



PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

Lecture 5 Answer-Set Programming Motivation and Introduction

* slides adapted from Torsten Schaub [Gebser et al.(2012)]

Sarah Gaggl

Dresden, 14th November 2019

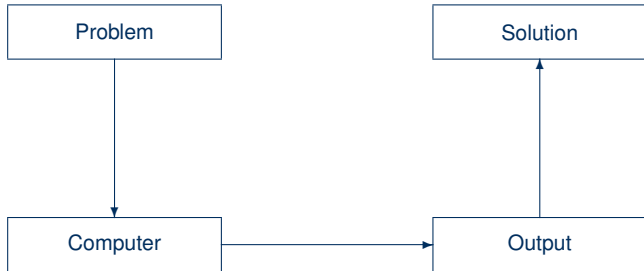
Agenda

- 1 Introduction
- 2 Uninformed Search versus Informed Search (Best First Search, A* Search, Heuristics)
- 3 Local Search, Stochastic Hill Climbing, Simulated Annealing
- 4 Tabu Search
- 5 Answer-set Programming (ASP)
- 6 Constraint Satisfaction (CSP)
- 7 Evolutionary Algorithms/ Genetic Algorithms
- 8 Structural Decomposition Techniques (Tree/Hypertree Decompositions)

Outline

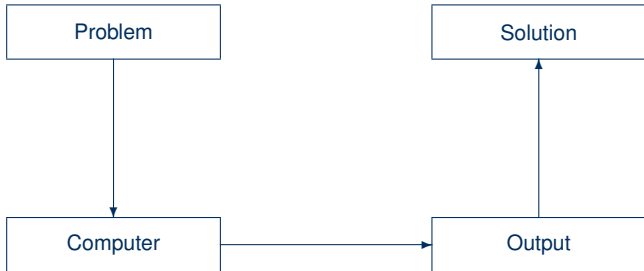
- 1 Motivation
 - Declarative Problem Solving
 - ASP in a Nutshell
 - ASP Paradigm
- 2 Introduction
 - Syntax
 - Semantics
 - Examples
 - Language Constructs
 - Modeling

Informatics



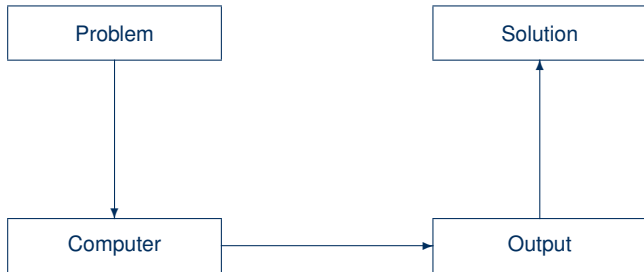
Informatics

“What is the problem?” versus “How to solve the problem?”



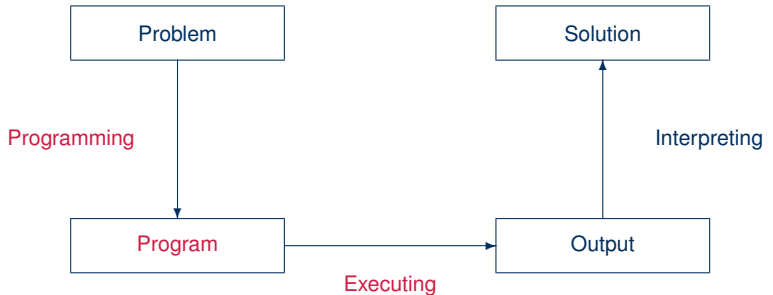
Traditional programming

“What is the problem?” versus “How to solve the problem?”



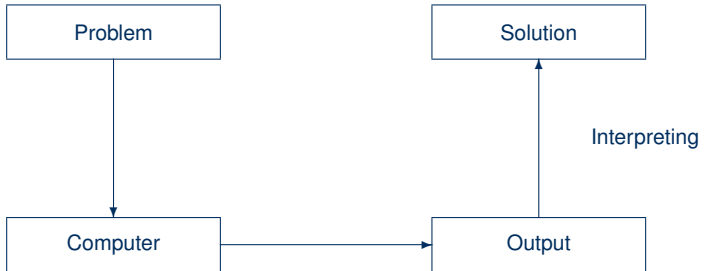
Traditional programming

“What is the problem?” versus “How to solve the problem?”



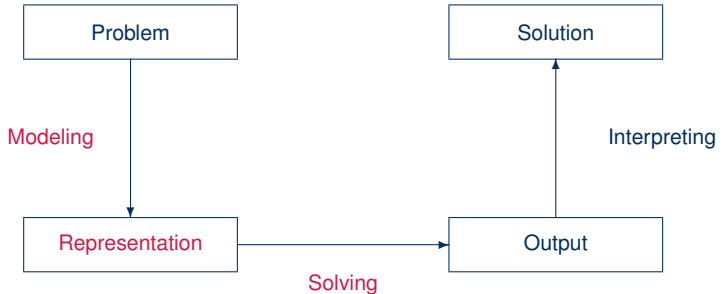
Declarative problem solving

“What is the problem?” versus “How to solve the problem?”

