Answer Set Programming in a Nutshell

Torsten Schaub

University of Potsdam

Potassco

Outline

Outline

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

Potassco

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

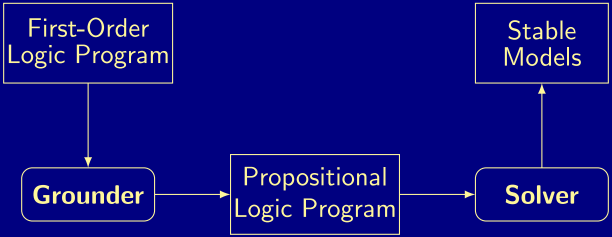
Potassco

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

Potassco

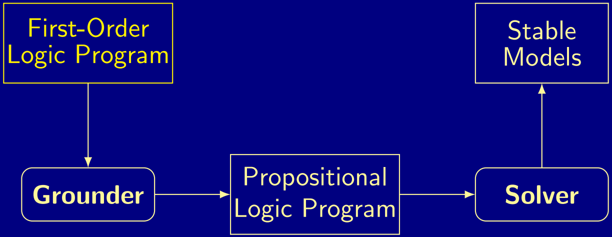
The ASP Solving Process



Expressive modeling language

Powerful grounding and solving tools

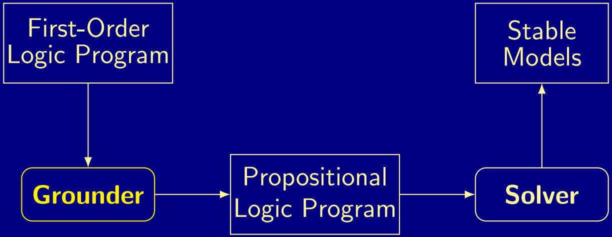
The ASP Solving Process



Expressive modeling language

Powerful grounding and solving tools

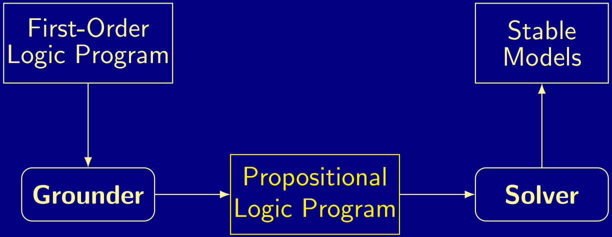
The ASP Solving Process



Expressive modeling language

Powerful grounding and solving tools

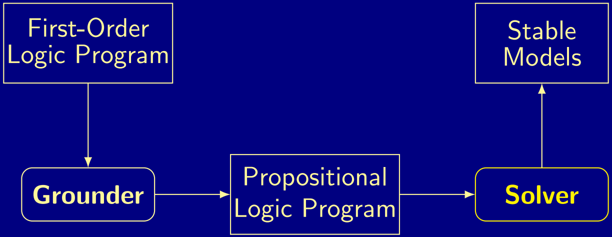
The ASP Solving Process



Expressive modeling language

Powerful grounding and solving tools

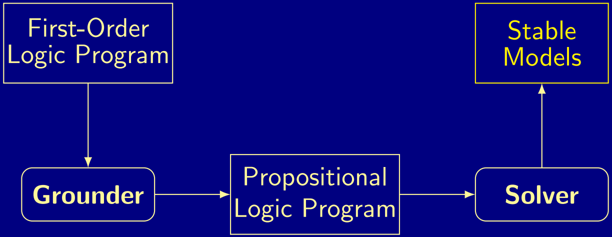
The ASP Solving Process



- Expressive modeling language
- Powerful grounding and solving tools

Potassco

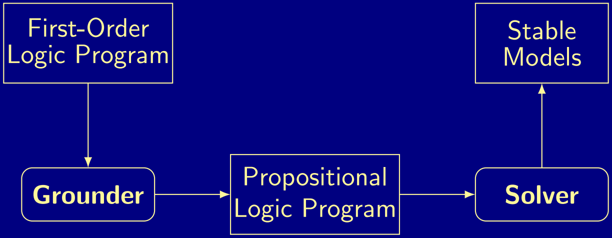
The ASP Solving Process



- Expressive modeling language
- Powerful grounding and solving tools

Potassco

The ASP Solving Process



- Expressive modeling language
- Powerful grounding and solving tools

Outline

Potassco

Propositional Normal Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \sim c_1, \dots, \sim c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\sim$ denote if, and, and default negation
  - intuitive reading: head must be true if body holds
- Semantics given by stable models, informally, sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \sim c_1, \dots, \sim c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\sim$ denote if, and, and default negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally, sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Potassco

Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \sim c_1, \dots, \sim c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\sim$ denote if, and, and default negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

Unsupported atoms

Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$RF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$: (only true atoms shown)

