Answer Set Programming, the Solving Paradigm for Knowledge Representation and Reasoning

Martin Gebser Torsten Schaub

University of Potsdam

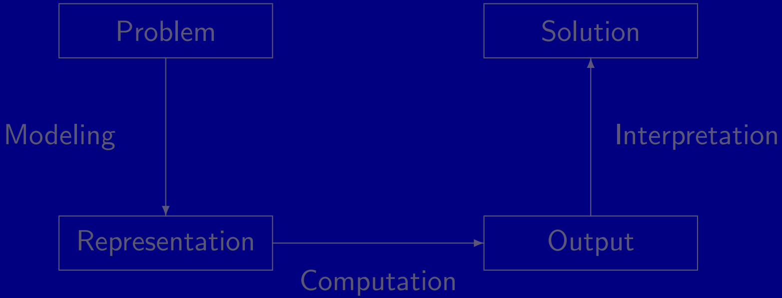
Outline

Outline

Goal: Declarative problem solving

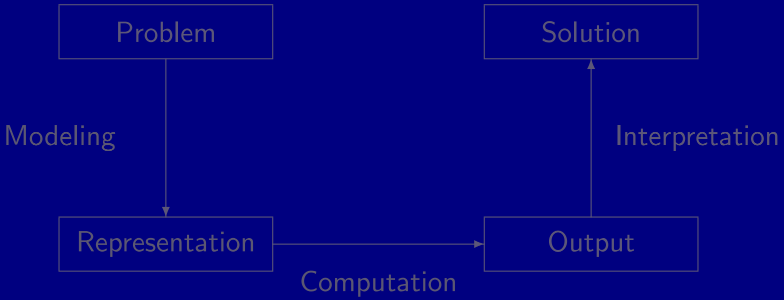
- *"What is the problem?"*

  instead of
- *"How to solve the problem?"*



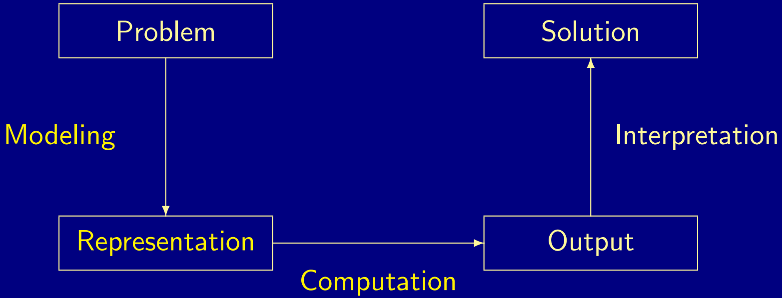
Goal: Declarative problem solving

- *"What is the problem?"*
  instead of
- *"How to solve the problem?"*



Goal: Declarative problem solving

- *"What is the problem?"*
  instead of
- *"How to solve the problem?"*



Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (logic-based) knowledge representation and reasoning
  - (deductive) databases
  - constraint solving (in particular, SAT solving)
  - logic programming (with negation)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way (being more compact than SAT)
- The versatility of ASP is reflected by the ASP solver clasp, winning first places at ASP'09, PB'09, and SAT'09
- ASP embraces many emerging application areas!

Outline

Answer set: Basic idea

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

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

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

This formula has one stable model, called answer set:

$$\{p, q\}$$

$$\Pi_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ not \ r \end{array}}$$

Informally, a set X of atoms is an answer set of a logic program Π

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

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

Answer set: Basic idea

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

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

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

This formula has one stable model, called answer set:

$\Pi_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \mathit{not}\ r \end{array}}$

$$\{p, q\}$$

Informally, a set X of atoms is an answer set of a logic program Π

if X is a (classical) model of Π and

if all atoms in X are justified by some rule in Π

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

Answer set: Basic idea

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

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

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

This formula has one stable model, called answer set:

$$\{p, q\}$$

$$\Pi_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q,\ \mathit{not}\ r \end{array}}$$

Informally, a set X of atoms is an answer set of a logic program Π

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

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

Answer set: Basic idea

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

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

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

This formula has one stable model, called answer set:

$$\{p, q\}$$

Π_Φ $\boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \; \mathit{not}\ r \end{array}}$

