Answer Set Programming: Boolean Constraint Solving for Knowledge Representation and Reasoning

Torsten Schaub

University of Potsdam

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

7 Summary

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

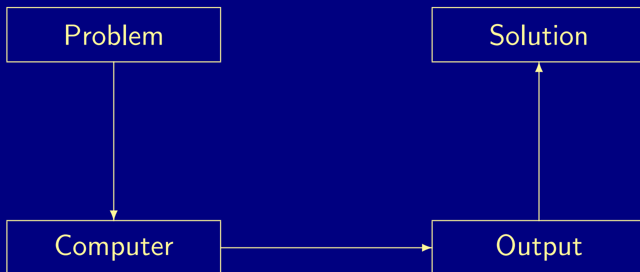
6 Potassco

7 Summary

Potassco

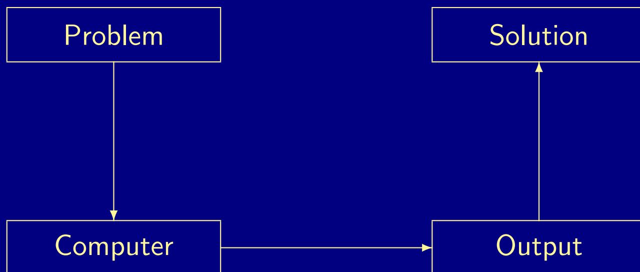
Informatics

"What is the problem?" versus *"How to solve the problem?"*



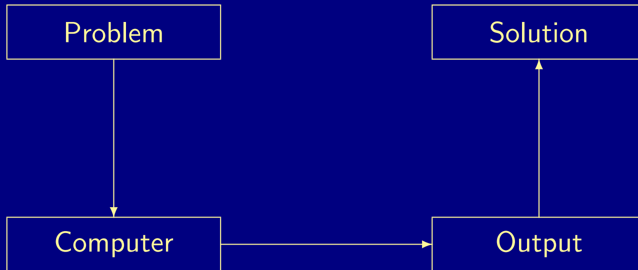
Informatics

"What is the problem?" versus *"How to solve the problem?"*



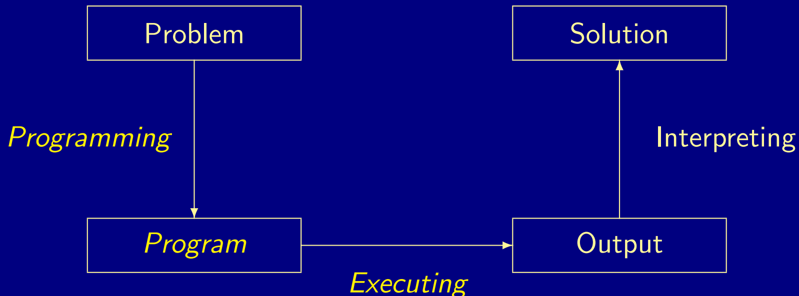
Traditional programming

"What is the problem?" versus *"How to solve the problem?"*



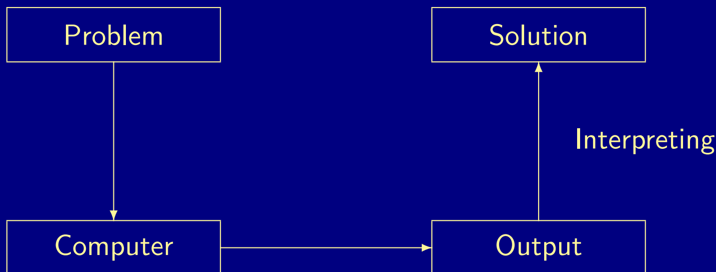
Traditional programming

"What is the problem?" versus *"How to solve the problem?"*



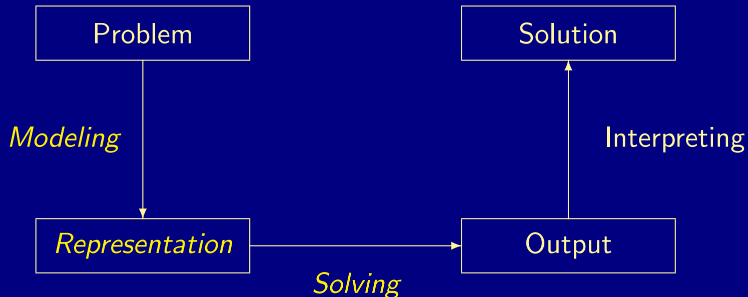
Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



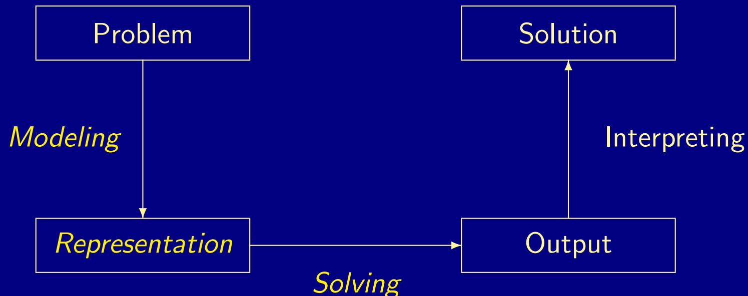
Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Potassco

Model Generation based Problem Solving

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions

Potassco

Model Generation based Problem Solving

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions
⋮	⋮

Potassco

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Potassco

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	**NMR**
default theories	extensions	**NMR**
⋮	⋮	

Potassco

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	**NMR**
default theories	extensions	**NMR**
⋮	⋮	

Potassco

Answer Set Programming *in general*

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	
propositional theories	minimal models	
propositional theories	stable models	**ASP**
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	**ASP**
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Potassco

Answer Set Programming *in general*

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	
propositional theories	minimal models	
propositional theories	stable models	**ASP**
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	**ASP**
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Potassco

Answer Set Programming *in practice*

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions
first-order programs	stable Herbrand models

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Potassco

LP-style playing with blocks

Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Prolog queries

```
?- above(a,c).
true.

?- above(c,a).
no.
```

Potassco

LP-style playing with blocks

Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Prolog queries

```
?- above(a,c).
true.

?- above(c,a).
no.
```

Potassco

LP-style playing with blocks

Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Prolog queries

```
?- above(a,c).
true.

?- above(c,a).
no.
```

Potassco

LP-style playing with blocks

Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Prolog queries (testing entailment)

```
?- above(a,c).
true.

?- above(c,a).
no.
```

Potassco

LP-style playing with blocks

Shuffled Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
```

Prolog queries

```
?- above(a,c).