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$RF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \ \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- Unfounded atoms

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- Unfounded atoms

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi) \cup LF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- ~~Unfounded atoms~~

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$CF(\Pi) = \{a \leftrightarrow (\bigvee_{(a \leftarrow B) \in \Pi} BF(B)) \mid a \in atom(\Pi)\}$

$BF(B) = \bigwedge_{b \in B \cap atom(\Pi)} b \wedge \bigwedge_{\sim c \in B} \neg c$

$LF(\Pi) = \{(\bigvee_{a \in L} a) \rightarrow (\bigvee_{a \in L, (a \leftarrow B) \in \Pi, B \cap L = \emptyset} BF(B)) \mid L \in loop(\Pi)\}$

Classical models of $CF(\Pi) \cup LF(\Pi)$:

Theorem (Lin and Zhao)

Let Π be a normal logic program and $X \subseteq atom(\Pi)$.
Then, X is a stable model of Π iff $X \models CF(\Pi) \cup LF(\Pi)$.

- Size of $CF(\Pi)$ is linear in the size of Π
- Size of $LF(\Pi)$ may be exponential in the size of Π

Potassco

Let's run it!

```
$ cat prg.lp

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

$ clingo 0 prg.lp

clingo version 4.5.0
Reading from prg.lp
Solving...
Answer: 1
a x
Answer: 2
b
SATISFIABLE

Models      : 2
Calls       : 1
Time        : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

Let's run it!

```
$ cat prg.lp

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

$ clingo 0 prg.lp

clingo version 4.5.0
Reading from prg.lp
Solving...
Answer: 1
a x
Answer: 2
b
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Let's run it!

```
$ cat prg.lp

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

$ clingo 0 prg.lp

clingo version 4.5.0
Reading from prg.lp
Solving...
Answer: 1
a x
Answer: 2
b
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Let's run it!

```
$ cat prg.lp

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

$ clingo 0 prg.lp

clingo version 4.5.0
Reading from prg.lp
Solving...
Answer: 1
a x
Answer: 2
b
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Let's run it!

```
$ cat prg.lp

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

$ clingo 0 prg.lp

clingo version 4.5.0
Reading from prg.lp
Solving...
Answer: 1
a x
Answer: 2
b
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Genuine Stable Models Semantics

- The reduct ϕ^X of a formula ϕ relative to a set X of atoms is defined as follows

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\phi^X = \top & \text{if } X \not\models \psi \text{ and } \phi = {\sim}\psi
\end{array}
$$

Let Φ be a formula and $X \subseteq \mathit{atom}(\Phi)$.
Then, X is a stable model of Φ if X is a $\subseteq$-minimal model of Φ^X

a and ${\sim}{\sim}a$ are not the same

Potassco

Genuine Stable Models Semantics

- The reduct ϕ^X of a formula ϕ relative to a set X of atoms is defined as follows

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\phi^X = \top & \text{if } X \not\models \psi \text{ and } \phi = \sim\psi
\end{array}
$$

Definition (Gelfond and Lifschitz et al.)

Let Φ be a formula and $X \subseteq \mathit{atom}(\Phi)$.
Then, X is a stable model of Φ if X is a $\subseteq$-minimal model of Φ^X

a and $\sim\sim a$ are not the same

Genuine Stable Models Semantics

- The reduct ϕ^X of a formula ϕ relative to a set X of atoms is defined as follows

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\phi^X = \top & \text{if } X \not\models \psi \text{ and } \phi = {\sim}\psi
\end{array}
$$

Definition (Gelfond and Lifschitz et al.)

Let Φ be a formula and $X \subseteq atom(\Phi)$.
Then, X is a stable model of Φ if X is a $\subseteq$-minimal model of Φ^X

- Note a and ${\sim}{\sim}a$ are not the same

Genuine Stable Models Semantics

- The reduct ϕ^X of a formula ϕ relative to a set X of atoms is defined as follows

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\phi^X = \top & \text{if } X \not\models \psi \text{ and } \phi = {\sim}\psi
\end{array}
$$

Definition (Gelfond and Lifschitz et al.)

Let Φ be a formula and $X \subseteq \mathit{atom}(\Phi)$.
Then, X is a stable model of Φ if X is a $\subseteq$-minimal model of Φ^X

- Note a and ${\sim}{\sim}a$ are not the same

Genuine Stable Models Semantics

- The reduct ϕ^X of a formula ϕ relative to a set X of atoms is defined as follows

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\phi^X = \top & \text{if } X \not\models \psi \text{ and } \phi = {\sim}\psi
\end{array}
$$

Definition (Gelfond and Lifschitz et al.)

Let Φ be a formula and $X \subseteq atom(\Phi)$.
Then, X is a stable model of Φ if X is a $\subseteq$-minimal model of Φ^X

- Note a and ${\sim}{\sim}a$ are not the same

Outline

Potassco

Some language constructs

- Variables
  - `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 #sum { X : p(X,Y), q(Y) } 7`

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)

Peanutshell

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

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)

Peanutshell

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

Satisfiability testing

$$(a \leftrightarrow b) \wedge c$$

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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

Maximum satisfiability testing

"$(a \leftrightarrow b) \wedge c$"

```
{ a ; b ; c }.