Informally, a set X of atoms is an answer set of a logic program Π

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

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

Answer set: Basic idea

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

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

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

This formula has one stable model, called answer set:

$$\{p, q\}$$

$$\Pi_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q,\ \mathit{not}\ r \end{array}}$$

Informally, a set X of atoms is an answer set of a logic program Π

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

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

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n,$$

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

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

    $$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n,$$

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

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

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

    $$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

$$a \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n,$$

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

The reduct, Π^X, of a program Π relative to a set X of atoms is defined by

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

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

$$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n,$$

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

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

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

    $$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n,$$

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

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

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

    $$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n,$$

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

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

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

    $$X = Cn(\Pi^X)$$

Formal Definition

- Syntax
  - A rule, r, is an expression of the form

    $$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n,$$

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

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

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

    $$X = Cn(\Pi^X)$$

Language Constructs

Variables (over the Herbrand Universe)

`p(X) :- q(X)` over constants $\{a, b, c\}$ stands for

`p(a) :- q(a), p(b) :- q(b), p(c) :- q(c)`

Conditional Literals

`p :- q(X) : r(X)` given `r(a), r(b), r(c)` stands for

`p :- q(a), q(b), q(c)`

Disjunction

`p(X) ; q(X) :- r(X)`

Integrity Constraints

`:- q(X), p(X)`

Choice

`2 { p(X,Y) : q(X) } 7 :- r(Y)`

Aggregates

`s(Y) :- r(Y), 2 #count { p(X,Y) : q(X) } 7`

also: `#sum, #times, #avg, #min, #max, #even, #odd`

Language Constructs

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

Language Constructs

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

Language Constructs

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

Language Constructs

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

Language Constructs

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

Language Constructs

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

Language Constructs

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

Reasoning Modes

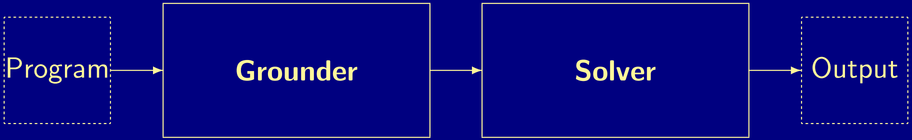
- Satisfiability
- Enumeration†
- Projection†
- Intersection‡
- Union‡
- Optimization
- Sampling

