        : 75084857 (Original: 75069989 Auxiliary: 14868)
Rules        : 100129956 (1: 50059992/100090100 2: 39990/29856 3: 10000/10000)
Bodies       : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight        : Yes
Variables    : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints  : 66664  (Binary: 35.6% Ternary:  0.0% Other: 64.4%)

Backjumps    : 3442   (Average: 681.19 Max: 169512 Sum: 2344658)
  Executed   : 3442   (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
  Bounded    : 0      (Average:  0.00 Max:   0 Sum:      0 Ratio:   0.00%)
```

Potassco

And sometimes it rocks

```
$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2
clingo version 4.1.0
Solving...
SATISFIABLE

Models       : 1+
Time         : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time     : 3758.320s

Choices      : 288594554
Conflicts    : 3442   (Analyzed: 3442)
Restarts     : 17     (Average: 202.47 Last: 3442)
Model-Level  : 7594728.0
Problems     : 1      (Average Length: 0.00 Splits: 0)
Lemmas       : 3442   (Deleted: 0)
  Binary     : 0      (Ratio:   0.00%)
  Ternary    : 0      (Ratio:   0.00%)
  Conflict   : 3442   (Average Length: 229056.5 Ratio: 100.00%)
  Loop       : 0      (Average Length:    0.0 Ratio:   0.00%)
  Other      : 0      (Average Length:    0.0 Ratio:   0.00%)

Atoms        : 75084857 (Original: 75069989 Auxiliary: 14868)
Rules        : 100129956 (1: 50059992/100090100 2: 39990/29856 3: 10000/10000)
Bodies       : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight        : Yes
Variables    : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints  : 66664  (Binary: 35.6% Ternary:  0.0% Other: 64.4%)

Backjumps    : 3442   (Average: 681.19 Max: 169512 Sum: 2344658)
  Executed   : 3442   (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
  Bounded    : 0      (Average:  0.00 Max:   0 Sum:      0 Ratio:   0.00%)
```

Potassco

Outline

Potassco

The traveling salesperson problem (TSP)

- Problem Instance A set of cities and distances among them, or simply a weighted graph
- Problem Class What is the shortest possible route visiting each city and returning to the city of origin?
- Note
  - TSP extends the Hamiltonian cycle problem: Is there a cycle in a graph visiting each node exactly once
  - TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem

Potassco

The traveling salesperson problem (TSP)

- Problem Instance A set of cities and distances among them, or simply a weighted graph
- Problem Class What is the shortest possible route visiting each city and returning to the city of origin?
- Note
  - TSP extends the Hamiltonian cycle problem: Is there a cycle in a graph visiting each node exactly once
  - TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem

Potassco

Traveling salesperson

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
```

Potassco

Traveling salesperson

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
```

Potassco

Traveling salesperson

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
```

Potassco

Traveling salesperson

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).

edge(X,Y) :- cost(X,Y,_).
```

Potassco

Traveling salesperson

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).

edge(X,Y) :- cost(X,Y,_).
node(X) :- cost(X,_,_).   node(Y) :- cost(_,Y,_).
```

Potassco

Traveling salesperson

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Traveling salesperson

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Traveling salesperson

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Potassco

Traveling salesperson

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Potassco

Outline

Potassco

Reviewer Assignment

- Problem Instance A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests
- Problem Class A nice assignment of three reviewers to each paper

Reviewer Assignment

- Problem Instance A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests
- Problem Class A "nice" assignment of three reviewers to each paper

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment
by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
reviewer(r1). paper(p1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
reviewer(r2). paper(p2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

#count { P,R : assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 <= #count { P,R : assigned(P,R), paper(P) } <= 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 <= #count { P,R : assignedB(P,R), paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
reviewer(r1). paper(p1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
reviewer(r2). paper(p2). classA(r1,p3). classB(r1,p4). coi(r1,p6).
[...]