 :- not a, b.
 :~ a, not b. [10@2]
 :~ not c.    [100@1]
```

n-queens

Basic encoding

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

 :- { 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.
```

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 (no instance)

```
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) }.
```

Company Controls

```
controls(X,Y) :-
        #sum+ { S: owns(X,Y,S);
                S,Z: controls(X,Z), owns(Z,Y,S) } > 50,
        company(X), company(Y), X != Y.
```

```
company(c_1).   owns(c_1,c_2,60).
                owns(c_1,c_3,20).
company(c_2).   owns(c_2,c_3,35).
company(c_3).   owns(c_3,c_4,51).
company(c_4).
```

Outline

Towards Conflict-Driven ASP

- Goal Conflict-driven approach to ASP solving
- Idea View inferences as unit propagation on nogoods

- Background
  - A nogood expresses an inadmissible assignment
  - For example, given a rule $a \leftarrow b$
    - $\{\mathbf{F}a, \mathbf{T}b\}$ is a nogood (stands for $\{a \mapsto \mathbf{F}, b \mapsto \mathbf{T}\}$)
    - Unit propagation on $\{\mathbf{F}a, \mathbf{T}b\}$ infers
      - $\mathbf{T}a$ wrt assignment containing $\mathbf{T}b$
      - $\mathbf{F}b$ wrt assignment containing $\mathbf{F}a$

Towards Conflict-Driven ASP

- Goal Conflict-driven approach to ASP solving
- Idea View inferences as unit propagation on nogoods

- Background
  - A nogood expresses an inadmissible assignment
  - For example, given a rule $a \leftarrow b$
    - $\{\mathbf{F}a, \mathbf{T}b\}$ is a nogood (stands for $\{a \mapsto \mathbf{F}, b \mapsto \mathbf{T}\}$)
    - Unit propagation on $\{\mathbf{F}a, \mathbf{T}b\}$ infers
      - $\mathbf{T}a$ wrt assignment containing $\mathbf{T}b$
      - $\mathbf{F}b$ wrt assignment containing $\mathbf{F}a$

Towards Conflict-Driven ASP

- Goal Conflict-driven approach to ASP solving
- Idea View inferences as unit propagation on nogoods

- Background
  - A nogood expresses an inadmissible assignment
  - For example, given a rule $a \leftarrow b$
    - $\{\mathbf{F}a, \mathbf{T}b\}$ is a nogood (stands for $\{a \mapsto \mathbf{F}, b \mapsto \mathbf{T}\}$)
    - Unit propagation on $\{\mathbf{F}a, \mathbf{T}b\}$ infers
      $\mathbf{T}a$ wrt assignment containing $\mathbf{T}b$
      $\mathbf{F}b$ wrt assignment containing $\mathbf{F}a$

Potassco

Towards Conflict-Driven ASP

- Goal Conflict-driven approach to ASP solving
- Idea View inferences as unit propagation on nogoods

- Background
  - A nogood expresses an inadmissible assignment
  - For example, given a rule $a \leftarrow b$
    - $\{\mathbf{F}a, \mathbf{T}b\}$ is a nogood (stands for $\{a \mapsto \mathbf{F}, b \mapsto \mathbf{T}\}$)
    - Unit propagation on $\{\mathbf{F}a, \mathbf{T}b\}$ infers
      $\mathbf{T}a$ wrt assignment containing $\mathbf{T}b$
      $\mathbf{F}b$ wrt assignment containing $\mathbf{F}a$

Towards Conflict-Driven ASP

- Goal Conflict-driven approach to ASP solving
- Idea View inferences as unit propagation on nogoods

- Background
  - A nogood expresses an inadmissible assignment
  - For example, given a rule $a \leftarrow b$
    - $\{\mathbf{F}a, \mathbf{T}b\}$ is a nogood (stands for $\{a \mapsto \mathbf{F}, b \mapsto \mathbf{T}\}$)
    - Unit propagation on $\{\mathbf{F}a, \mathbf{T}b\}$ infers
      $\mathbf{T}a$ wrt assignment containing $\mathbf{T}b$
      $\mathbf{F}b$ wrt assignment containing $\mathbf{F}a$