† without solution recording

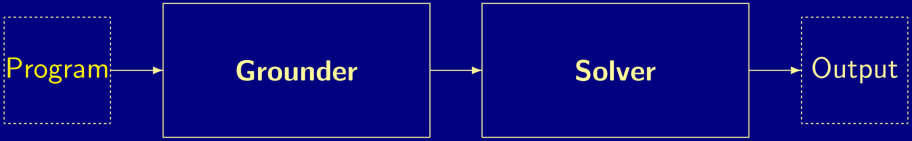
‡ without solution enumeration

Outline

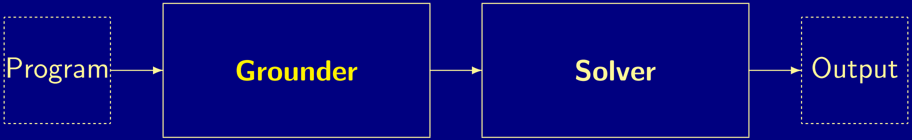
ASP Solving Process



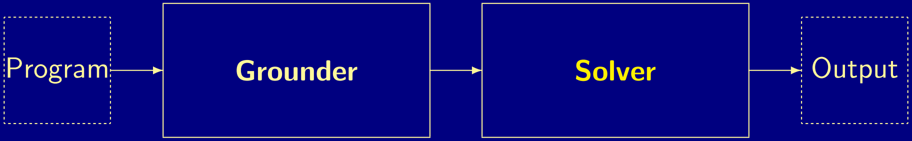
ASP Solving Process



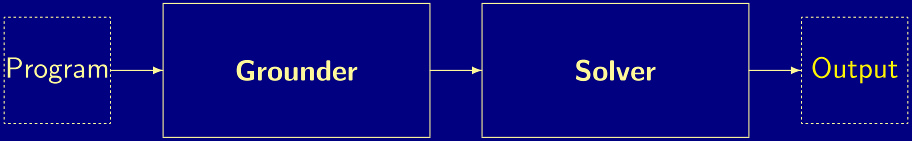
ASP Solving Process



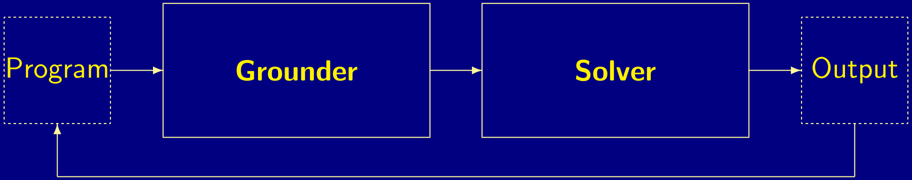
ASP Solving Process



ASP Solving Process



ASP Solving Process



Graph Coloring

```
node(1..6).

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

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

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

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

Graph Coloring

```
node(1..6).

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

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

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

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

Graph Coloring

```
node(1..6).

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

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

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

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

Graph Coloring

```
node(1..6).

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

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

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

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

Graph Coloring: Grounding

```
$ gringo -t color.lp
```

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

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

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

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

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

Graph Coloring: Grounding

```
$ gringo -t color.lp
```

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

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

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

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

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

Graph Coloring: Solving

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

```
clasp version 1.2.1
Reading from stdin
Reading      : Done(0.000s)
Preprocessing: Done(0.000s)
Solving...
Answer: 1
color(1,b) color(2,r) color(3,r) color(4,g) color(5,b) color(6,g) node(1) ... edge(1,2) ... col(r) ...
Answer: 2
color(1,g) color(2,r) color(3,r) color(4,b) color(5,g) color(6,b) node(1) ... edge(1,2) ... col(r) ...
Answer: 3
color(1,b) color(2,g) color(3,g) color(4,r) color(5,b) color(6,r) node(1) ... edge(1,2) ... col(r) ...
Answer: 4
color(1,g) color(2,b) color(3,b) color(4,r) color(5,g) color(6,r) node(1) ... edge(1,2) ... col(r) ...
Answer: 5
color(1,r) color(2,b) color(3,b) color(4,g) color(5,r) color(6,g) node(1) ... edge(1,2) ... col(r) ...
Answer: 6
color(1,r) color(2,g) color(3,g) color(4,b) color(5,r) color(6,b) node(1) ... edge(1,2) ... col(r) ...

Models      : 6
Time        : 0.000  (Solving: 0.000)
```

Graph Coloring: Solving

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

```
clasp version 1.2.1
Reading from stdin
Reading      : Done(0.000s)
Preprocessing: Done(0.000s)
Solving...
Answer: 1
color(1,b) color(2,r) color(3,r) color(4,g) color(5,b) color(6,g) node(1) ... edge(1,2) ... col(r) ...
Answer: 2
color(1,g) color(2,r) color(3,r) color(4,b) color(5,g) color(6,b) node(1) ... edge(1,2) ... col(r) ...
Answer: 3
color(1,b) color(2,g) color(3,g) color(4,r) color(5,b) color(6,r) node(1) ... edge(1,2) ... col(r) ...
Answer: 4
color(1,g) color(2,b) color(3,b) color(4,r) color(5,g) color(6,r) node(1) ... edge(1,2) ... col(r) ...
Answer: 5
color(1,r) color(2,b) color(3,b) color(4,g) color(5,r) color(6,g) node(1) ... edge(1,2) ... col(r) ...
Answer: 6
color(1,r) color(2,g) color(3,g) color(4,b) color(5,r) color(6,b) node(1) ... edge(1,2) ... col(r) ...

Models      : 6
Time        : 0.000  (Solving: 0.000)
```

Traveling Salesperson

```
node(1..6).

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

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

Traveling Salesperson

```
node(1..6).