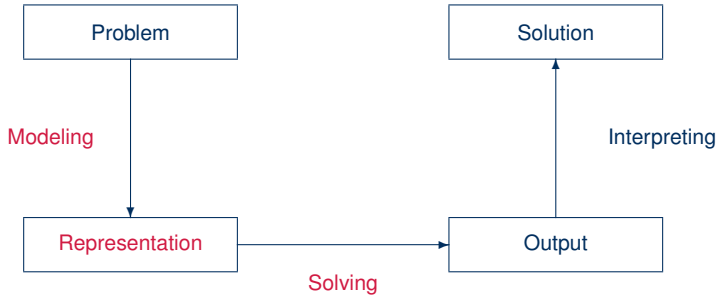


Declarative problem solving

“What is the problem?” versus “How to solve the problem?”



Declarative problem solving



Answer Set Programming

in a Nutshell

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

Answer Set Programming

in a Nutshell

- ASP is an approach to **declarative problem solving**, combining
 - a rich yet simple modeling language
 - with high-performance solving capacities
- ASP has its roots in
 - (deductive) databases
 - logic programming (with negation)
 - (logic-based) knowledge representation and (nonmonotonic) reasoning
 - constraint solving (in particular, SATisfiability testing)

Answer Set Programming

in a Nutshell

- ASP is an approach to **declarative problem solving**, combining
 - a rich yet simple modeling language
 - with high-performance solving capacities
- ASP has its roots in
 - (deductive) databases
 - logic programming (with negation)
 - (logic-based) knowledge representation and (nonmonotonic) reasoning
 - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way

Answer Set Programming

in a Nutshell

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

Answer Set Programming

in a Nutshell

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

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to **declarative problem solving**, combining
 - a rich yet simple modeling language
 - with high-performance solving capacitiestailored to **Knowledge Representation and Reasoning**

Answer Set Programming

in a Hazelnutshell

- ASP is an approach to **declarative problem solving**, combining
 - a rich yet simple modeling language
 - with high-performance solving capacitiestailored to Knowledge Representation and Reasoning

ASP = DB+LP+KR+SAT

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

Model Generation based Problem Solving

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

Model Generation based Problem Solving

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

Model Generation based Problem Solving

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

LP-style playing with blocks

Prolog program

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

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

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

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

LP-style playing with blocks

Prolog program

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

Prolog queries

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


LP-style playing with blocks

Prolog program

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

Prolog queries

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

LP-style playing with blocks

Prolog program

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

Prolog queries (testing entailment)

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

LP-style playing with blocks

Shuffled Prolog program

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

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

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

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

LP-style playing with blocks

Shuffled Prolog program

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

Prolog queries

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

LP-style playing with blocks

Shuffled Prolog program

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

Prolog queries (answered via fixed execution)

```
?- above(a,c).  
  
Fatal Error: local stack overflow.
```

SAT-style playing with blocks

Formula

$on(a, b)$
 $\wedge on(b, c)$
 $\wedge (on(X, Y) \rightarrow above(X, Y))$
 $\wedge (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))$

SAT-style playing with blocks

Formula

$on(a, b)$
 $\wedge on(b, c)$
 $\wedge (on(X, Y) \rightarrow above(X, Y))$
 $\wedge (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))$

Herbrand model

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

SAT-style playing with blocks

Formula

$on(a, b)$
 $\wedge on(b, c)$
 $\wedge (on(X, Y) \rightarrow above(X, Y))$
 $\wedge (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))$

Herbrand model (among 426!)

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

SAT-style playing with blocks

Formula

$on(a, b)$
 $\wedge on(b, c)$
 $\wedge (on(X, Y) \rightarrow above(X, Y))$
 $\wedge (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))$

Herbrand model (among 426!)

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

SAT-style playing with blocks

Formula

$on(a, b)$
 $\wedge on(b, c)$
 $\wedge (on(X, Y) \rightarrow above(X, Y))$
 $\wedge (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))$

Herbrand model (among 426!)