#count { P,R : assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 <= #count { P,R : assigned(P,R), paper(P) } <= 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 <= #count { P,R : assignedB(P,R), paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Outline

Potassco

Simplified STRIPS[1] Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2

  plan $\langle a, b \rangle$

[1] Stanford Research Institute Problem Solver, 1971

Simplified STRIPS[1] Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2
  - plan $\langle a, b \rangle$

[1] Stanford Research Institute Problem Solver, 1971

Simplified STRIPS[1] Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2
  - plan $\langle a, b \rangle$

[1] Stanford Research Institute Problem Solver, 1971

Simplified STRIPS[1] Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2
  - plan $\langle a, b \rangle$

[1] Stanford Research Institute Problem Solver, 1971

Simplistic STRIPS Planning

```
time(1..k).

fluent(p).      action(a).      action(b).      init(p).
fluent(q).        pre(a,p).       pre(b,q).
fluent(r).        add(a,q).       add(b,r).     query(r).
                  del(a,p).       del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Potassco

Simplistic STRIPS Planning

```
time(1..k).

fluent(p).      action(a).      action(b).        init(p).
fluent(q).        pre(a,p).       pre(b,q).
fluent(r).        add(a,q).       add(b,r).       query(r).
                  del(a,p).       del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Potassco

Simplistic STRIPS Planning

```
time(1..k).

fluent(p).      action(a).      action(b).        init(p).
fluent(q).        pre(a,p).       pre(b,q).
fluent(r).        add(a,q).       add(b,r).       query(r).
                  del(a,p).       del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Potassco

Simplistic STRIPS Planning

```
time(1..k).

fluent(p).      action(a).      action(b).        init(p).
fluent(q).        pre(a,p).       pre(b,q).
fluent(r).        add(a,q).       add(b,r).       query(r).
                  del(a,p).       del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Potassco

[1] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[2] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[3] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[4] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[5] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving

Potassco

In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[6] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.
In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[7] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.

Potassco

IOS Press, 2009.

[9] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[10] G. Brewka, I. Niemelä, and M. Truszczyński.
Answer set optimization.
In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[11] K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[12] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

Potassco

[13] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[14] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[15] M. Davis and H. Putnam.
A computing procedure for quantification theory.
Journal of the ACM, 7:201–215, 1960.

[16] E. Di Rosa, E. Giunchiglia, and M. Maratea.
Solving satisfiability problems with preferences.
Constraints, 15(4):485–515, 2010.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.

Potassco

Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.
In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

Potassco

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.
Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.

Potassco

In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

[26] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub.
Abstract Gringo.
Theory and Practice of Logic Programming, 15(4-5):449–463, 2015.
Available at `http://arxiv.org/abs/1507.06576`.

Potassco

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.
University of Potsdam, second edition edition, 2015.

[28] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[29] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[30] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.

Potassco

On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [49], pages 250–264.

[31] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [3], pages 260–265.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [3], pages 136–148.

[34] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [74], pages 386–392.

Potassco

[35] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[36] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[37] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*

Potassco

(CPAIOR'09), volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[38] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [49], pages 235–249.

[39] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[40] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

Potassco

[41] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[42] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[43] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[44] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.

Potassco

In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[45] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[46] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.
Journal of Automated Reasoning, 36(4):345–377, 2006.

[47] K. Gödel.
Zum intuitionistischen Aussagenkalkül.
Anzeiger der Akademie der Wissenschaften in Wien, page 65–66, 1932.

[48] A. Heyting.

Potassco

Die formalen Regeln der intuitionistischen Logik.
In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. Deutsche Akademie der Wissenschaften zu Berlin, 1930. Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[49] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[50] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [74], pages 2318–2323.

[51] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[52] J. Lee.

Potassco

A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[53] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[54] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[55] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[56] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?

Potassco

ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[57] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

[58] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[59] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[60] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

[61] J. Marques-Silva and K. Sakallah.

Potassco

GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[62] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.
In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[63] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[64] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

Potassco

[65] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[66] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[67] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[68] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.

Potassco

In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[69] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[70] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.
Theory and Practice of Logic Programming, 6(5):559–608, 2006.

[72] T. Syrjänen.
Lparse 1.0 user's manual, 2001.

[73] A. Van Gelder, K. Ross, and J. Schlipf.

Potassco

The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[74] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[75] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.
In R. Ernst, editor, *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. IEEE Computer Society Press, 2001.

Potassco