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

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

Traveling Salesperson

```
node(1..6).

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

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

Traveling Salesperson

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

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

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

#minimize [ cycle(X,Y) : cost(X,Y,C) = C ].
```

Traveling Salesperson

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

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

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

#minimize [ cycle(X,Y) : cost(X,Y,C) = C ].
```

Traveling Salesperson

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

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

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

#minimize [ cycle(X,Y) : cost(X,Y,C) = C ].
```

Traveling Salesperson

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

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

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

#minimize [ cycle(X,Y) : cost(X,Y,C) = C ].
```

Reviewer Assignment (by Ilkka Niemelä)

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

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

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

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

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

Reviewer Assignment (by Ilkka Niemelä)

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

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

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

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

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

Reviewer Assignment (by Ilkka Niemelä)

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

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

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

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

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

Reviewer Assignment (by Ilkka Niemelä)

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

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

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

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

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

Reviewer Assignment (by Ilkka Niemelä)

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

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

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

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

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

Simplistic STRIPS Planning

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

time(1..k).    lasttime(T) :- time(T), not time(T+1).

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

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

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

 :- query(F), not holds(F,T), lasttime(T).
```

Simplistic STRIPS Planning

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

time(1..k).    lasttime(T) :- time(T), not time(T+1).

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

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

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

 :- query(F), not holds(F,T), lasttime(T).
```

Simplistic STRIPS Planning

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

time(1..k).    lasttime(T) :- time(T), not time(T+1).

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

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

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

 :- query(F), not holds(F,T), lasttime(T).
```

Simplistic STRIPS Planning with iASP

```
#base.

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

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

#cumulative t.

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

ocdel(F,t) :- occ(A,t), del(A,F).
holds(F,t) :- occ(A,t), add(A,F).
holds(F,t) :- holds(F,t-1), not ocdel(F,t).

#volatile t.

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

Simplistic STRIPS Planning with iASP

```
#base.

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

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

#cumulative t.

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

ocdel(F,t) :- occ(A,t), del(A,F).
holds(F,t) :- occ(A,t), add(A,F).
holds(F,t) :- holds(F,t-1), not ocdel(F,t).

#volatile t.

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

Simplistic STRIPS Planning with iASP

```
#base.

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

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

#cumulative t.

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

ocdel(F,t) :- occ(A,t), del(A,F).
holds(F,t) :- occ(A,t), add(A,F).
holds(F,t) :- holds(F,t-1), not ocdel(F,t).

#volatile t.

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

Simplistic STRIPS Planning with iASP

```
#base.

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

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

#cumulative t.

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

ocdel(F,t) :- occ(A,t), del(A,F).
holds(F,t) :- occ(A,t), add(A,F).
holds(F,t) :- holds(F,t-1), not ocdel(F,t).

#volatile t.

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

What is ASP good for?

- Combinatorial search problems
  (some with substantial amount of data):
  - For instance, auctions, bio-informatics, computer-aided verification, configuration, constraint satisfaction, diagnosis, information integration, planning and scheduling, security analysis, semantic web, wire-routing, zoology and linguistics, and many more
- My favorite: Using ASP as a basis for a decision support system for NASA's space shuttle (Gelfond et al., Texas Tech)
- Our own applications:
  - Automatic synthesis of multiprocessor systems
  - Inconsistency detection in large biological networks
  - Home monitoring for risk prevention in assisted living
  - General game playing

What does ASP offer?

- Integration of KR, DB, and search techniques
- Compact, easily maintainable problem representations
- Rapid application development tool
- Easy handling of dynamic, knowledge intensive applications
  (including: data, frame axioms, exceptions, defaults, closures, etc.)

What does ASP offer?

- Integration of KR, DB, and search techniques
- Compact, easily maintainable problem representations
- Rapid application development tool
- Easy handling of dynamic, knowledge intensive applications
  (including: data, frame axioms, exceptions, defaults, closures, etc.)

$$\text{ASP} = \text{KR} + \text{DB} + \text{Search}$$

Outline

Conflict-Driven Answer Set Solving

- Idea
  - View inferences in Answer Set Programming (ASP) as unit propagation on nogoods.

    (A nogood expresses a constraint violated by any solution.)

- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CSP and SAT
  - Highly competitive implementation

- Nogoods from Logic Programs are
  - nogoods from Clark's Completion,
  - nogoods from Unfounded Sets, and
  - nogoods from Aggregates.

Conflict-Driven Answer Set Solving

- Idea
  - View inferences in Answer Set Programming (ASP) as unit propagation on nogoods.