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

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

KR's shift of paradigm

Model Generation based approach (eg. SATisfiability testing)

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

➡ **Answer Set Programming (ASP)**

Model Generation based Problem Solving

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

Answer Set Programming at large

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

Answer Set Programming commonly

Representation	Solution
propositional theories	stable models
propositional programs	stable models
first-order theories	stable models
⋮	⋮

Answer Set Programming in practice

Representation	Solution
propositional programs	stable models
⋮	⋮

Answer Set Programming in practice

Representation	Solution
propositional programs	stable models
first-order programs	stable Herbrand models

ASP-style playing with blocks

Logic program

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

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

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

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

ASP-style playing with blocks

Logic program

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

Stable Herbrand model

```
{ on(a,b), on(b,c), above(b,c), above(a,b), above(a,c) }
```

ASP-style playing with blocks

Logic program

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

Stable Herbrand model (and no others)

```
{ on(a,b), on(b,c), above(b,c), above(a,b), above(a,c) }
```

ASP-style playing with blocks

Logic program

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

Stable Herbrand model (and no others)

```
{ on(a,b), on(b,c), above(b,c), above(a,b), above(a,c) }
```

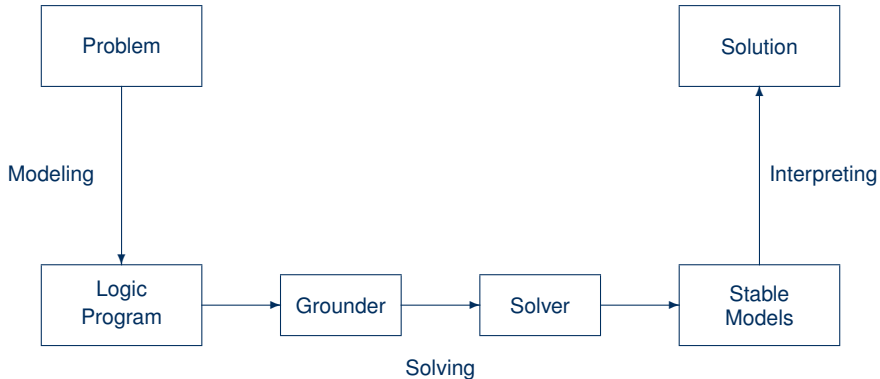
ASP versus LP

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

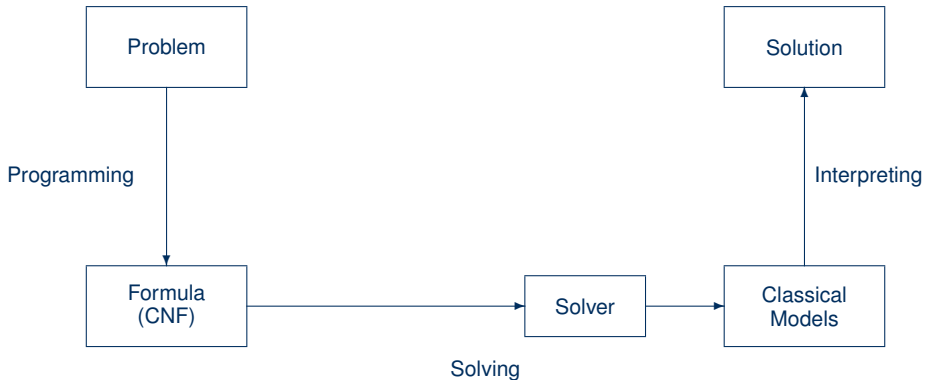
ASP versus SAT

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

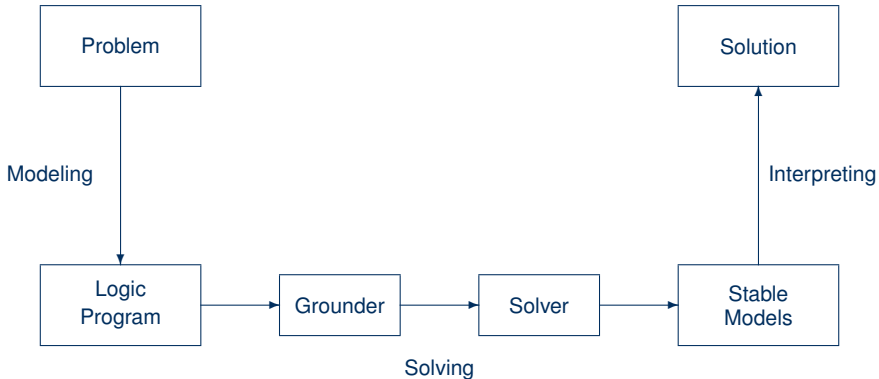
ASP solving



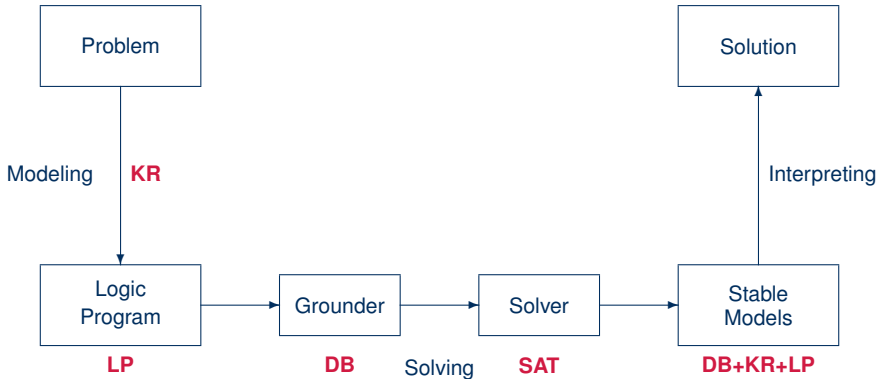
SAT solving



Rooting ASP solving



Rooting ASP solving



Two sides of a coin

- ASP as High-level Language
 - Express problem instance(s) as sets of facts
 - Encode problem (class) as a set of rules
 - Read off solutions from stable models of facts and rules

- ASP as Low-level Language
 - Compile a problem into a logic program
 - Solve the original problem by solving its compilation

What is ASP good for?