Nogoods from logic programs

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \qquad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad c \leftrightarrow \bot \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\begin{aligned}
\Delta_\Pi &= \{\dots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \dots\} \\
&\cup \{\dots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \dots\} \\
&\cup \{\dots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \dots\} \\
&\cup \{\dots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \dots\} \\
\Lambda_\Pi &= \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}
\end{aligned}$$

Size of Δ_Π is linear in the size of Π

Size of Λ_Π is (in general) exponential in the size of Π

Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \quad y \leftrightarrow B_5\}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\Delta_\Pi = \{\dots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \dots\}$$
$$\cup \{\dots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \dots\}$$
$$\cup \{\dots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \dots\}$$
$$\cup \{\dots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \dots\}$$
$$\Lambda_\Pi = \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \qquad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \qquad y \leftrightarrow B_5 \quad \}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\Delta_\Pi = \{\dots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \dots\}$$
$$\cup \{\dots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \dots\}$$
$$\cup \{\dots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \dots\}$$
$$\cup \{\dots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \dots\}$$

$$\Lambda_\Pi = \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \qquad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \qquad y \leftrightarrow B_5 \quad \}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\begin{aligned}
\Delta_\Pi &= \{\dots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \dots\} \\
&\cup \{\dots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \dots\} \\
&\cup \{\dots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \dots\} \\
&\cup \{\dots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \dots\} \\
\Lambda_\Pi &= \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}
\end{aligned}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \qquad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \qquad y \leftrightarrow B_5 \quad \}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\begin{aligned}
\Delta_\Pi &= \{\ldots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \ldots\} \\
&\cup \{\ldots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \ldots\} \\
&\cup \{\ldots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \ldots\} \\
&\cup \{\ldots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \ldots\} \\
\Lambda_\Pi &= \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}
\end{aligned}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \qquad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \qquad y \leftrightarrow B_5 \quad\}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\begin{aligned}
\Delta_\Pi &= \{\ldots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \ldots\} \\
&\cup \{\ldots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \ldots\} \\
&\cup \{\ldots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \ldots\} \\
&\cup \{\ldots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \ldots\} \\
\Lambda_\Pi &= \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}
\end{aligned}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Nogoods from logic programs

$$\Pi = \{a \leftarrow {\sim} b \quad b \leftarrow {\sim} a \quad x \leftarrow a, {\sim} c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \vee B_4 \quad y \leftrightarrow B_5\}$$
$$\cup \{B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \wedge \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \wedge b\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow B_3\}$$

Nogoods for $CF(\Pi)$ and $LF(\Pi)$

$$\begin{aligned}
\Delta_\Pi &= \{\dots, \{\mathbf{F}x, \mathbf{T}B_3\}, \{\mathbf{F}x, \mathbf{T}B_4\} \dots\} \\
&\cup \{\dots, \{\mathbf{T}x, \mathbf{F}B_3, \mathbf{F}B_4\}, \dots\} \\
&\cup \{\dots, \{\mathbf{F}B_3, \mathbf{T}a, \mathbf{F}c\}, \dots\} \\
&\cup \{\dots, \{\mathbf{T}B_3, \mathbf{F}a\}, \{\mathbf{T}B_3, \mathbf{T}c\}, \dots\} \\
\Lambda_\Pi &= \{\{\mathbf{T}x, \mathbf{F}B_3\}, \{\mathbf{T}y, \mathbf{F}B_3\}\}
\end{aligned}$$

- Size of Δ_Π is linear in the size of Π
- Size of Λ_Π is (in general) exponential in the size of Π
  - Satisfaction of Λ_Π can be tested in linear time

Stable Models as Solutions

Theorem

Let Π be a normal logic program and $X \subseteq atom(\Pi)$.
Then, X is a stable model of Π iff $X = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ for a (unique) solution $\mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$.[1]

Advantages

- Stable model computation as Boolean constraint solving
- All inferences can be seen as unit propagation on nogoods
- Nogoods readily available as conflict reasons

[1] A total assignment $\mathbf{A}$ is a solution for $\Delta_\Pi \cup \Lambda_\Pi$ if $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta_\Pi \cup \Lambda_\Pi$.

Stable Models as Solutions

Theorem

Let Π be a normal logic program and $X \subseteq atom(\Pi)$.
Then, X is a stable model of Π iff $X = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$
for a (unique) solution $\mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$.[1]

Advantages

- Stable model computation as Boolean constraint solving
- All inferences can be seen as unit propagation on nogoods
- Nogoods readily available as conflict reasons

[1] A total assignment $\mathbf{A}$ is a solution for $\Delta_\Pi \cup \Lambda_\Pi$ if $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta_\Pi \cup \Lambda_\Pi$.

