Knowledge Representation for the Semantic Web

Lecture 6: Answer Set Programming I

Daria Stepanova

partially based on slides by Thomas Eiter

D5: Databases and Information Systems
Max Planck Institute for Informatics

WS 2017/18

Unit Outline

Introduction

Horn Logic Programming

Negation in Logic Programs

Answer Set Semantics

French Phrases, Italian Soda

- Six people sit at a round table.
- Each drinks a different kind of soda.
- Each plans to visit a different French-speaking country.
- The person who is planning a trip to Quebec, who drank either blueberry or lemon soda, didn't sit in seat number one.
- Jeanne didn't sit next to the person who enjoyed the kiwi soda.
- The person who has a plane ticket to Belgium, who sat in seat four or seat five, didn't order the cherry soda.
- ...

Question:

- *What is each of them drinking, and where is each of them going?*

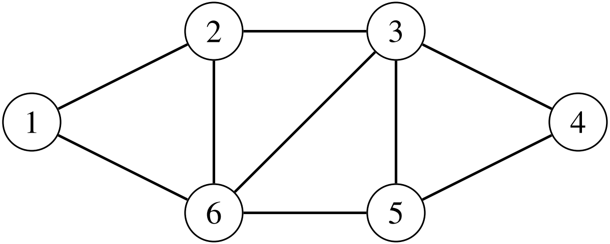
Sudoku

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Task:

Fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9.

Graph 3-colouring



Task:

Colour the nodes of the graph in three colors such that none of the two adjacent nodes share the same colour.

Wanted!

- A general-purpose approach for modeling and solving these and many other problems.
- Issues:
  - Diverse domains
  - Spatial and temporal reasoning
  - Constraints
  - Incomplete information
  - Frame problem
- Proposal:
  - Answer-set programming (ASP) paradigm!

Answer Set Programming

- Answer Set Programming (ASP) is a recent problem solving approach, based on declarative programming.
- The term was coined by Vladimir Lifschitz [1999,2002].
- Proposed by other people at about the same time, e.g., by Marek and Truszczyński [1999] and Niemelä [1999].
- It has roots in knowledge representation, logic programming, and nonmonotonic reasoning.
- At an abstract level, ASP relates to SAT solving and constraint satisfaction problems (CSPs).

Answer Set Programming (cont'd)

- Important logic programming method
- Developed in the early 1990s by Gelfond and Lifschitz.

Left: Michael Gelfond (Texas Tech Univ., Lubbock)
Right: Vladimir Lifschitz (Univ. of Texas, Austin)

- Both are graduates from the Steklov Mathematical Institute, St.Petersburg (then: Leningrad).

Answer Set Programming (cont'd)

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

Answer Set Programming (cont'd)

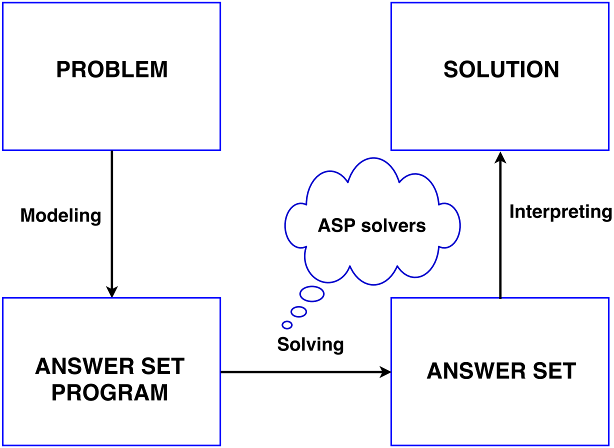
- ASP is an approach to **declarative programming**, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

- ASP has its roots in
  - deductive databases
  - logic programming with negation
  - knowledge representation and nonmonotonic reasoning
  - constraint solving (in particular, SATisfiability testing)

- ASP allows for solving all search problems in NP (and $\mathrm{NP}^{\mathrm{NP}}$) in a uniform way

Declarative Programming

Traditional programming: describe how to solve the problem
Declarative programming: describe what is the problem



Answer Set Programming (cont'd)

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

Nonmonotonic Reasoning

- Nonmonotonicity means that conclusions may be invalidated in the light of new information.
- More specifically, an inference relation $\models$ is *nonmonotonic* if it violates the monotonicity principle:

if $T \models \phi$ and $T \subseteq T'$, then $T' \models \phi$.