- Combinatorial search problems in the realm of P , NP , and NP^{NP} (some with substantial amount of data), like

What is ASP good for?

- Combinatorial search problems in the realm of *P*, *NP*, and *NP^{NP}* (some with substantial amount of data), like
 - Automated Planning
 - Code Optimization
 - Composition of Renaissance Music
 - Database Integration
 - Decision Support for NASA shuttle controllers
 - Model Checking
 - Product Configuration
 - Robotics
 - System Biology
 - System Synthesis
 - (industrial) Team-building
 - and many many more

What does ASP offer?

- Integration of DB, KR, and SAT techniques
- Succinct, elaboration-tolerant 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?

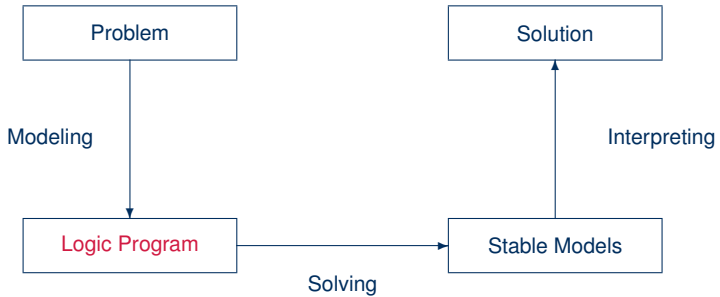
- Integration of DB, KR, and SAT techniques
- Succinct, elaboration-tolerant problem representations
 - Rapid application development tool
- Easy handling of dynamic, knowledge intensive applications
 - including: data, frame axioms, exceptions, defaults, closures, etc

ASP = DB+LP+KR+SAT

Agenda

- 1 Motivation
 - Declarative Problem Solving
 - ASP in a Nutshell
 - ASP Paradigm
- 2 Introduction
 - Syntax
 - Semantics
 - Examples
 - Language Constructs
 - Modeling

Problem solving in ASP: Syntax



Normal logic programs

- A (normal) **logic program** over a set \mathcal{A} of atoms is a finite **set** of rules
- A (normal) **rule**, r , is of the form

$$a_0 \leftarrow a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

Normal logic programs

- A (normal) **logic program** over a set \mathcal{A} of atoms is a finite **set** of rules
- A (normal) **rule**, r , is of the form

$$a_0 \leftarrow a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

- Notation

$$\begin{aligned} \text{head}(r) &= a_0 \\ \text{body}(r) &= \{a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n\} \\ \text{body}(r)^+ &= \{a_1, \dots, a_m\} \\ \text{body}(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

Normal logic programs

- A (normal) **logic program** over a set \mathcal{A} of atoms is a finite **set** of rules
- A (normal) **rule**, r , is of the form

$$a_0 \leftarrow a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

- Notation

$$\begin{aligned} \text{head}(r) &= a_0 \\ \text{body}(r) &= \{a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n\} \\ \text{body}(r)^+ &= \{a_1, \dots, a_m\} \\ \text{body}(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

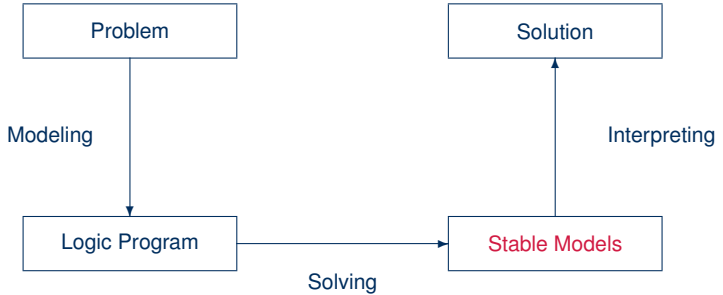
- A program is called **positive** if $\text{body}(r)^- = \emptyset$ for all its rules

