Answer Set Programming in a Nutshell

Torsten Schaub

University of Potsdam

Potassco
Outline

1 Introduction
2 Foundations
3 Modeling
4 Algorithms and Systems
5 Potassco
6 Summary
Outline

1. Introduction
2. Foundations
3. Modeling
4. Algorithms and Systems
5. Potassco
6. Summary
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
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
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
The ASP Solving Process

Expressive modeling language
Powerful grounding and solving tools
The ASP Solving Process

First-Order Logic Program

Grounder

Propositional Logic Program

Solver

Stable Models

Expressive modeling language
Powerful grounding and solving tools
The ASP Solving Process

- First-Order Logic Program
- **Grounder**
- Propositional Logic Program
- **Solver**
- Stable Models

Expressive modeling language
Powerful grounding and solving tools
The ASP Solving Process

First-Order Logic Program

Grounder

Propositional Logic Program

Solver

Stable Models

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
Outline

1. Introduction
2. Foundations
3. Modeling
4. Algorithms and Systems
5. Potassco
6. Summary
Propositional Normal Logic Programs

A logic program $\Pi$ is a set of rules of the form

$$ a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n $$

- $a$ and all $b_i, c_j$ are atoms (propositional variables)
- $\leftarrow, ,, \neg$ 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 $\Pi$ and
- each atom in $X$ is justified by some rule in $\Pi$
A logic program $\Pi$ is a set of rules of the form

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

- $a$ 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 $\Pi$ and
- each atom in $X$ is justified by some rule in $\Pi$
A logic program $\Pi$ is a set of rules of the form

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

- $a$ 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 $\Pi$ and
- Each atom in $X$ is justified by some rule in $\Pi$
Logic Programs as Propositional Formulas

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[ CF(\Pi) = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \land \neg c) \lor y \quad y \leftarrow x \land b \} \]
\[ \quad \cup \{ c \leftrightarrow \bot \} \]

\[ LF(\Pi) = \{(x \lor y) \rightarrow a \land \neg c \} \]

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

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

\[ RF(\Pi) = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \land \neg c) \lor y \quad y \leftarrow x \land b \} \]

\[ LF(\Pi) = \{(x \lor y) \rightarrow a \land \neg c\} \]

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

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[ \text{RF}(\Pi) = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \land \neg c) \lor y \quad y \leftarrow x \land b \} \]

\[ \cup \{ c \leftrightarrow \bot \} \]

\[ \text{LF}(\Pi) = \{(x \lor y) \rightarrow a \land \neg c\} \]

Classical models of \( \text{RF}(\Pi) \):

\{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

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

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

$$LF(\Pi) = \{(x \lor y) \rightarrow a \land \neg c\}$$

Classical models of $RF(\Pi)$:

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

- Unsupported atoms
- Unfounded atoms
Logic Programs as Propositional Formulas

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

$$CF(\Pi) = \{ a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \land \neg c) \lor y \quad y \leftrightarrow x \land b \}$$

$$ LF(\Pi) = \{(x \lor y) \rightarrow a \land \neg c\} $$

Classical models of $CF(\Pi)$:

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

- Unsupported atoms
- Unfounded atoms
Logic Programs as Propositional Formulas

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[ CF(\Pi) = \{ a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \land \neg c) \lor y \quad y \leftrightarrow x \land b \} \]

\[ LF(\Pi) = \{ (x \lor y) \rightarrow a \land \neg c \} \]

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

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[
CF(\Pi) = \{ a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \land \neg c) \lor y \quad y \leftrightarrow x \land b \}
\]

\[
\cup \{ c \leftrightarrow \bot \}
\]

\[
LF(\Pi) = \{ (x \lor y) \rightarrow a \land \neg c \}
\]

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
Logic Programs as Propositional Formulas

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

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

\[ BF(B) = \bigwedge_{b \in B \cap atom(\Pi)} b \land \bigwedge_{\sim c \in B} \neg c \]

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

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

Theorem (Lin and Zhao)