Fatal Error: local stack overflow.
```

Potassco

LP-style playing with blocks

Shuffled Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
```

Prolog queries

```
?- above(a,c).

Fatal Error: local stack overflow.
```

Potassco

LP-style playing with blocks

Shuffled Prolog program

```
on(a,b).
on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
```

Prolog queries (answered via fixed execution)

```
?- above(a,c).

Fatal Error: local stack overflow.
```

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Potassco

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{
\begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

SAT-style playing with blocks

Formula

LaTeX

$$
\begin{array}{rl}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{ \begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array} \right\}
$$

Potassco

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
& on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

DIMACS

```
p cnf 18 38
2 0
6 0
-1 10 0
-2 11 0
-3 12 0
-4 13 0
-5 14 0
-6 15 0
-7 16 0
-8 17 0
-9 18 0
-1 -10 -10 0
-2 -13 -10 0
-3 -16 -10 0
-4 -10 -13 0
```

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
& on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{
\begin{array}{rrrrr}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

Potassco

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{
\begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a, b) \\
\wedge & on(b, c) \\
\wedge & (on(X, Y) \rightarrow above(X, Y)) \\
\wedge & (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))
\end{array}
$$

Herbrand model

$$
\left\{ \begin{array}{rrrrr}
on(a, b), & on(b, c), & on(a, c), & on(b, b), & \\
above(a, b), & above(b, c), & above(a, c), & above(b, b), & above(c, b)
\end{array} \right\}
$$

Potassco

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{
\begin{array}{rrrrr}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

Potassco

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model (among 426!)

$$
\left\{
\begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

➥ Answer Set Programming (ASP)

Potassco

ASP-style playing with blocks

Logic program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Stable Herbrand model

$\{ \mathtt{on}(\mathtt{a},\mathtt{b}),\ \mathtt{on}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{a},\mathtt{b}),\ \mathtt{above}(\mathtt{a},\mathtt{c}) \}$

Potassco

ASP-style playing with blocks

Logic program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Stable Herbrand model

$$\{ \texttt{on}(\texttt{a},\texttt{b}), \texttt{on}(\texttt{b},\texttt{c}), \texttt{above}(\texttt{b},\texttt{c}), \texttt{above}(\texttt{a},\texttt{b}), \texttt{above}(\texttt{a},\texttt{c}) \}$$

Potassco

ASP-style playing with blocks

Logic program

```
on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Stable Herbrand model (and no others)

$$\{ \texttt{on}(\texttt{a},\texttt{b}),\ \texttt{on}(\texttt{b},\texttt{c}),\ \texttt{above}(\texttt{b},\texttt{c}),\ \texttt{above}(\texttt{a},\texttt{b}),\ \texttt{above}(\texttt{a},\texttt{c}) \}$$

Potassco

ASP-style playing with blocks

Logic program

```
on(a,b).
on(b,c).