Rough notational convention

We sometimes use the following notation interchangeably in order to stress the respective view:

	true, false	if	and	or	iff	default negation	classical negation
source code		<code>:-</code>	<code>,</code>	<code> </code>		<code>not</code>	<code>-</code>
logic program		<code>←</code>	<code>,</code>	<code>;</code>		<i>not</i>	\neg
formula	\perp, \top	\rightarrow	\wedge	\vee	\leftrightarrow	\sim	\neg

Problem solving in ASP: Semantics



Formal Definition

Stable models of positive programs

- A set of atoms X is **closed under** a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
 - X corresponds to a model of P (seen as a formula)

Formal Definition

Stable models of positive programs

- A set of atoms X is **closed under** a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
 - X corresponds to a model of P (seen as a formula)
- The **smallest** set of atoms which is closed under a positive program P is denoted by $Cn(P)$
 - $Cn(P)$ corresponds to the \subseteq -smallest model of P (ditto)

Formal Definition

Stable models of positive programs

- A set of atoms X is **closed under** a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
 - X corresponds to a model of P (seen as a formula)
- The **smallest** set of atoms which is closed under a positive program P is denoted by $Cn(P)$
 - $Cn(P)$ corresponds to the \subseteq -smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the **stable model** of a positive program P

Some “logical” remarks

- Positive rules are also referred to as **definite clauses**
 - Definite clauses are disjunctions with **exactly one** positive atom:

$$a_0 \vee \neg a_1 \vee \dots \vee \neg a_m$$

- A set of definite clauses has a (unique) smallest model

Some “logical” remarks

- Positive rules are also referred to as **definite clauses**
 - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \dots \vee \neg a_m$$

- A set of definite clauses has a (unique) smallest model
- **Horn clauses** are clauses with **at most** one positive atom
 - Every definite clause is a Horn clause but not vice versa
 - Non-definite Horn clauses can be regarded as integrity constraints
 - A set of Horn clauses has a smallest model or none

Some “logical” remarks

- Positive rules are also referred to as definite clauses
 - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \dots \vee \neg a_m$$

- A set of definite clauses has a (unique) **smallest model**
- Horn clauses are clauses with at most one positive atom
 - Every definite clause is a Horn clause but not vice versa
 - Non-definite Horn clauses can be regarded as integrity constraints
 - A set of Horn clauses has a **smallest model** or none
- This **smallest model** is the intended semantics of such sets of clauses
 - Given a positive program P , $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Basic idea

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

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

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

Basic idea

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

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

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

Basic idea

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

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

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



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

Basic idea

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

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

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

Basic idea

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

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

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

$\{p, q\}$

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

Basic idea

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

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

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

$\{p, q\}$

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

$$P_{\Phi} \quad \boxed{\begin{array}{l} q \leftarrow \\ p \leftarrow q, \text{ not } r \end{array}}$$

Basic idea

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

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

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

$$\{p, q\}$$

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

$$P_\Phi \quad \boxed{\begin{array}{l} q \quad \leftarrow \\ p \quad \leftarrow \quad q, \text{ not } r \end{array}}$$

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

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

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

Basic idea

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

$$\{p, q\}$$

$$P_{\Phi} \quad \begin{array}{l} q \leftarrow \\ p \leftarrow q, \text{ not } r \end{array}$$

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

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

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

Formal Definition

Stable model of normal programs

- The **Gelfond-Lifschitz Reduct**[Gelfond and Lifschitz(1991)], P^X , of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

Formal Definition

Stable model of normal programs

- The **Gelfond-Lifschitz Reduct**[Gelfond and Lifschitz(1991)], P^X , of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

- A set X of atoms is a **stable model** of a program P , if $Cn(P^X) = X$

Formal Definition

Stable model of normal programs

- The **Gelfond-Lifschitz Reduct**[Gelfond and Lifschitz(1991)], P^X , of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

- A set X of atoms is a **stable model** of a program P , if $Cn(P^X) = X$
- Note: $Cn(P^X)$ is the \subseteq -smallest (classical) model of P^X
- Note: Every atom in X is justified by an “applying rule from P ”

A closer look at P^X

- In other words, given a set X of atoms from P ,

P^X is obtained from P by **deleting**

- 1 each **rule** having *not* a in its body with $a \in X$
and then
- 2 all **negative atoms** of the form *not* a
in the bodies of the remaining rules

A closer look at P^X

- In other words, given a set X of atoms from P ,

P^X is obtained from P by deleting

- 1 each rule having *not* a in its body with $a \in X$
and then
- 2 all negative atoms of the form *not* a
in the bodies of the remaining rules

- Note: Only **negative body literals** are evaluated w.r.t. X

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X		$Cn(P^X)$
\emptyset		
$\{p\}$		
$\{q\}$		
$\{p, q\}$		