Conflict-Driven Constraint Learning (CDCL)

loop
    propagate // assign deterministic consequences

    if no conflict **then**
        if all variables assigned **then return** variable assignment
        else *decide* // non-deterministically assign some variable

    else
        if top-level conflict **then return** unsatisfiable
        else
            analyze // analyze conflict and add conflict constraint
            backjump // undo assignments violating conflict constraint

Conflict-Driven Constraint Learning (CDCL)

```
loop
    propagate                              // assign deterministic consequences
    if no conflict then
        if all variables assigned then return variable assignment
        else decide          // non-deterministically assign some variable
    else
        if top-level conflict then return unsatisfiable
        else
            analyze         // analyze conflict and add conflict constraint
            backjump     // undo assignments violating conflict constraint
```

The solver clasp

- Beyond deciding (stable) model existence, clasp allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- clasp allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- clasp pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

Potassco

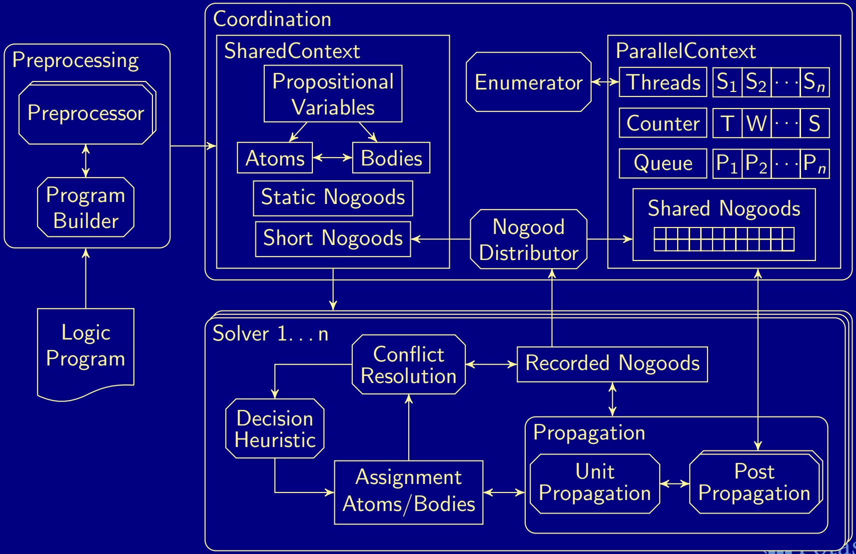
The solver clasp

- Beyond deciding (stable) model existence, clasp allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- clasp allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- clasp pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

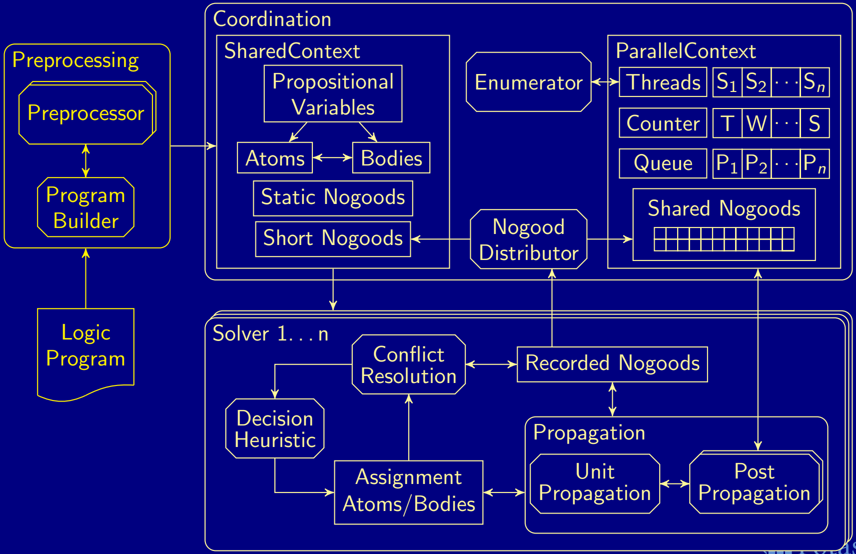
The solver clasp

- Beyond deciding (stable) model existence, clasp allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- clasp allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- clasp pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