above(X,Y) :- above(Z,Y), on(X,Z).
above(X,Y) :- on(X,Y).
```

Stable Herbrand model (and no others)

$\{ \texttt{on}(a,b),\ \texttt{on}(b,c),\ \texttt{above}(b,c),\ \texttt{above}(a,b),\ \texttt{above}(a,c) \}$

Potassco

ASP versus LP

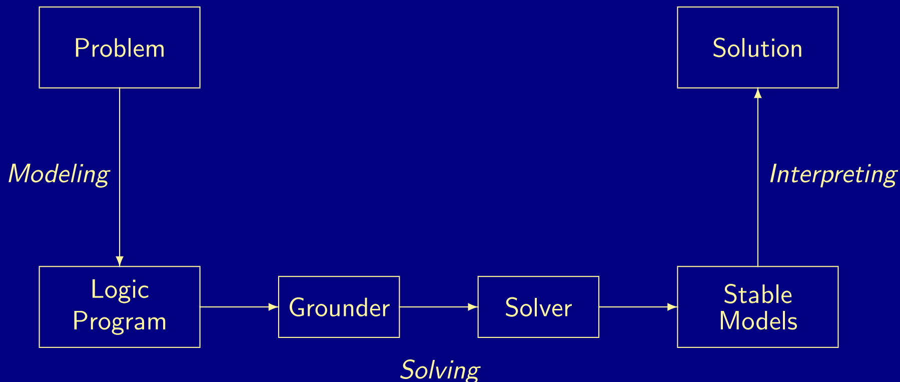
ASP	Prolog
Model generation	Query orientation
Bottom-up	Top-down
Modeling language	Programming language
Rule-based format	
Instantiation Flat terms	Unification Nested terms
(Turing +) $NP(^{NP})$	Turing

Potassco

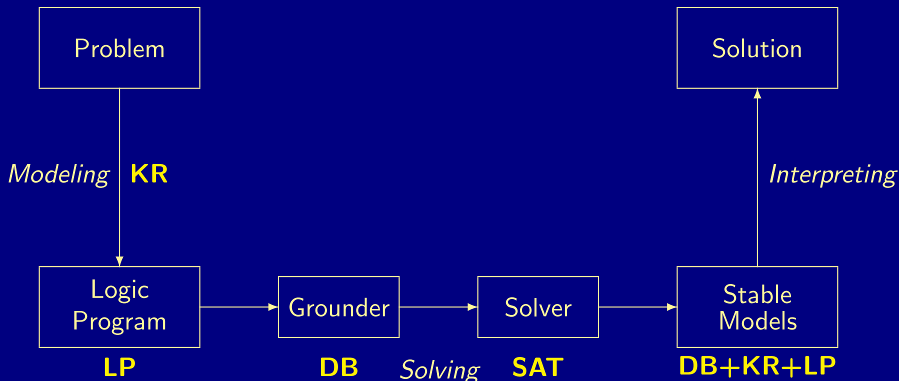
ASP versus SAT

ASP	SAT
Model generation	
Bottom-up	
Constructive Logic	Classical Logic
Closed (and open) world reasoning	Open world reasoning
Modeling language	—
Complex reasoning modes Satisfiability Enumeration/Projection Intersection/Union Optimization	Satisfiability testing Satisfiability — — —
(Turing +) $NP(^{NP})$	NP

ASP solving



Rooting ASP solving



Answer Set Programming

in a Hazelnutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

  tailored to Knowledge Representation and Reasoning

Potassco

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

  tailored to Knowledge Representation and Reasoning

- Declarativity ASP does separate a problem's representation from the algorithms used for solving it

Potassco

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

  tailored to Knowledge Representation and Reasoning

- Declarativity ASP does separate a problem's representation from the algorithms used for solving it
- Scalability ASP does not separate a problem's representation from its induced combinatorics

Potassco

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

  tailored to Knowledge Representation and Reasoning

ASP = DB+LP+KR+SAT

Potassco

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

  tailored to Knowledge Representation and Reasoning

$$\mathbf{ASP} = \mathbf{DB}+\mathbf{LP}+\mathbf{KR}+\mathbf{SMT}^{n}$$

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

7 Summary

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \wedge (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ
$$\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula Φ has one stable model, often called answer set:

P_Φ

$$\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \; {\sim}r \end{array}$$

$$\{p, q\}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$\{p,q\}, \{q,r\}$, and $\{p,q,r\}$

Φ $\quad q \wedge (q \wedge \neg r \rightarrow p)$

Formula Φ has one stable model, often called answer set:

$\{p,q\}$

$p \mapsto 1$
$q \mapsto 1$
$r \mapsto 0$

P_Φ
$q \leftarrow$
$p \leftarrow q, \sim r$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ

$$\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \; \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

$$\Phi \quad \boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$$

$$P_\Phi \quad \begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

P_Φ
$$\begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \; \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ
$$\begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \ {\sim}r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Semantic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

$$\Phi \quad \boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

$$P_\Phi \quad \begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Potassco

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{\, a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potassco

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{\, a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potasso

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules

- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{ a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potassco

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{ a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potasso

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{\, a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset\}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potassco

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{\, a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$