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

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

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow p$ $q \leftarrow$	$\{q\}$ x
$\{p\}$	$p \leftarrow p$	\emptyset
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p, q\}$	$p \leftarrow p$	\emptyset

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow p$ $q \leftarrow$	$\{q\}$ x
$\{p\}$	$p \leftarrow p$	\emptyset x
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p, q\}$	$p \leftarrow p$	\emptyset

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$	
\emptyset	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\}$	$p \leftarrow p$	\emptyset	✗
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✓
$\{p, q\}$	$p \leftarrow p$	\emptyset	

A first example

$$P = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$	
\emptyset	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\}$	$p \leftarrow p$	\emptyset	✗
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✓
$\{p, q\}$	$p \leftarrow p$	\emptyset	✗

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

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

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ x
$\{p\}$	$p \leftarrow$	$\{p\}$
$\{q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		\emptyset

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ ✗
$\{p\}$	$p \leftarrow$	$\{p\}$ ✓
$\{q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		\emptyset

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ ✗
$\{p\}$	$p \leftarrow$	$\{p\}$ ✓
$\{q\}$	$q \leftarrow$	$\{q\}$ ✓
$\{p, q\}$		\emptyset

A second example

$$P = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ ✗
$\{p\}$	$p \leftarrow$	$\{p\}$ ✓
$\{q\}$	$q \leftarrow$	$\{q\}$ ✓
$\{p, q\}$		\emptyset ✗

A third example

$$P = \{p \leftarrow \text{not } p\}$$

A third example

$$P = \{p \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$	$\{p\}$
$\{p\}$		\emptyset

A third example

$$P = \{p \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$
\emptyset	$p \leftarrow$	$\{p\}$ x
$\{p\}$		\emptyset

A third example

$$P = \{p \leftarrow \text{not } p\}$$

X	P^X	$Cn(P^X)$	
\emptyset	$p \leftarrow$	$\{p\}$	x
$\{p\}$		\emptyset	x

Some properties

- A logic program may have zero, one, or multiple stable models!

Some properties

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

Programs with Variables

Let P be a logic program

- Let \mathcal{T} be a set of (variable-free) **terms**
- Let \mathcal{A} be a set of (variable-free) **atoms** constructable from \mathcal{T}

Programs with Variables

Let P be a logic program

- Let \mathcal{T} be a set of variable-free **terms** (also called **Herbrand universe**)
- Let \mathcal{A} be a set of (variable-free) **atoms** constructable from \mathcal{T} (also called **alphabet** or **Herbrand base**)

Programs with Variables

Let P be a logic program

- Let \mathcal{T} be a set of (variable-free) terms
- Let \mathcal{A} be a set of (variable-free) atoms constructable from \mathcal{T}
- **Ground Instances** of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from \mathcal{T} :

$$\text{ground}(r) = \{r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T}, \text{var}(r\theta) = \emptyset\}$$

where $\text{var}(r)$ stands for the set of all variables occurring in r ;
 θ is a (ground) substitution

Programs with Variables

Let P be a logic program

- Let \mathcal{T} be a set of (variable-free) terms
- Let \mathcal{A} be a set of (variable-free) atoms constructable from \mathcal{T}
- Ground Instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from \mathcal{T} :

$$\text{ground}(r) = \{r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T}, \text{var}(r\theta) = \emptyset\}$$

where $\text{var}(r)$ stands for the set of all variables occurring in r ;
 θ is a (ground) substitution

- **Ground Instantiation** of P : $\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$

An example

$$P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \\ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \end{array} \right\}$$

An example

$$P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \\ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \end{array} \right\}$$

$$\text{ground}(P) = \left\{ \begin{array}{l} r(a, b) \leftarrow, \\ r(b, c) \leftarrow, \\ t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), \\ t(a, b) \leftarrow r(a, b), t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), \\ t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) \end{array} \right\}$$

An example

$$P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \\ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \end{array} \right\}$$

$$\text{ground}(P) = \left\{ \begin{array}{l} r(a, b) \leftarrow, \\ r(b, c) \leftarrow, \\ t(a, b) \leftarrow r(a, b), \\ t(b, c) \leftarrow r(b, c), \end{array} \right\}$$

- **Intelligent Grounding** aims at reducing the ground instantiation

Stable models of programs with Variables

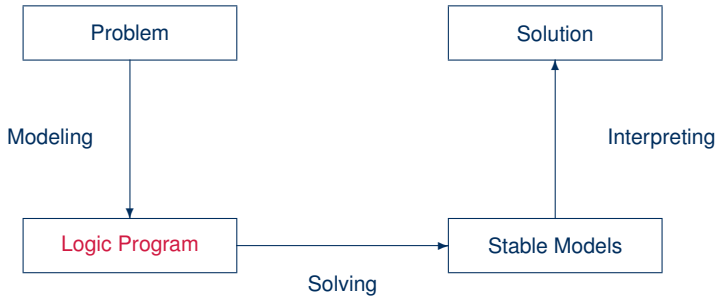
Let P be a normal logic program with variables