    (A nogood expresses a constraint violated by any solution.)
- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CSP and SAT
  - Highly competitive implementation
- Nogoods from Logic Programs are
  - nogoods from Clark's Completion,
  - nogoods from Unfounded Sets, and
  - nogoods from Aggregates.

Conflict-Driven Answer Set Solving

- Idea
  - View inferences in Answer Set Programming (ASP) as unit propagation on nogoods.

    (A nogood expresses a constraint violated by any solution.)
- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CSP and SAT
  - Highly competitive implementation
- Nogoods from Logic Programs are
  - nogoods from Clark's Completion,
  - nogoods from Unfounded Sets, and
  - nogoods from Aggregates.

Conflict-Driven Answer Set Solving

- Idea
  - View inferences in Answer Set Programming (ASP) as unit propagation on nogoods.

    (A nogood expresses a constraint violated by any solution.)
- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CSP and SAT
  - Highly competitive implementation
- Nogoods from Logic Programs are
  - nogoods from Clark's Completion, $O(n)$
  - nogoods from Unfounded Sets, and
  - nogoods from Aggregates.

Conflict-Driven Answer Set Solving

- Idea
  - View inferences in Answer Set Programming (ASP) as unit propagation on nogoods.

    (A nogood expresses a constraint violated by any solution.)
- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CSP and SAT
  - Highly competitive implementation
- Nogoods from Logic Programs are
  - nogoods from Clark's Completion, $O(n)$
  - nogoods from Unfounded Sets, and $O(2^n)$
  - nogoods from Aggregates. $O(2^n)$

Nogoods from Clark's Completion

- For example, for body $\{x, \mathit{not}\ y\}$, we obtain 3 nogoods

$$\boxed{\begin{array}{c} \ldots \leftarrow x, \mathit{not}\ y \\ \vdots \\ \ldots \leftarrow x, \mathit{not}\ y \end{array}}$$

$$\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$$

$$\{\,\{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T}y\}\,\}$$

- For nogood $\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$, the signed literal
  - $\mathbf{T}\{x, \mathit{not}\ y\}$ is unit-resulting wrt assignment $(\mathbf{T}x, \mathbf{F}y)$ and
  - $\mathbf{T}y$ is unit-resulting wrt assignment $(\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x)$.

- Similarly, there are atom-oriented nogoods (see IJCAI'07).

Nogoods from Clark's Completion

- For example, for body $\{x, \mathit{not}\ y\}$, we obtain 3 nogoods

$$\boxed{\begin{array}{c} \ldots \leftarrow x, \mathit{not}\ y \\ \vdots \\ \ldots \leftarrow x, \mathit{not}\ y \end{array}}$$

$$\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$$

$$\{\,\{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T}y\}\,\}$$

- For nogood $\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$, the signed literal
  - $\mathbf{T}\{x, \mathit{not}\ y\}$ is unit-resulting wrt assignment $(\mathbf{T}x, \mathbf{F}y)$ and
  - $\mathbf{T}y$ is unit-resulting wrt assignment $(\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x)$.

- Similarly, there are atom-oriented nogoods (see IJCAI'07).

Nogoods from Clark's Completion

- For example, for body $\{x, \mathit{not}\ y\}$, we obtain 3 nogoods

$$\boxed{\begin{array}{c} \ldots \leftarrow x, \mathit{not}\ y \\ \vdots \\ \ldots \leftarrow x, \mathit{not}\ y \end{array}}$$

$$\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$$

$$\{\,\{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T}y\}\,\}$$

- For nogood $\{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$, the signed literal
  - $\mathbf{T}\{x, \mathit{not}\ y\}$ is unit-resulting wrt assignment $(\mathbf{T}x, \mathbf{F}y)$ and
  - $\mathbf{T}y$ is unit-resulting wrt assignment $(\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x)$.

- Similarly, there are atom-oriented nogoods (see IJCAI'07).

