Answer Set Programming in a Nutshell

Torsten Schaub

University of Potsdam
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
Introduction

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

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
The ASP Solving Process

- Expressive modeling language
- Powerful grounding and solving tools
Propositional Normal Logic Programs

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)
- $\leftarrow, \land, \sim$ 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

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

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

$$\bigcup \{ c \leftrightarrow \bot \}$$

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

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

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

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

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

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

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

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

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

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

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\}, \{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

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

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

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

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 $CF(\Pi)$:

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

- Unsupported atoms
- Unfounded atoms
Logic Programs as Propositional Formulas

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

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

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

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

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

\{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

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

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

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

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

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

Theorem (Lin and Zhao)

Let \( \Pi \) be a normal logic program and \( X \subseteq \text{atom}(\Pi) \). 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!

$ 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
```
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 $\phi = \neg \psi$

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: $\neg \neg \neg a$ and $\neg a$ 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 $\circ \in \{\land, \lor, \to\}$
- $\phi^X = \top$ 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: $a$ 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
Genuine Stable Models Semantics

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

  $$
  \begin{align*}
  \phi^X &= \bot & \text{if } X \notmodels \phi \\
  \phi^X &= \phi & \text{if } \phi \in X \\
  \phi^X &= (\psi^X \circ \mu^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \mu) \text{ for } \circ \in \{\wedge, \lor, \rightarrow\} \\
  \phi^X &= \top & \text{if } X \notmodels \psi \text{ and } \phi = \neg \psi
  \end{align*}
  $$

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

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**

  - $s(Y) :- r(Y), 2 \sum \{ X : p(X,Y), q(Y) \} 7$
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

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

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

\[
\{ a ; b ; c \}.
\]

\[
:- \text{not } a, b.
\]

\[
:- a, \text{not } b.
\]

\[
:- \text{not } c.
\]
Maximum satisfiability testing

“(a ↔ b) ∧ c”

\{ a ; b ; c \}.

:- not a, b.

:- a, not b. [10@2]

:- not c. [100@1]
{ 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

\[
\{ \text{queen}(I,1..n) \} = 1 \iff I = 1..n.
\{ \text{queen}(1..n,J) \} = 1 \iff J = 1..n.
\]

\[
\text{:- \{ queen(D-J,J) \} } \geq 2, D = 2..2*n.
\text{:- \{ queen(D+J,J) \} } \geq 2, D = 1-n..n-1.
\]
n-queens
(Experimental) constraint encoding

\[
1 \leq \text{queen}(1..n) \leq n.
\]

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

\[
\text{#disjoint \{ X : queen(X) + X : X = 1..n \}.}
\]

\[
\text{#disjoint \{ X : queen(X) - X : X = 1..n \}.}
\]
Traveling salesman
Basic encoding (no instance)

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

\text{reached}(X) :- X = \#\text{min} \{ Y : \text{node}(Y) \}.
\text{reached}(Y) :- \text{cycle}(X,Y), \text{reached}(X).

:- \text{node}(Y), \text{not} \ \text{reached}(Y).

\#\text{minimize} \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.
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).
company(c_1). 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).
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 inadmissable 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

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

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

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

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

Nogoods for \( \text{CF}(\Pi) \) and \( \text{LF}(\Pi) \)

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

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

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

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

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

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

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

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

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

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

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

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

- 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 \( LF(\Pi) \)

\[ \Delta_\Pi = \{ \ldots, \{ F_x, T B_3 \}, \{ F_x, T B_4 \} \ldots \} \]

\[ \cup \{ \ldots, \{ T x, F B_3, F B_4 \}, \ldots \} \]

\[ \cup \{ \ldots, \{ F B_3, T a, F c \}, \ldots \} \]

\[ \cup \{ \ldots, \{ T B_3, F a \}, \{ T B_3, T c \}, \ldots \} \]

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

- 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

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

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

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

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

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
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

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

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

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

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

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

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

- 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

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

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

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

Nogoods for \( \text{CF}(\Pi) \) and \( \text{LF}(\Pi) \)

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

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

- 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
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
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 $\Delta_\Pi \cup \Lambda_\Pi$.\(^1\)

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 $\delta \nsubseteq A$ for all $\delta \in \Delta_\Pi \cup \Lambda_\Pi$.\(^1\)
Conflict-Driven Constraint Learning (CDCL)