Stable models of programs with Variables

Let P be a normal logic program with variables

- A set X of (ground) atoms is a **stable model** of P ,
if $Cn(\mathit{ground}(P)^X) = X$

Problem solving in ASP: Extended Syntax



Language Constructs

Language Constructs

- Variables (over the Herbrand Universe)
 - $p(X) :- q(X)$ over constants $\{a,b,c\}$ stands for
 $p(a) :- q(a), p(b) :- q(b), p(c) :- q(c)$

Language Constructs

- Conditional Literals

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

Language Constructs

- Disjunction

– $p(x) \mid q(x) :- r(x)$

Language Constructs

- Integrity Constraints

- $\text{:- } q(X), p(X)$

Language Constructs

- Choice

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

Language Constructs

- Aggregates

- `s(Y) :- r(Y), 2 #count { p(X,Y) : q(X) } 7`
- also: `#sum, #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
 $p :- q(a), q(b), q(c)$
- **Integrity Constraints**
 - $:- q(X), p(X)$
- **Choice**
 - $2 \{ p(X,Y) : q(X) \} 7 :- r(Y)$
- **Aggregates**
 - $s(Y) :- r(Y), 2 \#count \{ p(X,Y) : q(X) \} 7$
 - **also:** $\#sum, \#avg, \#min, \#max, \#even, \#odd$

Modeling

- For solving a problem class C for a problem instance I , encode
 - 1 the problem instance I as a set P_I of facts and
 - 2 the problem class C as a set P_C of rulessuch that the solutions to C for I can be (polynomially) extracted from the stable models of $P_I \cup P_C$

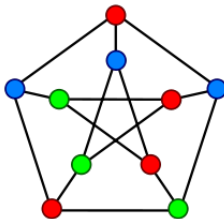
Modeling

- For solving a problem class \mathbf{C} for a problem instance \mathbf{I} , encode
 - 1 the problem instance \mathbf{I} as a set $P_{\mathbf{I}}$ of facts and
 - 2 the problem class \mathbf{C} as a set $P_{\mathbf{C}}$ of rulessuch that the solutions to \mathbf{C} for \mathbf{I} can be (polynomially) extracted from the stable models of $P_{\mathbf{I}} \cup P_{\mathbf{C}}$
- $P_{\mathbf{I}}$ is (still) called **problem instance**
- $P_{\mathbf{C}}$ is often called the **problem encoding**

Modeling

- For solving a problem class \mathbf{C} for a problem instance \mathbf{I} , encode
 - 1 the problem instance \mathbf{I} as a set $P_{\mathbf{I}}$ of facts and
 - 2 the problem class \mathbf{C} as a set $P_{\mathbf{C}}$ of rulessuch that the solutions to \mathbf{C} for \mathbf{I} can be (polynomially) extracted from the stable models of $P_{\mathbf{I}} \cup P_{\mathbf{C}}$
- $P_{\mathbf{I}}$ is (still) called **problem instance**
- $P_{\mathbf{C}}$ is often called the **problem encoding**
- An **encoding** $P_{\mathbf{C}}$ is **uniform**, if it can be used to solve all its problem instances
That is, $P_{\mathbf{C}}$ encodes the solutions to \mathbf{C} for any set $P_{\mathbf{I}}$ of facts

Example 3-Colorability



- Vertices are represented with predicates $\text{node}(X)$;
- Edges are represented with predicates $\text{edge}(X, Y)$.

Question: Is there a valid assignment of three colors for an input graph G such that no two adjacent vertices have the same color?

Graph coloring

node (1..6) .

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) .
```

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) .
```

Graph coloring

```
node (1..6) .
```

```
edge (1,2) .   edge (1,3) .   edge (1,4) .
```

```
edge (2,4) .   edge (2,5) .   edge (2,6) .
```

```
edge (3,1) .   edge (3,4) .   edge (3,5) .
```

```
edge (4,1) .   edge (4,2) .
```

```
edge (5,3) .   edge (5,4) .   edge (5,6) .
```

```
edge (6,2) .   edge (6,3) .   edge (6,5) .
```

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

**Problem
instance**

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).
```


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).
```

} Problem
encoding

Graph coloring

```
node(1..6).
```

```
edge(1,2). edge(1,3). edge(1,4).
```

```
edge(2,4). edge(2,5). edge(2,6).
```

```
edge(3,1). edge(3,4). edge(3,5).
```

```
edge(4,1). edge(4,2).
```

```
edge(5,3). edge(5,4). edge(5,6).
```

```
edge(6,2). edge(6,3). edge(6,5).
```