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

- Size of \( CF(\Pi) \) is linear in the size of \( \Pi \)
- Size of \( LF(\Pi) \) may be exponential in the size of \( \Pi \)
Let's run it!

```bash
$ 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 $\phi^X$ of a formula $\phi$ relative to a set $X$ of atoms is defined as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \mu^X)$ if $X \models \phi$ and $\phi = (\psi \circ \mu)$ for $\circ \in \{\land, \lor, \rightarrow\}$
- $\phi^X = \top$ if $X \not\models \psi$ and $\phi = \neg \psi$

Definition (Gelfond and Lifschitz et al.)

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

Note: $a$ and $\neg \neg a$ are not the same
Genuine Stable Models Semantics

The reduct $\phi^X$ of a formula $\phi$ relative to a set $X$ of atoms is defined as follows:

- $\phi^X = \perp$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \mu^X)$ if $X \models \phi$ and $\phi = (\psi \circ \mu)$ for $\circ \in \{\land, \lor, \rightarrow\}$
- $\phi^X = \top$ if $X \not\models \psi$ and $\phi = \sim \psi$

Definition (Gelfond and Lifschitz et al.)

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

Note: $\sim \sim a$ and $\sim \sim a$ are not the same
Genuine Stable Models Semantics

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

$$
\begin{align*}
\phi^X &= \bot & \text{if } X \not\models \phi \\
\phi^X &= \phi & \text{if } \phi \in X \\
\phi^X &= (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\land, \lor, \to\} \\
\phi^X &= \top & \text{if } X \not\models \psi \text{ and } \phi = \neg \psi
\end{align*}
$$

Definition (Gelfond and Lifschitz et al.)

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

- Note: $a$ and $\neg\neg a$ are not the same
The reduct $\phi^X$ of a formula $\phi$ relative to a set $X$ of atoms is defined as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \mu^X)$ if $X \models \phi$ and $\phi = (\psi \circ \mu)$ for $\circ \in \{\land, \lor, \rightarrow\}$
- $\phi^X = \top$ if $X \not\models \psi$ and $\phi = \sim \psi$

**Definition (Gelfond and Lifschitz et al.)**

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

**Note**  
$a$ and $\sim \sim a$ are not the same
The reduct $\phi^X$ of a formula $\phi$ relative to a set $X$ of atoms is defined as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \mu^X)$ if $X \models \phi$ and $\phi = (\psi \circ \mu)$ for $\circ \in \{\land, \lor, \rightarrow\}$
- $\phi^X = \top$ if $X \not\models \psi$ and $\phi = \neg \psi$

Definition (Gelfond and Lifschitz et al.)

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

Note: $a$ and $\neg \neg a$ are not the same.
1. Introduction
2. Foundations
3. Modeling
4. Algorithms and Systems
5. Potassco
6. Summary
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**
  - \( s(Y) :- r(Y), 2 \sum \{ X : p(X,Y), q(Y) \} 7 \)
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

\[(a \leftrightarrow b) \land c\]
Satisfiability testing

\((a \leftrightarrow b) \land c\)

\{ a ; b ; c \}.

:- not a, b.
:- a, not b.
:- not c.
Maximum satisfiability testing

\[ (a \Leftrightarrow b) + (a \leftrightarrow b) \land c \]

\{ a ; b ; c \}.

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

\sim a, b. [42@1]
\sim not a, not b. [69@2]
Modeling

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

\[
\{ \text{queen}(I,1\ldots n) \} = 1 :- I = 1\ldots n.
\]
\[
\{ \text{queen}(1\ldots n,J) \} = 1 :- J = 1\ldots n.
\]
\[
:- \{ \text{queen}(D-J,J) \} \geq 2, D = 2\ldots 2n.
\]
\[
:- \{ \text{queen}(D+J,J) \} \geq 2, D = 1-n\ldots n-1.
\]
n-queens

(Experimental) constraint encoding

1 $\leq$ $\text{queen}(1..n) \leq n$.

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

1. Introduction
2. Foundations
3. Modeling
4. Algorithms and Systems
5. Potassco
6. Summary
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 $a \leftarrow b$
  
  $\{F_a, T_b\}$ is a nogood (stands for $\{a \mapsto F, b \mapsto T\}$)

  Unit propagation on $\{F_a, T_b\}$ infers
  - $T_a$ wrt assignment containing $T_b$
  - $F_b$ wrt assignment containing $F_a$