Basic Decision Algorithm

loop

Propagate // (Boolean) constraint propagation

if no conflict **then**

if all variables assigned **then return** solution

else *Decide* // pick and assign some free literal

else if top-level conflict **then**

return unsatisfiable

else

Analyze // resolve conflict and record a conflict constraint

Backjump // undo assignments until conflict constraint is unit

Algorithm 1: Nogood Propagation

Input : A logic program Π, a set ∇ of nogoods, and an assignment A.
Output : An extended assignment and set of nogoods.

$U \leftarrow \emptyset$ *// set of unfounded atoms*
loop
  repeat
    if $\delta \subseteq A$ for some $\delta \in \Delta_\Pi \cup \nabla$ **then return** (A, ∇) *// conflict*
    $\Sigma \leftarrow \{\delta \in \Delta_\Pi \cup \nabla \mid (\delta \setminus A) = \{\sigma\}, \overline{\sigma} \notin A\}$ *// unit-resulting nogoods*
    if $\Sigma \neq \emptyset$ **then**
      let $\sigma \in (\delta \setminus A)$ for some $\delta \in \Sigma$ **in**
        $A \leftarrow A \circ (\overline{\sigma})$ // $dl(\overline{\sigma}) = max(\{dl(\rho) \mid \rho \in (\delta \setminus \{\sigma\})\} \cup \{0\})$
  until $\Sigma = \emptyset$
  if Π is tight **then return** (A, ∇) // *no unfounded set* $\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}})$
  else
    $U \leftarrow (U \setminus A^{\mathbf{F}})$
    if $U = \emptyset$ **then** $U \leftarrow \text{UnfoundedSet}(\Pi, A)$
    if $U = \emptyset$ **then return** (A, ∇) // *no unfounded set* $\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}})$
    let $p \in U$ **in**
      $\nabla \leftarrow \nabla \cup \{\lambda(p, U)\}$ *// record unit-resulting or violated loop nogood*

Outline

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro

Benchmarking: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: `http://asparagus.cs.uni-potsdam.de`

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: http://asparagus.cs.uni-potsdam.de

http://potassco.sourceforge.net

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

- *Grounder*: Gringo, pyngo
- *Solver*: clasp, claspD, claspar
- *Grounder+Solver*: Clingo, iClingo, Clingcon
- *Further Tools*: claspfolio, coala, inca, plasp, sbass, xorro
- *Benchmarking*: `http://asparagus.cs.uni-potsdam.de`

clasp

- clasp is a native ASP solver for extended logic programs
- clasp can also be run as SAT or PB solver
  - From version 1.3, input formats are recognized and distinguished
  - Search engine unmodified
- clasp's search algorithm relies on conflict-driven learning, featuring:

  Conflict Analysis via First-UIP Scheme
  Conflict Constraint Recording and Deletion
  Backjumping
  Restarts
  Lookback-based Decision Heuristics
  Progress Saving
  Unit Propagation via Watched Literals
  Dedicated Propagation of Binary and Ternary Constraints
  Dedicated Propagation of Cardinality and Weight Constraints
  Equivalence Reasoning and Resolution-based Preprocessing

clasp

- clasp is a native ASP solver for extended logic programs
- clasp can also be run as SAT or PB solver
  - From version 1.3, input formats are recognized and distinguished
  - Search engine unmodified
- clasp's search algorithm relies on conflict-driven learning, featuring:
  - Conflict Analysis via First-UIP Scheme
  - Conflict Constraint Recording and Deletion
  - Backjumping
  - Restarts
  - Lookback-based Decision Heuristics
  - Progress Saving
  - Unit Propagation via Watched Literals
  - Dedicated Propagation of Binary and Ternary Constraints
  - Dedicated Propagation of Cardinality and Weight Constraints
  - Equivalence Reasoning and Resolution-based Preprocessing

clasp

- clasp is a native ASP solver for extended logic programs
- clasp can also be run as SAT or PB solver
  - From version 1.3, input formats are recognized and distinguished
  - Search engine unmodified
- clasp's search algorithm relies on conflict-driven learning, featuring:
  - Conflict Analysis via First-UIP Scheme
  - Conflict Constraint Recording and Deletion
  - Backjumping
  - Restarts
  - Lookback-based Decision Heuristics
  - Progress Saving
  - Unit Propagation via Watched Literals
  - Dedicated Propagation of Binary and Ternary Constraints
  - Dedicated Propagation of Cardinality and Weight Constraints
  - Equivalence Reasoning and Resolution-based Preprocessing