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

**Problem
instance**

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

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

**Problem
encoding**

color.lp

```
node(1..6).
```

```
edge(1,2). edge(1,3). edge(1,4).
```

```
edge(2,4). edge(2,5). edge(2,6).
```

```
edge(3,1). edge(3,4). edge(3,5).
```

```
edge(4,1). edge(4,2).
```

```
edge(5,3). edge(5,4). edge(5,6).
```

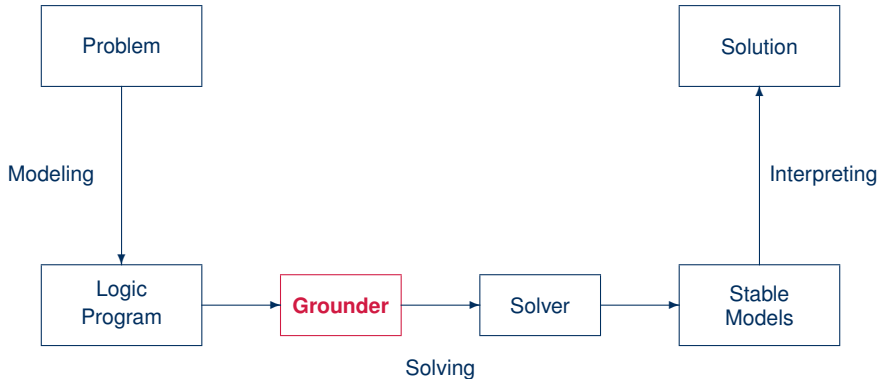
```
edge(6,2). edge(6,3). edge(6,5).
```

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

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

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

ASP solving process



Graph coloring: Grounding

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

Graph coloring: Grounding

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

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

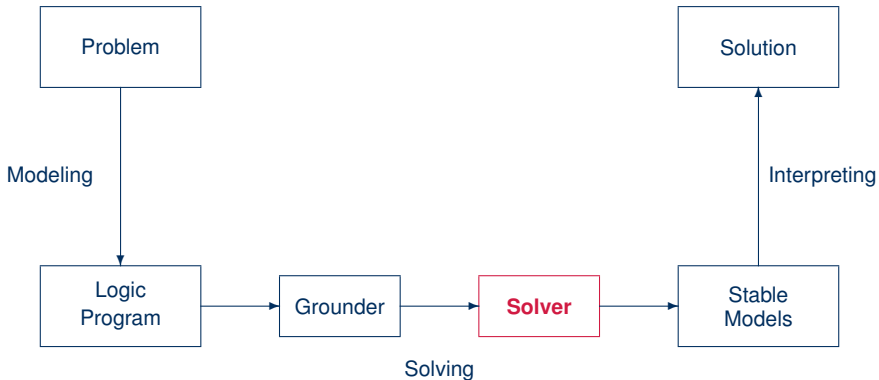
```
edge(1,2). edge(1,3). edge(1,4). edge(2,4). edge(2,5). edge(2,6).  
edge(3,1). edge(3,4). edge(3,5). edge(4,1). edge(4,2). edge(5,3).  
edge(5,4). edge(5,6). edge(6,2). edge(6,3). edge(6,5).
```

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

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

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

ASP solving process



Graph coloring: Solving

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

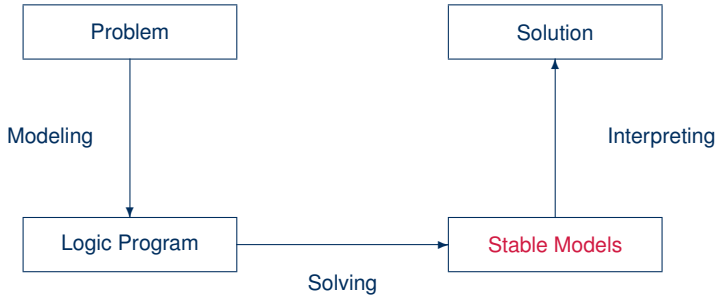
Graph coloring: Solving

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

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

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

Problem solving in ASP: Reasoning Modes



Reasoning Modes

- Satisfiability
- Enumeration[†]
- Projection[†]
- Intersection[‡]
- Union[‡]
- Optimization
- and combinations of them

[†] without solution recording

[‡] without solution enumeration

References



Martin Gebser, Benjamin Kaufmann Roland Kaminski, and Torsten Schaub.

Answer Set Solving in Practice.

Synthesis Lectures on Artificial Intelligence and Machine Learning.

Morgan and Claypool Publishers, 2012.

doi=10.2200/S00457ED1V01Y201211AIM019.



Michael Gelfond and Vladimir Lifschitz.

Classical negation in logic programs and disjunctive databases.

New Generation Comput., 9(3–4):365–386, 1991.

- See also: <http://potassco.sourceforge.net>