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 $a \leftarrow b$
  - $\{F_a, T_b\}$ is a nogood (stands for $\{a \mapsto F, b \mapsto T\}$)
  - Unit propagation on $\{F_a, T_b\}$ infers
    - $T_a$ wrt assignment containing $T_b$
    - $F_b$ wrt assignment containing $F_a$
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 $a \leftarrow b$
  - \{F_a, T_b\} is a nogood (stands for \{a \mapsto F, b \mapsto T\})
  - Unit propagation on \{F_a, T_b\} infers
    - $T_a$ wrt assignment containing $T_b$
    - $F_b$ wrt assignment containing $F_a$
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 $a \leftarrow b$
  - $\{F_a, T_b\}$ is a **nogood** (stands for $\{a \mapsto F, b \mapsto T\}$)
  - Unit propagation on $\{F_a, T_b\}$ infers
    - $T_a$ wrt assignment containing $T_b$
    - $F_b$ wrt assignment containing $F_a$
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 $a \leftarrow b$
  - $\{F_a, T_b\}$ is a nogood (stands for $\{a \mapsto F, b \mapsto T\}$)
  - **Unit propagation** on $\{F_a, T_b\}$ infers
    - $T_a$ wrt assignment containing $T_b$
    - $F_b$ wrt assignment containing $F_a$
Nogoods from logic programs

\[ \Pi = \{ a \leftarrow \neg b \ b \leftarrow \neg a \ x \leftarrow a, \neg c \ x \leftarrow y \ y \leftarrow x, b \} \]

\[ CF(\Pi) = \{ a \leftrightarrow \neg b \ b \leftrightarrow \neg a \ c \leftrightarrow \bot \ x \leftrightarrow (a \land \neg c) \lor y \ y \leftrightarrow x \land b \} \]

\[ LF(\Pi) = \{ (x \lor y) \rightarrow a \land \neg c \} \]

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

\[ \Delta_\Pi = \{ \ldots, \{ Fx, TB_3 \}, \{ Fx, TB_4 \} \ldots \} \]
\[ \cup \{ \ldots, \{ Tx, FB_3, FB_4 \}, \ldots \} \]
\[ \cup \{ \ldots, \{ FB_3, Ta,Fc \}, \ldots \} \]
\[ \cup \{ \ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \} \ldots \} \]

\[ \Lambda_\Pi = \{ \{ Tx, FB_3 \}, \{ Ty, FB_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

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

\[ LF(\Pi) = \{ (x \lor y) \rightarrow B_3 \} \]

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

\[ \Delta_\Pi = \{ \ldots, \{ Fx, TB_3 \}, \{ Fx, TB_4 \} \ldots \} \]
\[ \cup \{ \ldots, \{ Tx, FB_3, FB_4 \}, \ldots \} \]
\[ \cup \{ \ldots, \{ FB_3, Ta, Fc \}, \ldots \} \]
\[ \cup \{ \ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \}, \ldots \} \]

\[ \Lambda_\Pi = \{ \{ Tx, FB_3 \}, \{ Ty, FB_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

\[
\Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} 
\]

\[
CF(\Pi) = \{ a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \lor B_4 \quad y \leftrightarrow B_5 \} 
\]

\[
\cup \{ B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \land \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \land b \} 
\]

\[
LF(\Pi) = \{(x \lor y) \rightarrow B_3\} 
\]

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

\[
\Delta_\Pi = \{ \ldots, \{ Fx, TB_3 \}, \{ Fx, TB_4 \} \ldots \} 
\]

\[
\cup \{ \ldots, \{ Tx, FB_3, FB_4 \} \ldots \} 
\]

\[
\cup \{ \ldots, \{ FB_3, Ta,Fc \} \ldots \} 
\]

\[
\cup \{ \ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \} \ldots \} 
\]

\[
\Lambda_\Pi = \{ \{ Tx, FB_3 \}, \{ Ty, FB_3 \} \} 
\]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

\[ \Pi = \{ \begin{array}{l}
    a \leftarrow \sim b \\
    b \leftarrow \sim a \\
    x \leftarrow a, \sim c \\
    x \leftarrow y \\
    y \leftarrow x, b
\end{array} \} \]

\[ CF(\Pi) = \{ \begin{array}{l}
    a \leftrightarrow B_1 \\
    b \leftrightarrow B_2 \\
    c \leftrightarrow \bot \\
    x \leftrightarrow B_3 \lor B_4 \\
    y \leftrightarrow B_5
\end{array} \} \]

\[ LF(\Pi) = \{ (x \lor y) \rightarrow B_3 \} \]

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

\[ \Delta_\Pi = \{ \ldots, \{ Fx, TB_3 \}, \{ Fx, TB_4 \} \ldots \} \]
\[ \cup \{ \ldots, \{ Tx, FB_3, FB_4 \}, \ldots \} \]
\[ \cup \{ \ldots, \{ FB_3, Ta, Fc \}, \ldots \} \]
\[ \cup \{ \ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \}, \ldots \} \]