clasp

- clasp is a native ASP solver for extended logic programs
- clasp can also be run as SAT or PB solver
  - From version 1.3, input formats are recognized and distinguished
  - Search engine unmodified
- clasp's search algorithm relies on conflict-driven learning, featuring:
  - Conflict Analysis via First-UIP Scheme
  - Conflict Constraint Recording and Deletion
  - Backjumping
  - Restarts
  - Lookback-based Decision Heuristics
  - Progress Saving
  - Unit Propagation via Watched Literals
  - Dedicated Propagation of Binary and Ternary Constraints
  - Dedicated Propagation of Cardinality and Weight Constraints
  - Equivalence Reasoning and Resolution-based Preprocessing

Solving the 2009 ASP Competition (NP)

Benchmark	#		clasp		clasp^+		cmodels[m]		smodels	
15Puzzle	16	(16/0)	33.01	(0)	20.18	(0)	31.36	(0)	600.00	(48)
BlockedNQueens	29	(15/14)	5.09	(0)	4.91	(0)	9.04	(0)	29.37	(0)
ChannelRouting	10	(6/4)	120.13	(6)	120.14	(6)	120.58	(6)	120.90	(6)
EdgeMatching	29	(29/0)	0.23	(0)	0.41	(0)	59.32	(0)	60.32	(0)
Fastfood	29	(10/19)	1.17	(0)	0.90	(0)	29.22	(0)	83.93	(3)
GraphColouring	29	(9/20)	421.55	(60)	357.88	(39)	422.66	(57)	453.77	(63)
Hanoi	15	(15/0)	11.76	(0)	3.97	(0)	2.92	(0)	523.77	(39)
HierarchicalClustering	12	(8/4)	0.16	(0)	0.17	(0)	0.76	(0)	1.56	(0)
SchurNumbers	29	(13/16)	17.44	(0)	49.60	(0)	75.70	(0)	504.17	(72)
Solitaire	27	(22/5)	204.78	(27)	162.82	(21)	175.69	(21)	316.96	(36)
Sudoku	10	(10/0)	0.15	(0)	0.16	(0)	2.55	(0)	0.25	(0)
WeightBoundedDomSet	29	(29/0)	123.13	(15)	102.18	(12)	300.26	(36)	400.84	(51)
$\varnothing(\varnothing)$ *(tight)*	264	(182/82)	78.22	(9)	68.61	(6.50)	102.50	(10)	257.99	(26.50)
ConnectedDomSet	21	(10/11)	40.42	(3)	36.11	(3)	7.46	(0)	183.76	(15)
GeneralizedSlitherlink	29	(29/0)	0.10	(0)	0.22	(0)	1.92	(0)	0.16	(0)
GraphPartitioning	13	(6/7)	9.27	(0)	7.98	(0)	20.19	(0)	92.10	(3)
HamiltonianPath	29	(29/0)	0.07	(0)	0.06	(0)	0.21	(0)	2.22	(0)
KnightTour	10	(10/0)	124.29	(6)	91.80	(3)	242.48	(12)	150.55	(3)
Labyrinth	29	(29/0)	123.82	(12)	82.92	(6)	142.24	(6)	594.10	(81)
MazeGeneration	29	(10/19)	91.17	(12)	89.89	(12)	90.41	(12)	293.62	(42)
Sokoban	29	(9/20)	0.73	(0)	0.80	(0)	3.39	(0)	176.01	(15)
TravellingSalesperson	29	(29/0)	0.05	(0)	0.06	(0)	317.82	(7)	0.22	(0)
WireRouting	23	(12/11)	42.81	(3)	36.36	(3)	175.73	(12)	448.32	(45)
$\varnothing(\varnothing)$ *(nontight)*	241	(173/68)	43.27	(3.60)	34.62	(2.70)	100.19	(4.90)	194.11	(20.40)
$\varnothing(\varnothing)$	505	(355/150)	62.33	(6.55)	53.16	(4.77)	101.45	(7.68)	228.95	(23.73)

clasp (version 1.3.1)
clasp^+ = clasp –sat-prepro –trans-ext=dynamic

cmodels[m] (version 3.79 with minisat 2.0)
smodels (version 2.34 with option -restart)