  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potassco

Making things precise

- Syntax
  - A (normal) rule is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, \sim c_1, \dots, \sim c_n,$$

    where $0 \leq m, n$ and each a, b_i, c_j is an atom
  - A logic program is a finite set of rules
- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is defined by

    $$P^X = \{\, a \leftarrow b_1, \dots, b_m \mid r \in P \text{ and } \{c_1, \dots, c_n\} \cap X = \emptyset \}$$
  - The $\subseteq$–smallest model of P^X is denoted by $Cn(P^X)$
  - A set X of atoms is an stable model of a program P, if

    $$X = Cn(P^X)$$

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p\}$	$p \leftarrow p$	$\emptyset$	
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✗
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✗

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\quad\}$	$p \leftarrow p$	$\emptyset$	✗
$\{\quad q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✗

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\quad\}$	$p \leftarrow p$	$\emptyset$	
$\{\quad q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Potassco

A simple example

$$P = \{p \leftarrow p, \; q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\quad\}$	$p \leftarrow p$	$\emptyset$	✗
$\{\quad q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✗

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Potassco

A simple example

$$P = \{p \leftarrow p,\ q \leftarrow \neg p\}$$

X	P^X	$Cn(P^X)$	*stable?*
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Potassco

A simple example

$P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

X	P^X	$Cn(P^X)$	*stable?*	*model?*
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘	✔
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘	✔

Potassco

Some properties

- A logic program may have zero, one, or multiple stable models
- If X is a stable model of a logic program P, then X is a model of P (seen as a formula)
- If X and Y are stable models of a *normal* program P, then $X \not\subset Y$

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

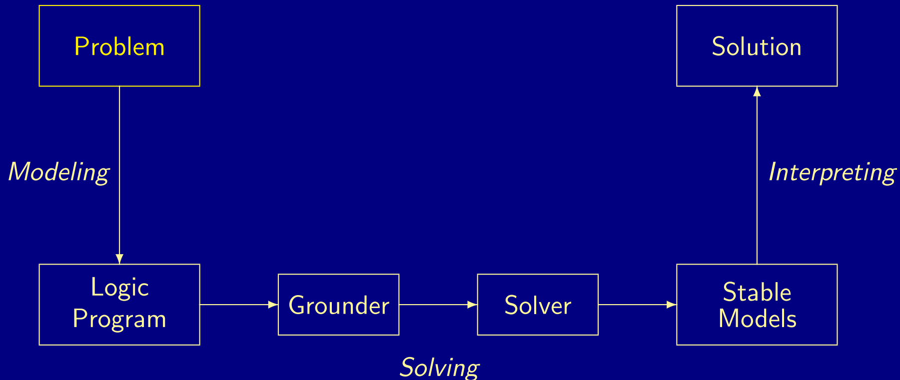
5 Solving

6 Potassco

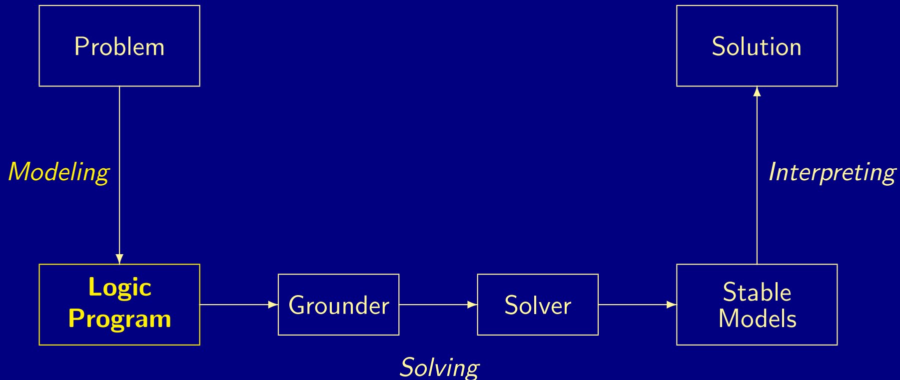
7 Summary

Potassco

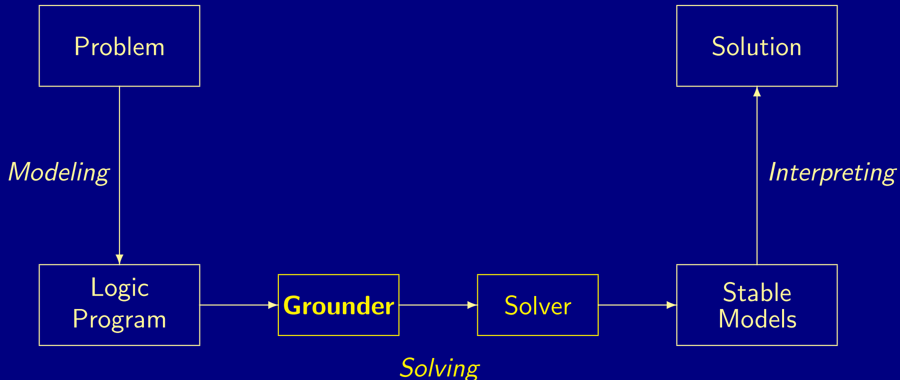
ASP solving process



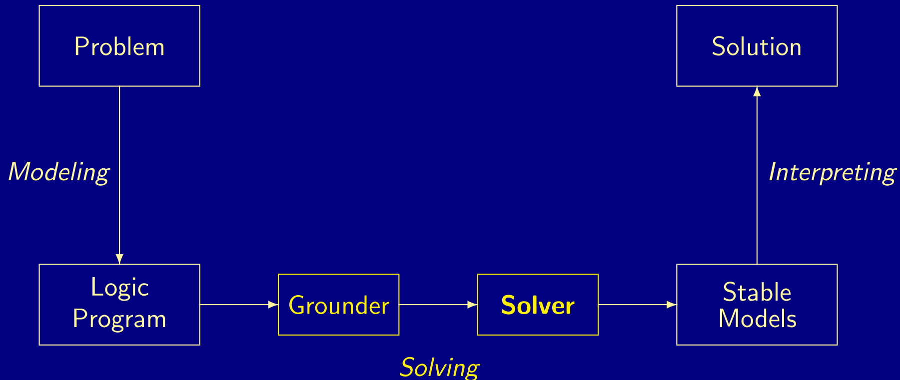
ASP solving process



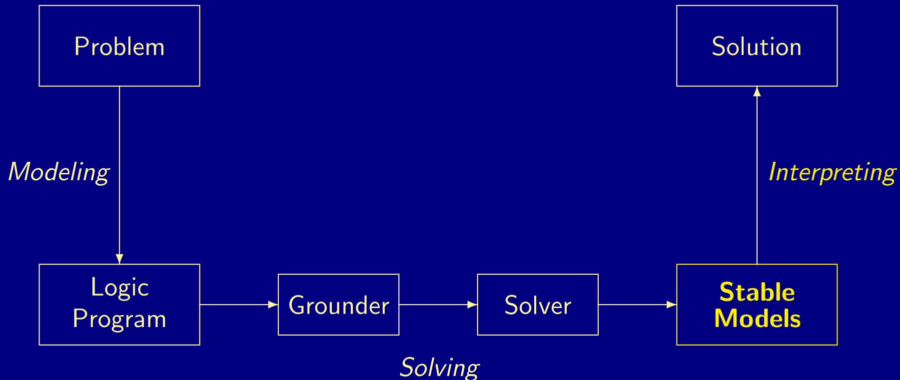
ASP solving process



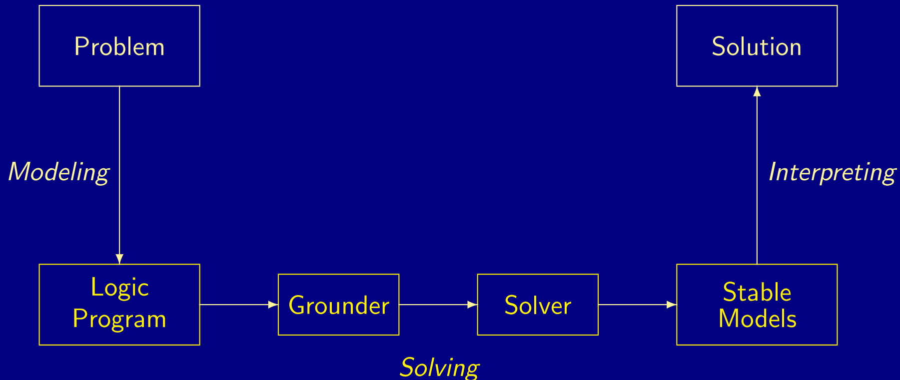
ASP solving process



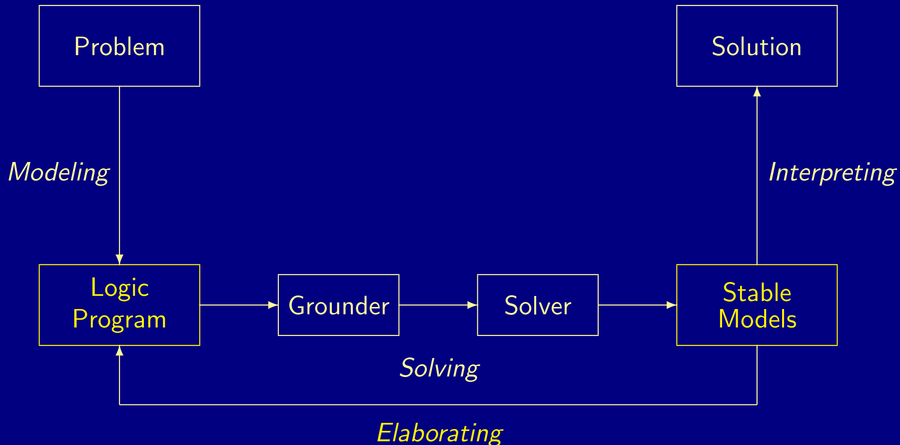
ASP solving process



ASP solving process



ASP solving process

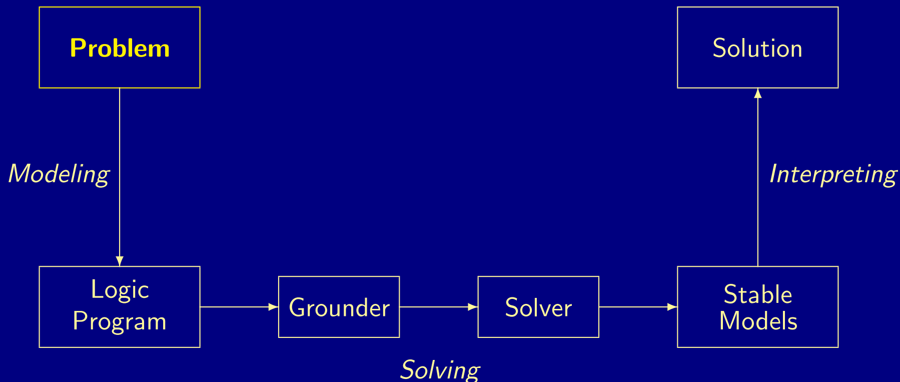


Some language constructs

- Variables (over the Herbrand universe)
  - `p(X) :- q(X)` over constants $\{a, b, c\}$ stands for
    `p(a) :- q(a), p(b) :- q(b), p(c) :- q(c)`
- Conditional Literals
  - `p :- q(X) : r(X)` given `r(a), r(b), r(c)` stands for
    `p :- q(a), q(b), q(c)`
- Disjunction
  - `p(X) | q(X) :- r(X)`
- Integrity Constraints
  - `:- q(X), p(X)`
- Choice
  - `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates
  - `s(Y) :- r(Y), 2 #count { p(X,Y) : q(X) } 7`
  - also: `#sum, #avg, #min, #max, #even, #odd`

Potassco

A case-study: Graph coloring



Graph coloring

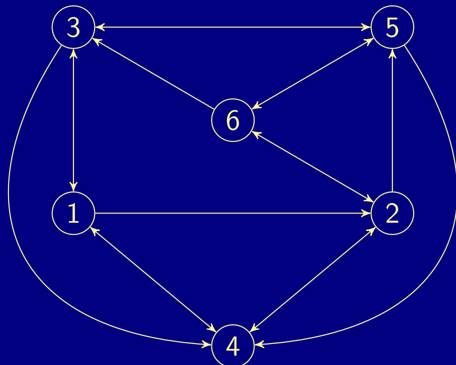
- Problem instance A graph consisting of nodes and edges

Graph coloring

- Problem instance A graph consisting of nodes and edges

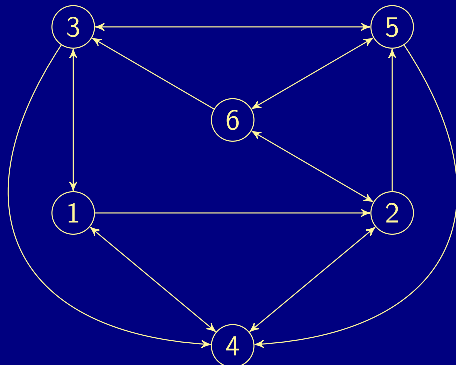
Graph coloring

- Problem instance A graph consisting of nodes and edges



Graph coloring

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



Graph coloring

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

Graph coloring

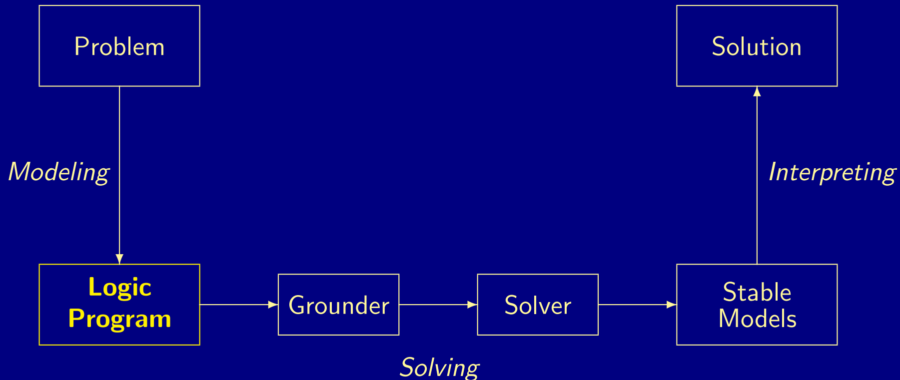
- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `col/1`
- Problem class Assign each node one color such that no two nodes connected by an edge have the same color

  In other words,

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

Potassco

ASP solving process



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).

col(r).   col(b).   col(g).
```

} Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

} Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,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).

col(r).   col(b).   col(g).
```

Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,C).
```

Problem encoding

Potassco

color.lp

```
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).

col(r).   col(b).   col(g).
```

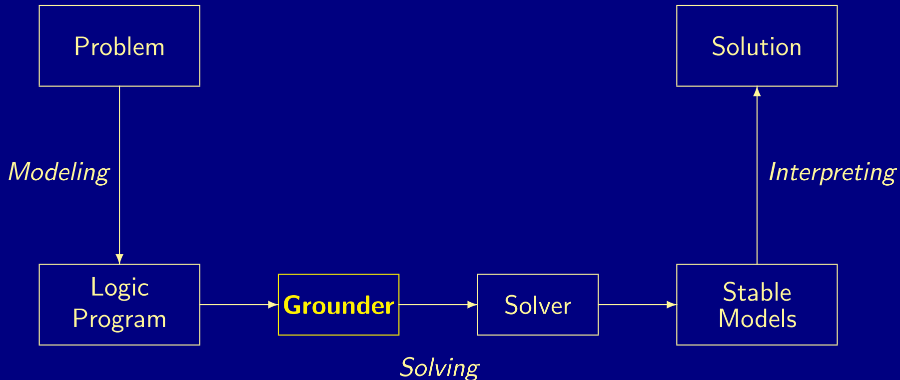
Problem instance

```
1 { color(X,C) : col(C) } 1 :- node(X).

:- edge(X,Y), color(X,C), color(Y,C).
```

Problem encoding

Potassco

ASP solving process



Graph coloring: Grounding

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

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

col(r).  col(b).  col(g).

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

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

Potassco

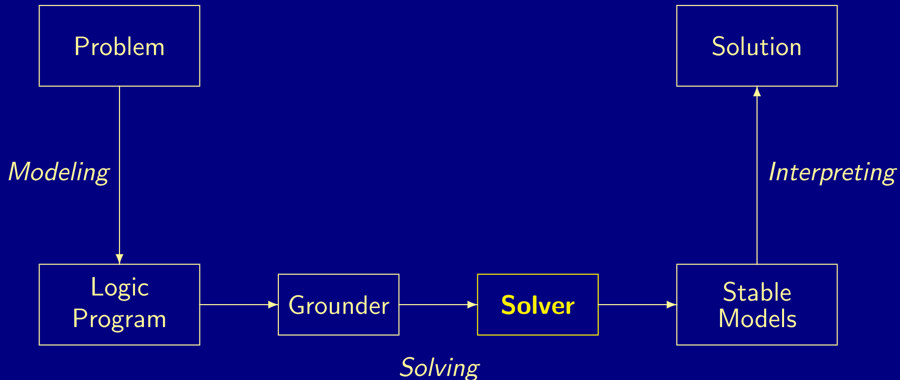
Graph coloring: Grounding

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

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

col(r).  col(b).  col(g).

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

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

Potassco

ASP solving process



Potassco

Graph coloring: Solving

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

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

Models      : 6
Time        : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

Potassco

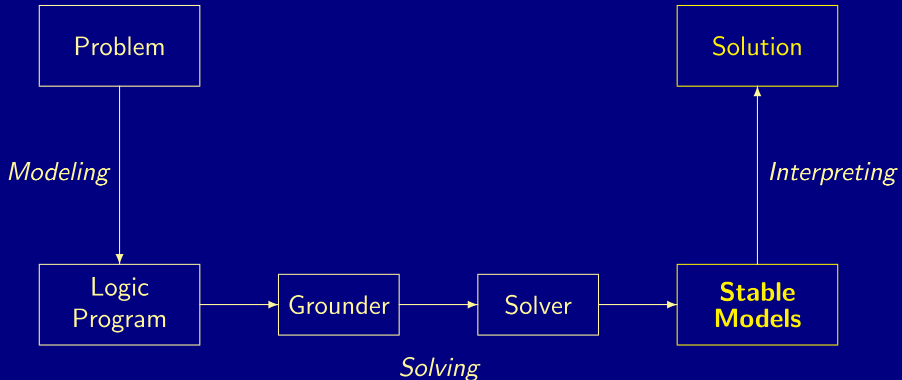
Graph coloring: Solving

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

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

Models      : 6
Time        : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

Potassco

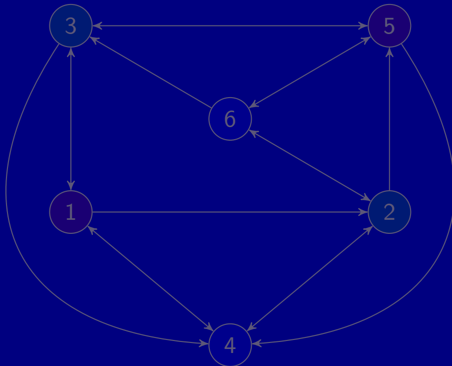
ASP solving process



Potassco

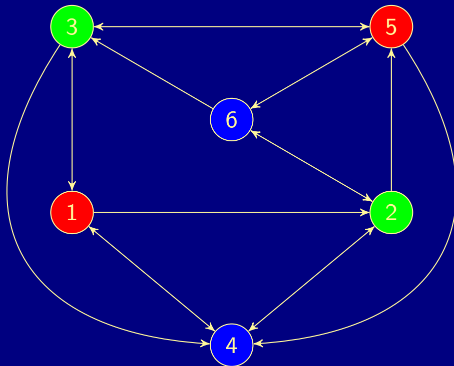
A coloring

```
Answer: 6
edge(1,2) ... col(r) ... node(1) ...                                  \
color(6,b) color(5,r) color(4,b) color(3,g) color(2,g) color(1,r)
```



A coloring

```
Answer: 6
edge(1,2) ... col(r) ... node(1) ...                                   \
color(6,b) color(5,r) color(4,b) color(3,g) color(2,g) color(1,r)
```



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

Satisfiability testing

$(a \leftrightarrow b) \land c$

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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


Potassco

Maximum satisfiability testing

"$(a \not\leftrightarrow b)$" + $(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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

 :~ a, b. [42@1]
 :~ not a, not b. [69@2]
```

n-queens
Basic encoding

```
{ queen(1..n,1..n) }.