\[ \Lambda_\Pi = \{ \{ Tx, FB_3 \}, \{ Ty, FB_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[ CF(\Pi) = \{ a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \lor B_4 \quad y \leftrightarrow B_5 \} \]
\[ \cup \{ B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \land \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \land b \} \]

\[ LF(\Pi) = \{(x \lor y) \rightarrow B_3\} \]

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

\[ \Delta_\Pi = \{ \ldots, \{ \text{T} \text{x}, \text{T} B_3 \}, \{ \text{F} \text{x}, \text{T} B_4 \} \ldots \} \]
\[ \cup \{ \ldots, \{ \text{T} \text{x}, \text{F} B_3, \text{F} B_4 \}, \ldots \} \]
\[ \cup \{ \ldots, \{ \text{F} B_3, \text{T} a, \text{F} c \}, \ldots \} \]
\[ \cup \{ \ldots, \{ \text{T} B_3, \text{F} a \}, \{ \text{T} B_3, \text{T} c \}, \ldots \} \]

\[ \Lambda_\Pi = \{ \{ \text{T} \text{x}, \text{F} B_3 \}, \{ \text{T} \text{y}, \text{F} B_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

\[ \Pi = \{ a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b \} \]

\[ CF(\Pi) = \{ a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \lor B_4 \quad y \leftrightarrow B_5 \} \]

\[ LF(\Pi) = \{ (x \lor y) \rightarrow B_3 \} \]

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

\[ \Delta_\Pi = \{\ldots, \{ \text{Fx, } TB_3 \}, \{ \text{Fx, } TB_4 \} \ldots \} \]

\[ \bigcup \{\ldots, \{ \text{Tx, } FB_3, FB_4 \}, \ldots \} \]

\[ \bigcup \{\ldots, \{ FB_3, Ta, Fc \}, \ldots \} \]

\[ \bigcup \{\ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \}, \ldots \} \]

\[ \Lambda_\Pi = \{ \{ \text{Tx, } FB_3 \}, \{ \text{Ty, } FB_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Nogoods from logic programs

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

\[ CF(\Pi) = \{ a \leftrightarrow B_1 \quad b \leftrightarrow B_2 \quad c \leftrightarrow \bot \quad x \leftrightarrow B_3 \lor B_4 \quad y \leftrightarrow B_5 \} \]
\[ \cup \{ B_1 \leftrightarrow \neg b \quad B_2 \leftrightarrow \neg a \quad B_3 \leftrightarrow a \land \neg c \quad B_4 \leftrightarrow y \quad B_5 \leftrightarrow x \land b \} \]

\[ LF(\Pi) = \{ (x \lor y) \rightarrow B_3 \} \]

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

\[ \Delta_\Pi = \{ \ldots, \{ Fx, TB_3 \}, \{ Fx, TB_4 \} \ldots \} \]
\[ \cup \{ \ldots, \{ Tx, FB_3, FB_4 \}, \ldots \} \]
\[ \cup \{ \ldots, \{ FB_3, Ta, Fc \}, \ldots \} \]
\[ \cup \{ \ldots, \{ TB_3, Fa \}, \{ TB_3, Tc \}, \ldots \} \]
\[ \Lambda_\Pi = \{ \{ Tx, FB_3 \}, \{ Ty, FB_3 \} \} \]

- Size of \( \Delta_\Pi \) is linear in the size of \( \Pi \)
- Size of \( \Lambda_\Pi \) is (in general) exponential in the size of \( \Pi \)
- Satisfaction of \( \Lambda_\Pi \) can be tested in linear time
Stable Models as Solutions

Theorem

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

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 $A$ is a solution for $\Delta_\Pi \cup \Lambda_\Pi$ if $\delta \not\subseteq A$ for all $\delta \in \Delta_\Pi \cup \Lambda_\Pi$. 

Torsten Schaub (KRR@UP)  Answer Set Programming in a Nutshell
Theorem

Let \( \Pi \) be a normal logic program and \( X \subseteq \text{atom}(\Pi) \).
Then, \( X \) is a stable model of \( \Pi \) iff \( X = A^T \cap \text{atom}(\Pi) \)
for a (unique) solution \( A \) 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 \( A \) is a solution for \( \Delta_\Pi \cup \Lambda_\Pi \) if \( \delta \not\subseteq A \) for all \( \delta \in \Delta_\Pi \cup \Lambda_\Pi \).
Conflict-Driven Constraint Learning (CDCL)

\[
\text{loop}
\]

\[
\text{propagate} \quad // \text{assign deterministic consequences}
\]

\[
\text{if no conflict then}
\]

\[
\text{if all variables assigned then return variable assignment}
\]

\[
\text{else decide} \quad // \text{non-deterministically assign some variable}
\]

\[
\text{else}
\]

\[
\text{if top-level conflict then return unsatisfiable}
\]

\[
\text{else}
\]

\[
\text{analyze} \quad // \text{analyze conflict and add conflict constraint}
\]

\[
\text{backjump} \quad // \text{undo assignments violating conflict constraint}
\]
Conflict-Driven Constraint Learning (CDCL)

\[
\text{loop} \\