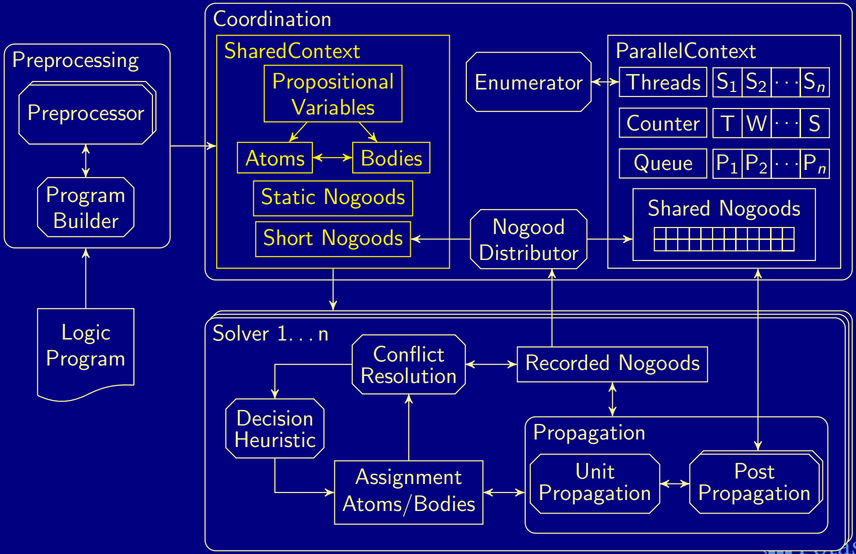
Multi-threaded architecture of clasp



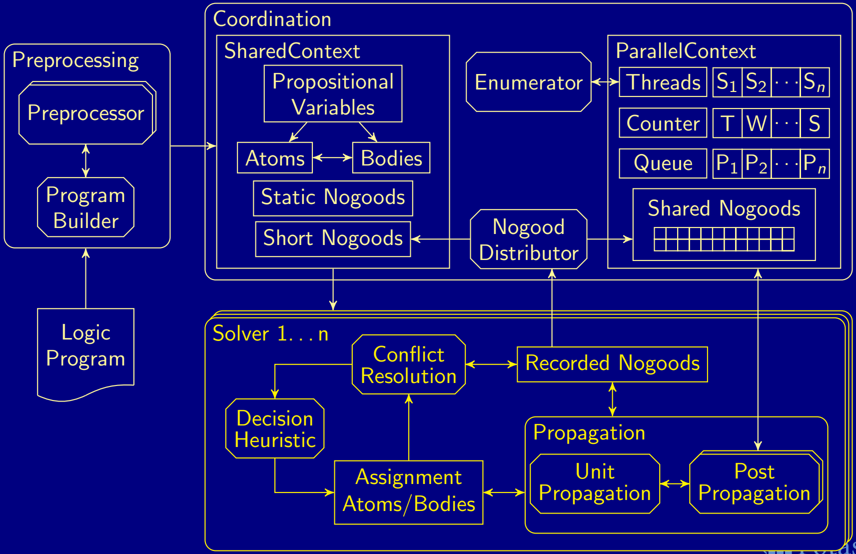
Multi-threaded architecture of clasp



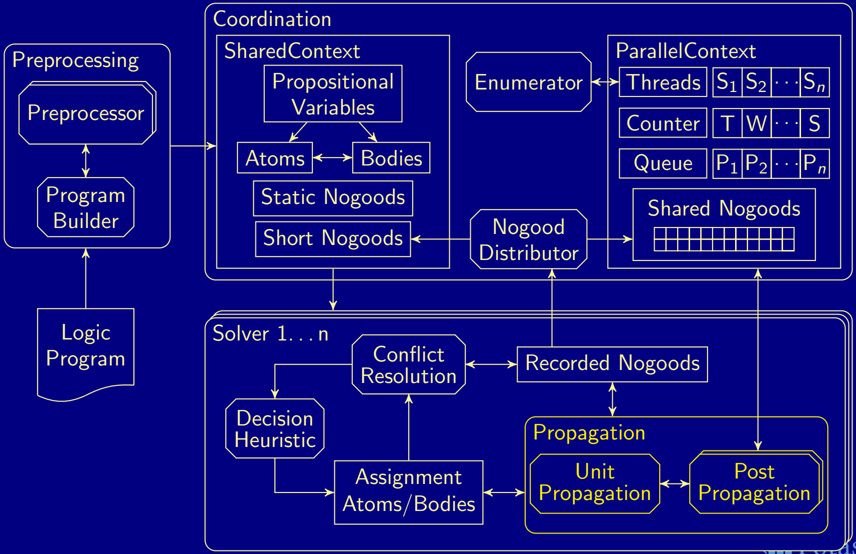
Multi-threaded architecture of clasp



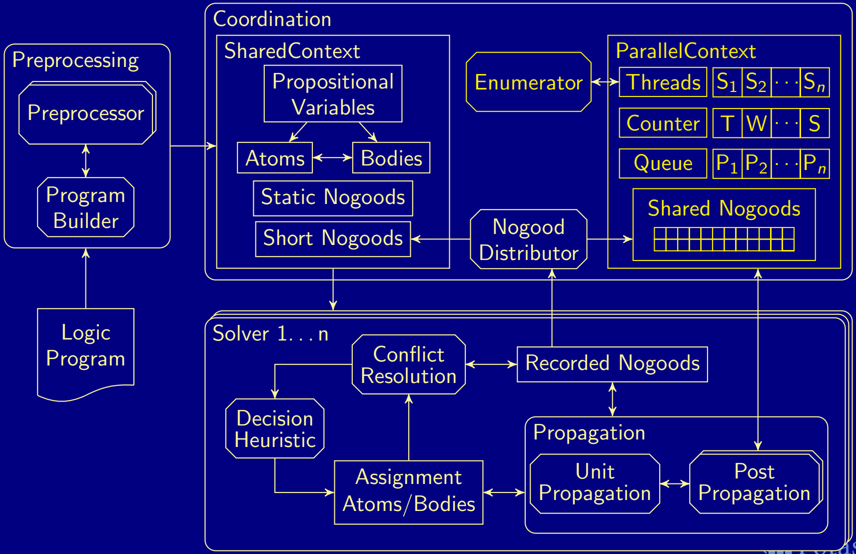
Multi-threaded architecture of clasp

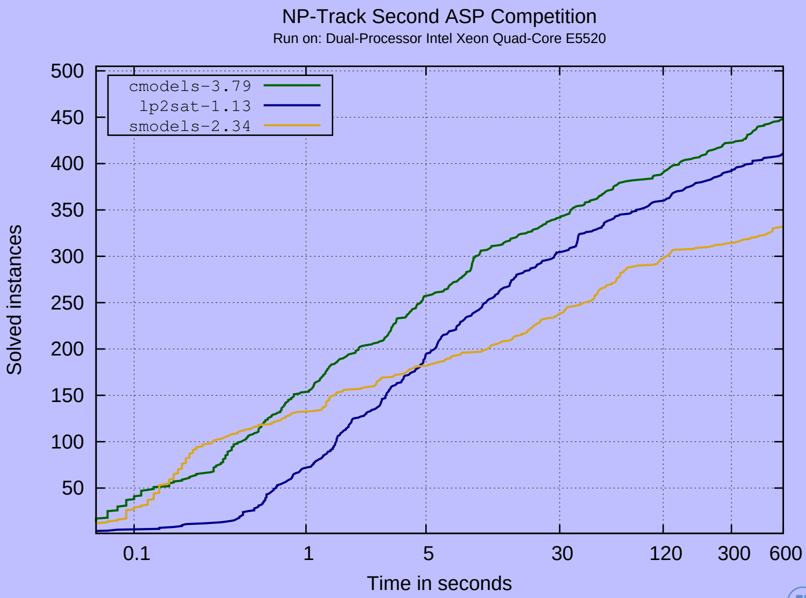


Multi-threaded architecture of clasp

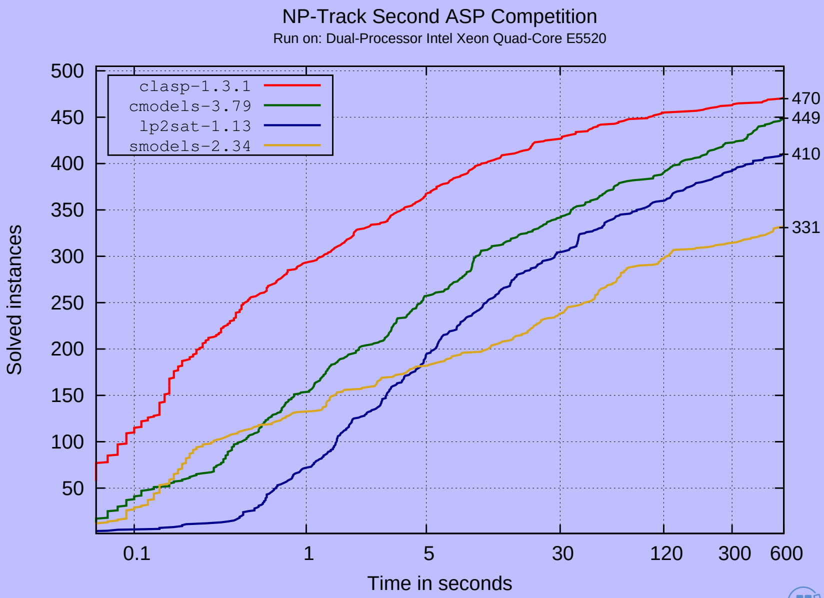


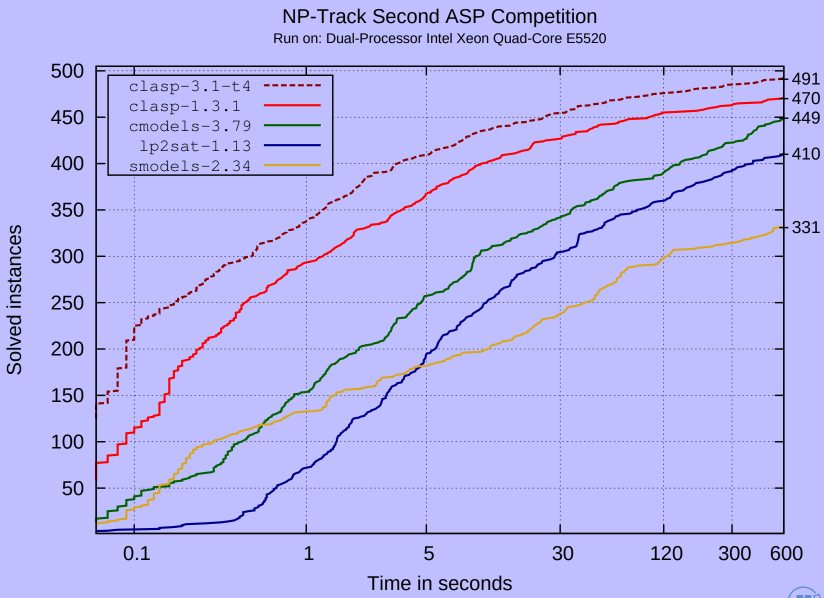
Multi-threaded architecture of clasp




NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520
cmodels-3.79
lp2sat-1.13
smodels-2.34
Solved instances
500
450
400
350
300
250
200
150
100
50
0.1
1
5
30
120
300
600
Time in seconds

Potassco


NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520
clasp-1.3.1
cmodels-3.79
lp2sat-1.13
smodels-2.34
Solved instances
500
450
400
350
300
250
200
150
100
50
470
449
410
331
0.1
1
5
30
120
300
600
Time in seconds


NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520
clasp-3.1-t4
clasp-1.3.1
cmodels-3.79
lp2sat-1.13
smodels-2.34
Solved instances
500
450
400
350
300
250
200
150
100
50
491
470
449
410
331
0.1
1
5
30
120
300
600
Time in seconds

Potassco

Outline

potassco.sourceforge.net

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

- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