```plaintext
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
```

Torsten Schaub (KRR@UP) Answer Set Programming in a Nutshell 24 / 31
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
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
  - 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
Multi-threaded architecture of clasp

- **Preprocessing**
  - Preprocessor
    - Program Builder
      - Logic Program

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

- **Coordination**
  - SharedContext
    - Propositional Variables
      - Atoms
      - Bodies
      - Static Nogoods
      - Short Nogoods
  - Nogood Distributor
  - Enumerator
    - ParallelContext
      - Threads: \( S_1, S_2, \ldots, S_n \)
      - Counter: \( T, W, \ldots, S \)
      - Queue: \( P_1, P_2, \ldots, P_n \)
  - Shared Nogoods

- **Torsten Schaub (KRR@UP)**

---

- **Answer Set Programming in a Nutshell**

---

- **Page 26 / 31**
Multi-threaded architecture of clasp

- Preprocessing
  - Preprocessor
  - Program Builder

- Logic Program

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

- Conflict Resolution
  - Decision Heuristic
    - Assignment
    - Atoms/Bodies

- Recorded Nogoods

- Propagation
  - Unit Propagation
  - Post Propagation

- Enumerator
  - Parallel Context
    - Threads: \(S_1, S_2, \ldots, S_n\)
    - Counter: \(T, W, \ldots, S\)
    - Queue: \(P_1, P_2, \ldots, P_n\)
  - Nogood Distributor
    - Shared Nogoods
Multi-threaded architecture of clasp
Multi-threaded architecture of clasp

Preprocessing

Preprocessor

Program Builder

Logic Program

Solver 1...n

Conflict Resolution

Decision Heuristic

Assignment

Atoms/Bodies

Recorded Nogoods

Unit Propagation

Post Propagation

Propagation

Recorded Nogoods

Shared Nogoods

Enumerator

ParallelContext

Threads S₁ S₂ ... Sₙ

Counter T W ... S

Queue P₁ P₂ ... Pₙ

Nogood Distributor

Coordination

SharedContext

Propositional Variables

Atoms

Bodies

Static Nogoods

Short Nogoods
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
- Shared Context
  - Propositional Variables
    - Atoms
    - Bodies
  - Static Nogoods
  - Short Nogoods
- Nogood Distributor
- Parallel Context
  - Threads \( S_1, S_2, \ldots, S_n \)
  - Counter \( T, W, \ldots, S \)
  - Queue \( P_1, P_2, \ldots, P_n \)
  - Shared Nogoods

Torsten Schaub (KRR@UP)
NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520

The graph shows the number of solved instances over time for different solvers:
- cmodels-3.79 (green line)
- lp2sat-1.13 (blue line)
- smodels-2.34 (orange line)

The x-axis represents time in seconds, ranging from 0.1 to 600 seconds, and the y-axis represents the number of solved instances, ranging from 0 to 500 instances.
NP-Track Second ASP Competition
Run on: Dual-Processor Intel Xeon Quad-Core E5520

clasp-1.3.1
clasp-1.3.1
lp2sat-1.13
lp2sat-1.13
smodels-2.34
smodels-2.34

Solved instances

Time in seconds

clasp-1.3.1
clasp-1.3.1
lp2sat-1.13
lp2sat-1.13
smodels-2.34
smodels-2.34

470
470
449
449
410
410
331
331
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 vs. Time in seconds
Outline

1. Introduction
2. Foundations
3. Modeling
4. Algorithms and Systems
5. Potassco
6. Summary
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, asp cuda, 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

---

**Abstract Gringo**

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

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

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

VLADIMIR LIFSCHITZ  
University 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.


---

The 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.

Outline

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

ASP = DB + LP + KR + 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