 :- not { queen(I,J) } == n.
 :- queen(I,J), queen(I,JJ), J != JJ.
 :- queen(I,J), queen(II,J), I != II.
 :- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
 :- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

Potassco

n-queens

Advanced encoding

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

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

n-queens

(Experimental) constraint encoding

```
1 $<= $queen(1..n) $<= n.

#disjoint { X : $queen(X) $+ 0 : X=1..n }.
#disjoint { X : $queen(X) $+ X : X=1..n }.
#disjoint { X : $queen(X) $- X : X=1..n }.
```

Traveling salesperson
Basic encoding

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

reached(X) :- X = #min { Y : node(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

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

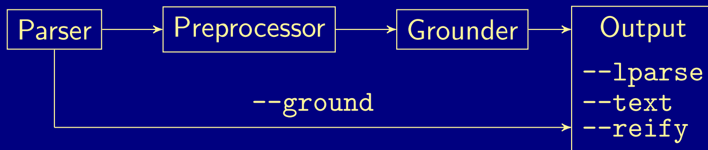
7 Summary

Potassco

gringo

- Accepts safe programs with aggregates
- Tolerates unrestricted use of function symbols
  (as long as it yields a finite ground instantiation :)
- Expressive power of a Turing machine

- Basic architecture of *gringo*:



Grounding by example

$$
\begin{array}{l}
d(a) \\
d(c) \\
d(d) \\
p(a,b) \\
p(b,c) \\
p(c,d) \\
p(X,Z) \leftarrow p(X,Y), p(Y,Z) \\
q(a) \\
q(b) \\
q(X) \leftarrow \sim r(X), d(X) \\
r(X) \leftarrow \sim q(X), d(X) \\
s(X) \leftarrow \sim r(X), p(X,Y), q(Y)
\end{array}
$$

Potassco

Grounding by example

Safe ?

$d(a)$
$d(c)$
$d(d)$

$p(a, b)$
$p(b, c)$
$p(c, d)$
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$

$q(a)$
$q(b)$
$q(X) \leftarrow \sim r(X), d(X)$

$r(X) \leftarrow \sim q(X), d(X)$
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$

Potassco

Grounding by example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Potassco

Grounding by example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Potassco

Grounding by example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	✔
$r(X) \leftarrow \sim q(X), d(X)$	✔
$s(X) \leftarrow \sim r(X), p(X,Y), q(Y)$	✔

Potassco

Match

- A substitution is a mapping from variables to terms
- Given sets B and D of atoms, a substitution θ is a match of B in D, if $B\theta \subseteq D$
- Given a set B of atoms and a set D of ground atoms, define

$$\Theta(B, D) = \{ \theta \mid \theta \text{ is a } \subseteq\text{-minimal match of } B \text{ in } D \}$$

- Example $\{X \mapsto 1\}$ and $\{X \mapsto 2\}$ are $\subseteq$-minimal matches of $\{p(X)\}$ in $\{p(1), p(2), p(3)\}$, while match $\{X \mapsto 1, Y \mapsto 2\}$ is not

Potassco

Match

- A substitution is a mapping from variables to terms
- Given sets B and D of atoms, a substitution θ is a match of B in D, if $B\theta \subseteq D$
- Given a set B of atoms and a set D of ground atoms, define

$$\Theta(B, D) = \{ \theta \mid \theta \text{ is a } \subseteq\text{-minimal match of } B \text{ in } D \}$$

- Example $\{X \mapsto 1\}$ and $\{X \mapsto 2\}$ are $\subseteq$-minimal matches of $\{p(X)\}$ in $\{p(1), p(2), p(3)\}$, while match $\{X \mapsto 1, Y \mapsto 2\}$ is not

Potassco

Match

- A substitution is a mapping from variables to terms
- Given sets B and D of atoms, a substitution θ is a match of B in D, if $B\theta \subseteq D$
- Given a set B of atoms and a set D of ground atoms, define

$$\Theta(B, D) = \{\, \theta \mid \theta \text{ is a } \subseteq\text{-minimal match of } B \text{ in } D \,\}$$

- Example $\{X \mapsto 1\}$ and $\{X \mapsto 2\}$ are $\subseteq$-minimal matches of $\{p(X)\}$ in $\{p(1), p(2), p(3)\}$, while match $\{X \mapsto 1, Y \mapsto 2\}$ is not

Potassco

Naive instantiation

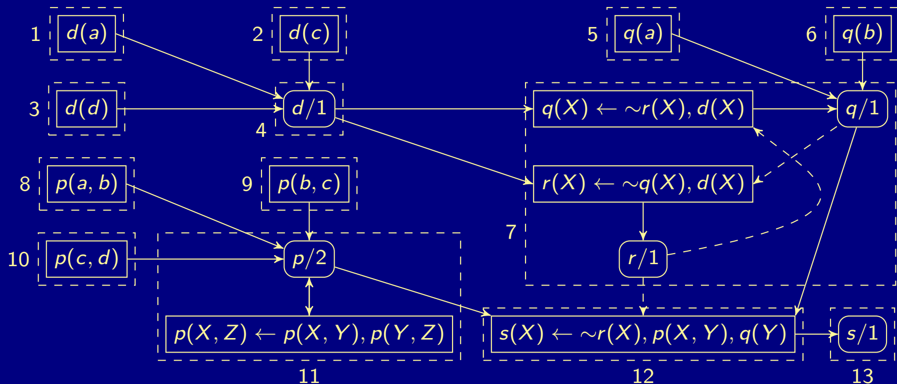
Algorithm 1: NaiveInstantiation

Input : A safe (first-order) logic program P
Output : A ground logic program P'

```
D := ∅
P' := ∅
repeat
    D' := D
    foreach r ∈ P do
        B := body(r)^+
        foreach θ ∈ Θ(B, D) do
            D := D ∪ {head(r)θ}
            P' := P' ∪ {rθ}
until D = D'
```