- Note: inference in description logics is monotonic.

Example: Monotonicity of description logics

- $T = \{Bird \sqsubseteq Flier, Bird(tweety)\}$
- $T \models Flier(tweety)$
- $T' = T \cup \{\neg Flier(tweety)\}$
- $T' \models Flier(tweety)$ (actually T' is inconsistent)

Nonmonotonic Reasoning

- Nonmonotonicity means that conclusions may be invalidated in the light of new information.
- More specifically, an inference relation $\models$ is *nonmonotonic* if it violates the monotonicity principle:

if $T \models \phi$ and $T \subseteq T'$, then $T' \models \phi$.

- Note: inference in description logics is monotonic.

Example: Nonmonotonic inference

If $bird(x)$ holds and there is no evidence for $\neg flies(x)$, then infer $flies(x)$. I.e., if $bird(x)$, assume $flies(x)$ *by default*.

ASP Systems

ASP gains increasing importance for knowledge representation

- High expressiveness
- Efficient solvers available: DLV, `clasp`, ...

Platform			Features					Mechanics
Name	**OS**	**Licence**	**Variables**	**Function symbols**	**Explicit sets**	**Explicit lists**	**Disjunctive (choice rules) support**	
ASPeRiX	Linux	GPL	Yes				No	on-the-fly grounding
ASSAT	Solaris	Freeware						SAT-solver based
Clasp Answer Set Solver	Linux, macOS, Windows	GPL	Yes, in Clingo	Yes	No	No	Yes	incremental, SAT-solver inspired (nogood, conflict-driven)
Cmodels	Linux, Solaris	GPL	Requires grounding				Yes	incremental, SAT-solver inspired (nogood, conflict-driven)
DLV	Linux, macOS, Windows[14]	free for academic and non-commercial educational use, and for non-profit organizations[14]	Yes	Yes	No	No	Yes	not Lparse compatible
DLV-Complex	Linux, macOS, Windows	GPL		Yes	Yes	Yes	Yes	built on top of DLV — not Lparse compatible
GnT	Linux	GPL	Requires grounding				Yes	built on top of smodels
nomore++	Linux	GPL						combined literal+rule-based
Platypus	Linux, Solaris, Windows	GPL						distributed, multi-threaded nomore++, smodels
Pbmodels	Linux	?						pseudo-boolean solver based
Smodels	Linux, macOS, Windows	GPL	Requires grounding	No	No	No	No	
Smodels-cc	Linux	?	Requires grounding					SAT-solver based; smodels w/conflict clauses
Sup	Linux	?						SAT-solver based

Source: Wikipedia (Dec 6, 2017)

ASP: General Idea

- ASP are logic programs;
- Their semantics adheres to the multiple preferred models approach:
  - given as a selection of the collection of all classical models;
  - selected (intended) models are called stable models or answer sets.

ASP: General Idea

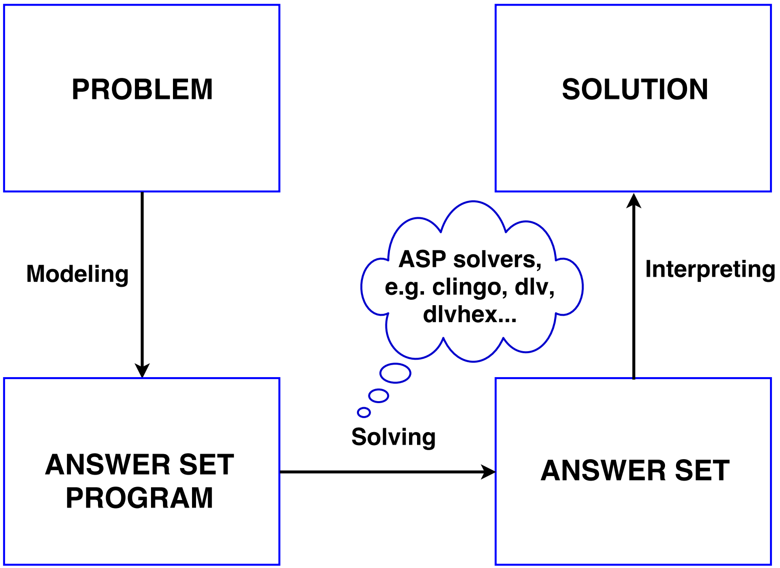
- ASP are logic programs;
- Their semantics adheres to the multiple preferred models approach:
  - given as a selection of the collection of all classical models;
  - selected (intended) models are called stable models or answer sets.