Solving the 2009 ASP Competition (NP)

Benchmark	#		clasp		clasp^{+}		cmodels[m]		smodels	
15Puzzle	16	(16/0)	33.01	(0)	20.18	(0)	31.36	(0)	600.00	(48)
BlockedNQueens	29	(15/14)	5.09	(0)	4.91	(0)	9.04	(0)	29.37	(0)
ChannelRouting	10	(6/4)	120.13	(6)	120.14	(6)	120.58	(6)	120.90	(6)
EdgeMatching	29	(29/0)	0.23	(0)	0.41	(0)	59.32	(0)	60.32	(0)
Fastfood	29	(10/19)	1.17	(0)	0.90	(0)	29.22	(0)	83.93	(3)
GraphColouring	29	(9/20)	421.55	(60)	357.88	(39)	422.66	(57)	453.77	(63)
Hanoi	15	(15/0)	11.76	(0)	3.97	(0)	2.92	(0)	523.77	(39)
HierarchicalClustering	12	(8/4)	0.16	(0)	0.17	(0)	0.76	(0)	1.56	(0)
SchurNumbers	29	(13/16)	17.44	(0)	49.60	(0)	75.70	(0)	504.17	(72)
Solitaire	27	(22/5)	204.78	(27)	162.82	(21)	175.69	(21)	316.96	(36)
Sudoku	10	(10/0)	0.15	(0)	0.16	(0)	2.55	(0)	0.25	(0)
WeightBoundedDomSet	29	(29/0)	123.13	(15)	102.18	(12)	300.26	(36)	400.84	(51)
∅(∅) *(tight)*	264	(182/82)	78.22	(9)	68.61	(6.50)	102.50	(10)	257.99	(26.50)
ConnectedDomSet	21	(10/11)	40.42	(3)	36.11	(3)	7.46	(0)	183.76	(15)
GeneralizedSlitherlink	29	(29/0)	0.10	(0)	0.22	(0)	1.92	(0)	0.16	(0)
GraphPartitioning	13	(6/7)	9.27	(0)	7.98	(0)	20.19	(0)	92.10	(3)
HamiltonianPath	29	(29/0)	0.07	(0)	0.06	(0)	0.21	(0)	2.22	(0)
KnightTour	10	(10/0)	124.29	(6)	91.80	(3)	242.48	(12)	150.55	(3)
Labyrinth	29	(29/0)	123.82	(12)	82.92	(6)	142.24	(6)	594.10	(81)
MazeGeneration	29	(10/19)	91.17	(12)	89.89	(12)	90.41	(12)	293.62	(42)
Sokoban	29	(9/20)	0.73	(0)	0.80	(0)	3.39	(0)	176.01	(15)
TravellingSalesperson	29	(29/0)	0.05	(0)	0.06	(0)	317.82	(7)	0.22	(0)
WireRouting	23	(12/11)	42.81	(3)	36.36	(3)	175.73	(12)	448.32	(45)
∅(∅) *(nontight)*	241	(173/68)	43.27	(3.60)	34.62	(2.70)	100.19	(4.90)	194.11	(20.40)
∅(∅)	505	(355/150)	62.33	(6.55)	53.16	(4.77)	101.45	(7.68)	228.95	(23.73)

clasp (version 1.3.1)
clasp^{+} = clasp –sat-prepro –trans-ext=dynamic

cmodels[m] (version 3.79 with minisat 2.0)
smodels (version 2.34 with option -restart)

Outline

Summary

- ASP is emerging as a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
  - `http://potassco.sourceforge.net`
  - ASP'09, PB'09, and SAT'09
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications

Summary

- ASP is emerging as a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
  - `http://potassco.sourceforge.net`
  - ASP'09, PB'09, and SAT'09
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications

$$\text{ASP} = \text{KR} + \text{DB} + \text{Search}$$

Summary

- ASP is emerging as a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
  - `http://potassco.sourceforge.net`
  - ASP'09, PB'09, and SAT'09
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications

$$\text{ASP} = \text{KR} + \text{DB} + \text{SAT}$$