Potassco

Predicate-rule dependency graph



Instantiation

SCC	$\Theta(B,D)$	D	P'
1	$\{\emptyset\}$	$d(a)$	$d(a) \leftarrow$
2	$\{\emptyset\}$	$d(c)$	$d(c) \leftarrow$
3	$\{\emptyset\}$	$d(d)$	$d(d) \leftarrow$
5	$\{\emptyset\}$	$q(a)$	$q(a) \leftarrow$
6	$\{\emptyset\}$	$q(b)$	$q(b) \leftarrow$
7	$\{\{X \mapsto a\},$		$q(a) \leftarrow {\sim}r(a), d(a)$
	$\{X \mapsto c\},$	$q(c)$	$q(c) \leftarrow {\sim}r(c), d(c)$
	$\{X \mapsto d\},$	$q(d)$	$q(d) \leftarrow {\sim}r(d), d(d)$
	$\{X \mapsto a\},$		$r(a) \leftarrow {\sim}q(a), d(a)$
	$\{X \mapsto c\},$	$r(c)$	$r(c) \leftarrow {\sim}q(c), d(c)$
	$\{X \mapsto d\}\}$	$r(d)$	$r(d) \leftarrow {\sim}q(d), d(d)$

Potassco

Instantiation

SCC	$\Theta(B,D)$	D	P'
8	$\{\emptyset\}$	$p(a,b)$	$p(a,b) \leftarrow$
9	$\{\emptyset\}$	$p(b,c)$	$p(b,c) \leftarrow$
10	$\{\emptyset\}$	$p(c,d)$	$p(c,d) \leftarrow$
11	$\{\{X \mapsto a, Y \mapsto b, Z \mapsto c\},$	$p(a,c)$	$p(a,c) \leftarrow p(a,b), p(b,c)$
	$\{X \mapsto b, Y \mapsto c, Z \mapsto d\}\}$	$p(b,d)$	$p(b,d) \leftarrow p(b,c), p(c,d)$
	$\{\{X \mapsto a, Y \mapsto c, Z \mapsto d\},$	$p(a,d)$	$p(a,d) \leftarrow p(a,c), p(c,d)$
	$\{X \mapsto a, Y \mapsto b, Z \mapsto d\}\}$		$p(a,d) \leftarrow p(a,b), p(b,d)$
12	$\{\{X \mapsto a, Y \mapsto b\},$	$s(a)$	$s(a) \leftarrow {\sim}r(a), p(a,b), q(b)$
	$\{X \mapsto a, Y \mapsto c\},$		$s(a) \leftarrow {\sim}r(a), p(a,c), q(c)$
	$\{X \mapsto a, Y \mapsto d\},$		$s(a) \leftarrow {\sim}r(a), p(a,d), q(d)$
	$\{X \mapsto b, Y \mapsto c\},$	$s(b)$	$s(b) \leftarrow {\sim}r(b), p(b,c), q(c)$
	$\{X \mapsto b, Y \mapsto d\},$		$s(b) \leftarrow {\sim}r(b), p(b,d), q(d)$
	$\{X \mapsto c, Y \mapsto d\}\}$	$s(c)$	$s(c) \leftarrow {\sim}r(c), p(c,d), q(d)$

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

7 Summary

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on “source pointers”
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Reasoning modes of *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for:
  - Optimization
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solution computation)
  - and combinations thereof

- *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Potassco

Reasoning modes of *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for:
  - Optimization
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solution computation)
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Potassco

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Potassco

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Potassco

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

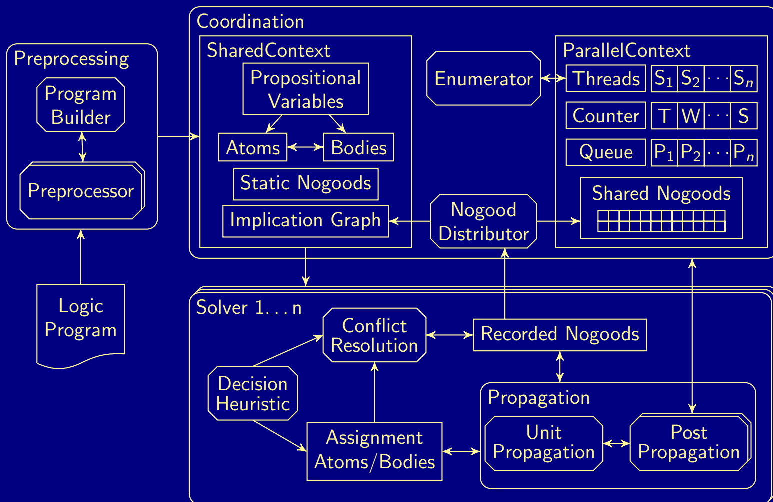
Potassco

Parallel search in *clasp*

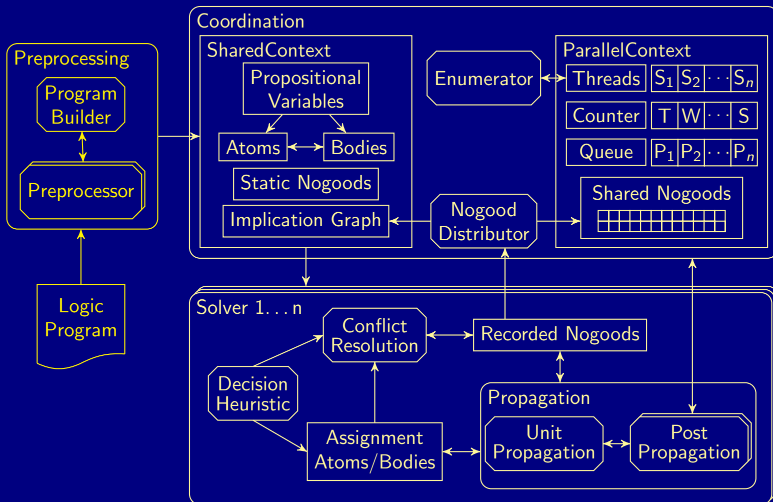
- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Potassco