- Fundamental characteristics:
  - models, not proofs, represent solutions;
  - requires techniques to compute models (rather than techniques to compute proofs)

ASP: General Idea (cont'd)

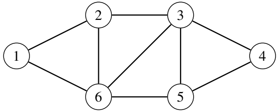
- Given a search problem Π and an instance I, reduce it to the problem of computing intended models of a logic program:
  1. Encode (Π, I) as a logic program P such that the solutions of Π for the instance I are represented by the intended models of P.
  2. Compute some intended model M of P.
  3. Extract a solution for I from M.
- Variant:
  - Compute multiple/all intended models to obtain multiple/all solutions

Example


PROBLEM
SOLUTION
Modeling
ASP solvers,
e.g. clingo, dlv,
dlvhex...
Interpreting
Solving
ANSWER SET
PROGRAM
ANSWER SET

Example

Graph 3-colorability



Modeling

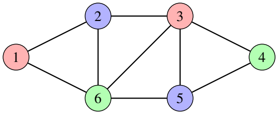
$node(1 \ldots 6); \quad edge(1,2); \quad \ldots$
$col(V, red) \leftarrow not\ \ col(V, blue), not\ \ col(V, green), node(V);$
$col(V, green) \leftarrow not\ \ col(V, blue), not\ \ col(V, red), node(V);$
$col(V, blue) \leftarrow not\ \ col(V, green), not\ \ col(V, red), node(V);$
$\bot \leftarrow col(V, C), col(V, C'), C \neq C';$
$\bot \leftarrow col(V, C), col(V', C), edge(V, V')$

ANSWER SET PROGRAM

Solving

$node(1 \ldots 6); \quad edge(1,2); \ldots$
$col(1, red), col(2, blue),$
$col(3, red), col(4, green),$
$col(6, green), col(5, blue)$

Interpreting



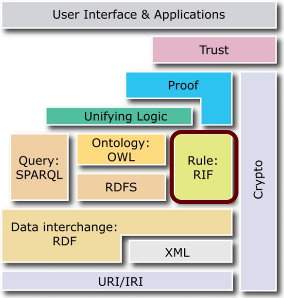
ASP Applications

Use ASP to solve *search problems*, like

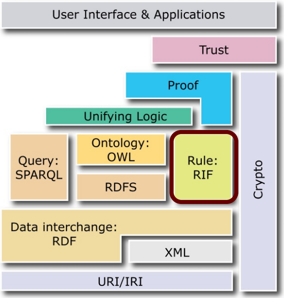
- k-colourability:
  - assign one of k colours to each node of a given graph such that adjacent nodes always have different colours
- Sudoku:
  - find a solution to a given Sudoku puzzle
- Satisfiability (SAT):
  - find all models of a propositional formula
- Time Tabling:
  - find a lecture room assignment for courses

ASP Applications (cont'd)

- Semantic Web



ASP Applications (cont'd)



- Semantic Web
- games, puzzles
- information integration
- constraint satisfaction, configuration
- planning, routing, scheduling
- diagnosis, repair
- security, verification
- systems biology / biomedicine
- knowledge management
- musicology
- ...