asparagus.cs.uni-potsdam.de

potassco.sourceforge.net

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

- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmark repository asparagus.cs.uni-potsdam.de

potassco.sourceforge.net

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

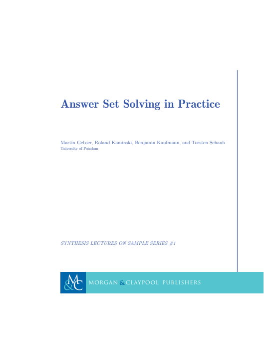
- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmark repository `asparagus.cs.uni-potsdam.de`

potassco.sourceforge.net

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

lingo
spfolio
Clin Clingo
parta chasp
asp,
tory -po

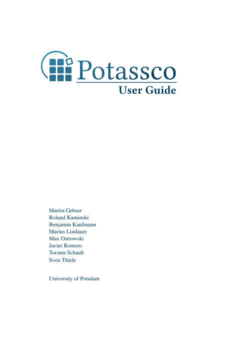


Answer Set Solving in Practice

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub
University of Potsdam

SYNTHESIS LECTURES ON SAMPLE SERIES #1

MORGAN & CLAYPOOL PUBLISHERS



Potassco
User Guide

Martin Gebser
Roland Kaminski
Benjamin Kaufmann
Marius Lindauer
Max Ostrowski
Javier Romero
Torsten Schaub
Sven Thiele

University of Potsdam

Under consideration for publication in Theory and Practice of Logic Programming 1

arXiv:1507.06576v1 [cs.PL] 23 Jul 2015

Abstract Gringo

MARTIN GEBSER*
Aalto University, HIIT, Finland
University of Potsdam, Germany
gebser@cs.uni-potsdam.de

AMELIA HARRISON†
Univeristy of Texas at Austin, USA
ameliaj@cs.utexas.edu

ROLAND KAMINSKI*
University of Potsdam, Germany
kaminski@cs.uni-potsdam.de

VLADIMIR LIFSCHITZ†
Univeristy of Texas at Austin, USA
vl@cs.utexas.edu

TORSTEN SCHAUB*‡
University of Potsdam, Germany
INRIA Rennes, France
torsten@cs.uni-potsdam.de

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

This paper defines the syntax and semantics of the input language of the ASP grounder GRINGO. The definition covers several constructs that were not discussed in earlier work on the semantics of that language, including intervals, pools, division of integers, aggregates with non-numeric values, and lparse-style aggregate expressions. The definition is abstract in the sense that it disregards some details related to representing programs by strings of ASCII characters. It serves as a specification for GRINGO from Version 4.5 on.

To appear in Theory and Practice of Logic Programming (TPLP), Proceedings of ICLP 2015.

* Supported by AoF (grant 251170) and DFG (grants SCHA 550/8 and 550/9).
† Partially supported by the National Science Foundation under Grant IIS-1422455.
‡ Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

Outline

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

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

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

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