Multi-threaded architecture of *clasp*

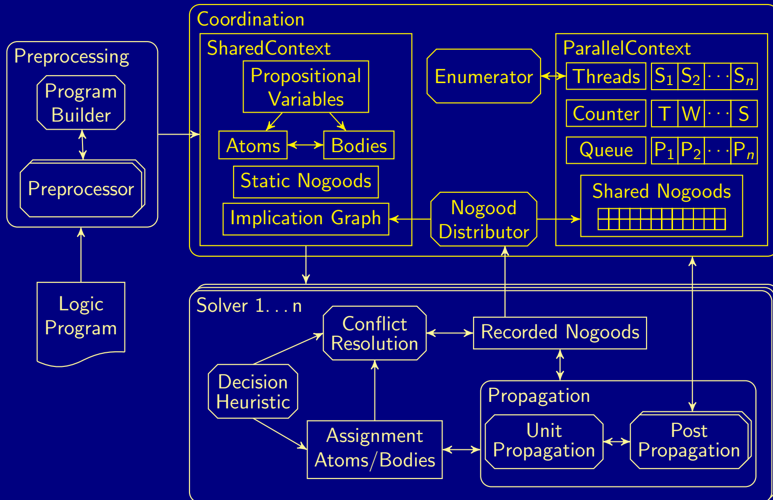


Multi-threaded architecture of *clasp*



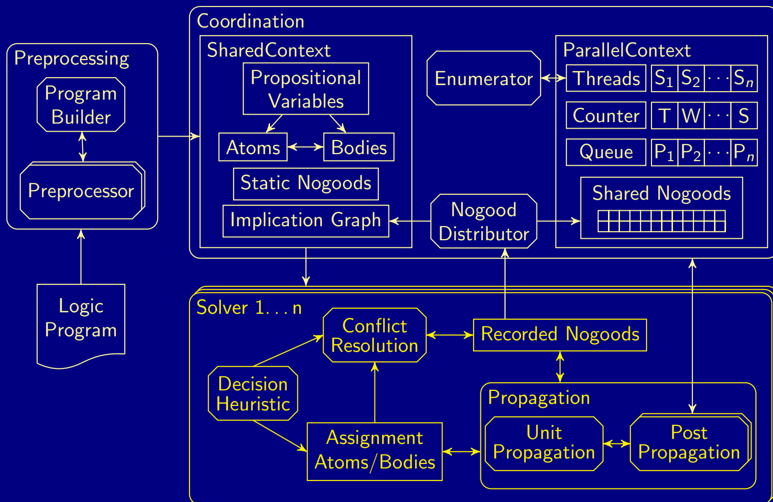
Potassco

Multi-threaded architecture of *clasp*



Potassco

Multi-threaded architecture of *clasp*



Using *clasp*

```
--help[=<n>],-h          : Print 1=basic|2=more|3=full help and exit
```

Using *clasp*

```
--help[=<n>],-h         : Print 1=basic|2=more|3=full help and exit

--configuration=<arg>   : Configure default configuration [frumpy]
    <arg>: frumpy|jumpy|handy|crafty|trendy|chatty
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems
      chatty: Use 4 competing threads initialized via the default portfolio

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n 1..64>[,<mode compete|split>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]
--print-portfolio,-g    : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--configuration=<arg>   : Configure default configuration [frumpy]
    <arg>: frumpy|jumpy|handy|crafty|trendy|chatty
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems
      chatty: Use 4 competing threads initialized via the default portfolio
```

Potassco

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

default

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on advanced n-queens encoding

n	frumpy	jumpy	handy	crafty	trendy	chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Using *clasp*

```
--help[=<n>],-h          : Print 1=basic|2=more|3=full help and exit

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: frumpy|jumpy|handy|crafty|trendy|chatty
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems
      chatty: Use 4 competing threads initialized via the default portfolio

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n 1..64>[,<mode compete|split>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]
--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n 1..64>[,<mode compete|split>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]
```

Potassco

clasp in context

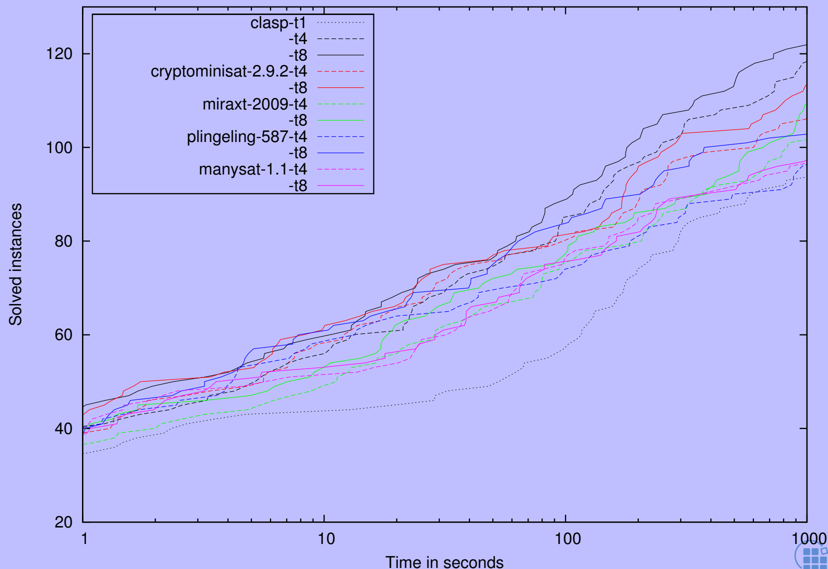
- Compare *clasp* (2.0.5) to the multi-threaded SAT solvers
  - *cryptominisat* (2.9.2)
  - *manysat* (1.1)
  - *miraxt* (2009)
  - *plingeling* (587f)

  all run with four and eight threads in their default settings

- 160/300 benchmarks from crafted category at SAT'11
  - all solvable by *ppfolio* in 1000 seconds
  - crafted SAT benchmarks are closest to ASP benchmarks

Potassco

clasp in context



Potassco

Using *clasp*

```
--help[=<n>],-h          : Print 1=basic|2=more|3=full help and exit

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: frumpy|jumpy|handy|crafty|trendy|chatty
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems
      chatty: Use 4 competing threads initialized via the default portfolio

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n 1..64>[,<mode compete|split>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]
--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--print-portfolio,-g    : Print default portfolio and exit
```

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

Potassco

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

7 Summary

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, {a,h,un}clasp, claspD, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, iClingo, {ros}oClingo, Clingcon
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.sourceforge.net/teaching.html

potassco.sourceforge.net/book.html

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection, bundles tools for ASP developed at the University of Potsdam, for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, {a,h,un}clasp, claspD, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, iClingo, {ros}oClingo, Clingcon
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.sourceforge.net/teaching.html
- Book potassco.sourceforge.net/book.html

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, {a,h,un}clasp, claspD, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, iClingo, {ros}oClingo, Clingcon
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

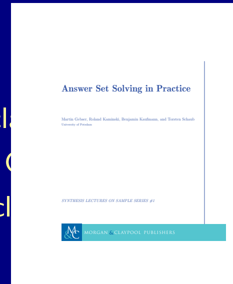
- Benchmark repository `asparagus.cs.uni-potsdam.de`
- Teaching material `potassco.sourceforge.net/teaching.html`
- Book `potassco.sourceforge.net/book.html`

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, {a,h,un}clasp, claspD, claspfolio, cl
- Grounder+Solver Clingo, iClingo, {ros}oClingo, C
- Further Tools aspartame, asp{un}cud, claspre, cl
  insight, metasp, plasp, piclasp, quontroller, etc



- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.sourceforge.net/teaching.html
- Book potassco.sourceforge.net/book.html

Potassco

Outline

1 Motivation

2 Introduction

3 Modeling

4 (Grounding)

5 Solving

6 Potassco

7 Summary

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

$$\textbf{ASP} = \textbf{DB}+\textbf{LP}+\textbf{KR}+\textbf{SAT}$$

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

$$\mathbf{ASP} = \mathbf{DB}+\mathbf{LP}+\mathbf{KR}+\mathbf{SMT}^n$$

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

http://potassco.sourceforge.net

Potassco