  See AI Magazine article on ASP [Erdem *et al.*, 2016] for overview

ASP Applications (cont'd)

- **USA-Advisor** [Nogueira *et al.*, 2001]
  - decision support system to control the Space Shuttle during flight
  - issue: problems with the oxygen transport (pipes and valves)
  - failure scenario: also multiple system failures occur

- **Biological Network Repair** [Kaminski *et al.*, 2013]
  - model nodes (substances, etc) in a large scale biological influence graph, with roles (e.g. inhibitor, activator)
  - repair inconsistencies (modify roles, add links between nodes, etc)

- **Anton** [Boenn *et al.*, 2011] `http://www.cs.bath.ac.uk/~mjb/anton/`
  - automatic system for the composition of renaissance-style music.
  - musical knowledge $\approx$ 500 ASP rules (melody, harmony, rhythm)
  - can generate musical pieces, check pieces for violations.

Horn Logic Programming

Alfred Horn

Syntax

- Assume a vocabulary Φ comprised of nonempty finite sets of
  - constants (e.g., $\mathit{frankfurt}$)
  - variables (e.g., X)
  - predicate symbols (e.g., $\mathit{connected}$)

Syntax

- Assume a vocabulary Φ comprised of nonempty finite sets of
  - constants (e.g., $frankfurt$)
  - variables (e.g., X)
  - predicate symbols (e.g., $connected$)

- A term is either a variable, a constant, or inductively built from other terms using function symbols.

Syntax

- Assume a vocabulary Φ comprised of nonempty finite sets of
  - constants (e.g., $\mathit{frankfurt}$)
  - variables (e.g., X)
  - predicate symbols (e.g., $\mathit{connected}$)

- A term is either a variable, a constant, or inductively built from other terms using function symbols.

- An atom is an expression of form $p(t_1, \ldots, t_n)$, where
  - p is a predicate symbol of arity $n \geq 0$ from Φ, and
  - $t_1, \ldots, t_n$ are terms.

  (e.g., $\mathit{connected}(\mathit{frankfurt})$)

Syntax

- Assume a vocabulary Φ comprised of nonempty finite sets of
  - constants (e.g., $\mathit{frankfurt}$)
  - variables (e.g., X)
  - predicate symbols (e.g., $\mathit{connected}$)

- A term is either a variable, a constant, or inductively built from other terms using function symbols.

- An atom is an expression of form $p(t_1, \ldots, t_n)$, where
  - p is a predicate symbol of arity $n \geq 0$ from Φ, and
  - $t_1, \ldots, t_n$ are terms.

  (e.g., $\mathit{connected}(\mathit{frankfurt})$)

- A term or an atom is *ground* if it contains no variable.
  (e.g., $\mathit{connected}(\mathit{frankfurt})$ is ground, $\mathit{connected}(X)$ is nonground.)

Positive Logic Programs

Def.: Positive logic programs

A positive logic program, P, is a finite set of rules (clauses) of the form

$$a \leftarrow b_1, \ldots, b_m, \tag{1}$$

where a, b_1, ..., b_m are atoms.

- a is the head of the rule
- $b_1, \ldots, b_m$ is the body of the rule.
- If $m = 0$, the rule is a fact (written shortly a)

Intuitively, (1) can be seen as material implication

$$\forall \vec{x}\, b_1 \wedge \cdots \wedge b_m \rightarrow a, \text{ where } \vec{x}$$

is the list of all variables occurring in (1).

Example

- **Ground rule:** *"If Franfurt is a hub airport, and there is a link between Frankfurt and Saarbrücken, then Saarbrücken is a connected airport."*

$$connected(srb) \leftarrow hub\_airport(frankfurt), link(frankfurt, srb)$$

Example

- **Ground rule:** *"If Franfurt is a hub airport, and there is a link between Frankfurt and Saarbrücken, then Saarbrücken is a connected airport."*

$$connected(srb) \leftarrow hub\_airport(frankfurt), link(frankfurt, srb)$$

- **Non-ground rule:** *"All airports with a link to a hub airport are connected."*

$$connected(X) \leftarrow hub\_airport(Y), link(Y, X)$$

can be read as a universally quantified clause

$$\forall X, Y \; hub\_airport(Y) \wedge link(Y, X) \rightarrow connected(X).$$

Herbrand Semantics

Def.: **Herbrand universe, base, interpretation**

- Given a logic program P, the Herbrand universe of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp., the vocabulary Φ of P, if explicitly known).
- The Herbrand base of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.