\text{propagate} \quad \text{// assign deterministic consequences} \\
\text{if no conflict then} \\
\quad \text{if all variables assigned then return variable assignment} \\
\quad \text{else decide} \quad \text{// non-deterministically assign some variable} \\
\text{else} \\
\quad \text{if top-level conflict then return unsatisfiable} \\
\quad \text{else} \\
\quad \text{analyze} \quad \text{// analyze conflict and add conflict constraint} \\
\quad \text{backjump} \quad \text{// undo assignments violating conflict constraint}
\]
The solver clasp

- Beyond deciding (stable) model existence, clasp allows for
  - Enumeration
  - Projective enumeration
  - Intersection and Union
  - 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
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

- Preprocessing
  - Preprocessor
    - Program Builder
      - Logic Program
- Coordination
  - SharedContext
    - Propositional Variables
      - Atoms
      - Bodies
      - Static Nogoods
      - Short Nogoods
  - Nogood Distributor
    - Recorded Nogoods
    - Shared Nogoods
- Solver 1...n
  - Decision Heuristic
    - Assignment Atoms/Bodies
  - Conflict Resolution
  - Recorded Nogoods
    - Propagation
      - Unit Propagation
      - Post Propagation
- Enumerator
  - ParallelContext
    - Threads: $S_1, S_2, \ldots, S_n$
    - Counter: $T, W, \ldots, S$
    - Queue: $P_1, P_2, \ldots, P_n$

Torsten Schaub (KRR@UP)
Multi-threaded architecture of clasp

- Preprocessing
  - Preprocessor
    - Program Builder
      - Logic Program

- Solver 1...n
  - Decision Heuristic
    - Assignment Atoms/Bodies
  - Conflict Resolution
  - Recorded Nogoods
  - Propagation
    - Unit Propagation
    - Post Propagation

- Coordination
  - SharedContext
    - Propositional Variables
      - Atoms
        - Static Nogoods
        - Short Nogoods
      - Bodies
  - Enumerator
  - ParallelContext
    - Threads $S_1 S_2 \cdots S_n$
    - Counter $T W \cdots S$
    - Queue $P_1 P_2 \cdots P_n$
  - Nogood Distributor
  - Shared Nogoods
Multi-threaded architecture of clasp
Multi-threaded architecture of clasp

Preprocessing
- Preprocessor
  - Program Builder

Program Builder

Logic Program

Solver 1...n
- Decision Heuristic
- Assignment Atoms/Bodies
- Conflict Resolution

Conflict Resolution

Recorded Nogoods

Recorded Nogoods

Nogood Distributor

Shared Context
- Propositional Variables
  - Atoms
  - Bodies
- Static Nogoods
- Short Nogoods

Parallel Context
- Threads $S_1|S_2|...|S_n$
- Counter $T|W|...|S$
- Queue $P_1|P_2|...|P_n$
- Shared Nogoods

Enumerator

Logic Program

Preprocessing
- Preprocessor
  - Program Builder
NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520

Solved instances vs. Time in seconds

- cmodels-3.79
- lp2sat-1.13
- smodels-2.34
NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520

clasp-1.3.1
conmodels-3.79
lp2sat-1.13
smodels-2.34

Solved instances
Time in seconds

470
449
410
331
NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520

Solved instances vs. Time in seconds

- clasp-3.1-t4
- clasp-1.3.1
- cmodels-3.79
- lp2sat-1.13
- smodels-2.34

Torsten Schaub (KRR@UP)
Answer Set Programming in a Nutshell
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, 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, 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, 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

Answer Set Solving in Practice
Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub
University of Potsdam

Abstract Gringo

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.


* 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.
1 Introduction
2 Foundations
3 Modeling
4 Algorithms and Systems
5 Potassco
6 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
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

\[
\text{ASP} = \text{DB} + \text{LP} + \text{KR} + \text{SAT}
\]
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

\[
\text{ASP} = \text{DB} + \text{LP} + \text{KR} + \text{SMT}^n
\]
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