Herbrand Semantics

Def.: **Herbrand universe, base, interpretation**

- Given a logic program P, the Herbrand universe of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp., the vocabulary Φ of P, if explicitly known).
- The Herbrand base of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.
- A (Herbrand) interpretation is a first-order interpretation $I = (D, \cdot^I)$ of the vocabulary with domain $D = HU(P)$ where each term $t \in HU(P)$ is interpreted by itself, i.e., $t^I = t$.

Herbrand Semantics

Def.: **Herbrand universe, base, interpretation**

- Given a logic program P, the Herbrand universe of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp., the vocabulary Φ of P, if explicitly known).
- The Herbrand base of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.
- A (Herbrand) interpretation is a first-order interpretation $I = (D, \cdot^I)$ of the vocabulary with domain $D = HU(P)$ where each term $t \in HU(P)$ is interpreted by itself, i.e., $t^I = t$.
- I is identified with the set $\{\, p(t_1, \ldots, t_n) \in HB(P) \mid \langle t_1^I, \ldots, t_n^I \rangle \in p^I \,\}$.

Herbrand Semantics

Def.: Herbrand universe, base, interpretation

- Given a logic program P, the Herbrand universe of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp., the vocabulary Φ of P, if explicitly known).
- The Herbrand base of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.
- A (Herbrand) interpretation is a first-order interpretation $I = (D, \cdot^I)$ of the vocabulary with domain $D = HU(P)$ where each term $t \in HU(P)$ is interpreted by itself, i.e., $t^I = t$.
- I is identified with the set $\{ p(t_1, \ldots, t_n) \in HB(P) \mid \langle t_1^I, \ldots, t_n^I \rangle \in p^I \}$.

Informally, a (Herbrand) interpretation can be seen as a set denoting which ground atoms are true in a given scenario.

Herbrand Semantics

Def.: **Herbrand universe, base, interpretation**

- Given a logic program P, the Herbrand universe of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp., the vocabulary Φ of P, if explicitly known).
- The Herbrand base of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.
- A (Herbrand) interpretation is a first-order interpretation $I = (D, \cdot^I)$ of the vocabulary with domain $D = HU(P)$ where each term $t \in HU(P)$ is interpreted by itself, i.e., $t^I = t$.
- I is identified with the set $\{\, p(t_1, \ldots, t_n) \in HB(P) \mid \langle t_1^I, \ldots, t_n^I \rangle \in p^I \,\}$.

Informally, a (Herbrand) interpretation can be seen as a set denoting which ground atoms are true in a given scenario.

Named after logician Jacques Herbrand.

Example

Program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
&\quad p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

Example

Program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
& p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

- Constant symbols: 0, a, b, r.

Example

Program P:

$$p(X,Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$h(X,Z') \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).$$

- Constant symbols: 0, a, b, r.
- Herbrand universe $HU(P)$: $\{0,a,b,r\}$

Example

Program P:

$$p(X,Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$h(X,Z') \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).$$

- Constant symbols: 0, a, b, r.
- Herbrand universe $HU(P)$: $\{0,a,b,r\}$
- Herbrand base $HB(P)$: $\{\ p(0,0,0), p(0,0,a), \ldots, p(r,r,r),$ $h(0,0), h(0,a), \ldots, h(r,r,r),$ $t(0,0,0), t(0,0,a), \ldots, t(r,r,r)\}$

Example

Program P:

$$p(X,Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$h(X,Z') \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).$$

- Constant symbols: 0, a, b, r.
- Herbrand universe $HU(P)$: $\{0,a,b,r\}$
- Herbrand base $HB(P)$: $\{\ p(0,0,0), p(0,0,a), \ldots, p(r,r,r), h(0,0), h(0,a), \ldots, h(r,r,r), t(0,0,0), t(0,0,a), \ldots, t(r,r,r)\}$
- Some Herbrand interpretations:

$$I_1 = \emptyset; \qquad I_2 = HB(P); \qquad I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}.$$

Grounding Example

Program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
p(0,0,b). &\qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

Grounding Example

Program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
& \qquad p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

- The ground instances of the first rule are

$$
\begin{array}{lr}
p(0,0,0) \leftarrow p(0,0,0), h(0,0), t(0,0,r). & X=Y=Z=Z'=0\\
\quad \ldots & \\
p(0,r,0) \leftarrow p(0,r,0), h(0,r), t(0,0,r). & X=Z=Z'=0, Y=r\\
\quad \ldots & \\
p(r,r,r) \leftarrow p(r,r,r), h(r,r), t(r,r,r). & X=Y=Z=Z'=r
\end{array}
$$

- The single ground instance of the last rule is

$$t(a,b,r)$$

Herbrand Models

Def.: Herbrand models

An interpretation I is a (Herbrand) model of

- a ground (variable-free) clause $C = a \leftarrow b_1, \ldots, b_m$, symbolically $I \models C$, if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $a \in I$;
- a clause C, symbolically $I \models C$, if $I \models C'$ for every $C' \in grnd(C)$;
- a program P, symbolically $I \models P$, if $I \models C$ for every clause C in P.

Herbrand Models

Def.: **Herbrand models**

An interpretation I is a (Herbrand) model of

- a ground (variable-free) clause $C = a \leftarrow b_1, \ldots, b_m$, symbolically $I \models C$, if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $a \in I$;
- a clause C, symbolically $I \models C$, if $I \models C'$ for every $C' \in grnd(C)$;
- a program P, symbolically $I \models P$, if $I \models C$ for every clause C in P.

Proposition

For every positive logic program P, $HB(P)$ is a model of P.

Example

Reconsider program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
&p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

Which of the following interpretations are models of P?

- $I_1 = \emptyset$
- $I_2 = HB(P)$
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$

Example

Reconsider program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
p(0,0,b). &\qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

Which of the following interpretations are models of P?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P)$
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$

Example

Reconsider program P:

$$
\begin{aligned}
p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\
&\quad p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).
\end{aligned}
$$

Which of the following interpretations are models of P?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P)$ **yes**
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$

Example

Reconsider program P:

$$p(X,Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$h(X,Z') \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).$$

Which of the following interpretations are models of P?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P)$ **yes**
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$ **no**

Minimal Model Semantics

- A logic program has multiple models in general.
- Select one of these models as the canonical model.
- Commonly accepted: truth of an atom in model I should be "founded" by clauses.

Given:

$$P_1 = \{a \leftarrow b. \quad b \leftarrow c. \quad c\},$$

truth of a in the model $I = \{a, b, c\}$ is "founded".

Given:

$$P_2 = \{a \leftarrow b. \quad b \leftarrow a. \quad c\},$$

truth of a in the model $I = \{a, b, c\}$ is not founded.

Minimal Model Semantics (cont'd)

Semantics follows Occam's razor principle: prefer models with true-part as small as possible.

Def: Minimal models

A model I of P is minimal, if there exists no model J of P such that $J \subset I$.

Minimal Model Semantics (cont'd)

Semantics follows Occam's razor principle: prefer models with true-part as small as possible.

Def: Minimal models

A model I of P is minimal, if there exists no model J of P such that $J \subset I$.

Theorem

Every positive logic program P has a single minimal model (called the least model), denoted $LM(P)$.

Minimal Model Semantics (cont'd)

Semantics follows Occam's razor principle: prefer models with true-part as small as possible.

Def: Minimal models

A model I of P is minimal, if there exists no model J of P such that $J \subset I$.

Theorem

Every positive logic program P has a single minimal model (called the least model), denoted $LM(P)$.

This is a consequence of the following property:

Proposition (Intersection closure)

If I and J are models of a positive program P, then $I \cap J$ is also a model of P.

Example

- For $P_1 = \{\, a \leftarrow b. \quad b \leftarrow c. \quad c \,\}$, we have $LM(P_1) = \{a, b, c\}$.

- For $P_2 = \{\, a \leftarrow b. \quad b \leftarrow a. \quad c \,\}$, we have $LM(P_2) = \{c\}$.

- For P from above,

$$\begin{aligned} p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \\ h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \\ p(0,0,b). \qquad & h(0,0). \qquad t(a,b,r). \end{aligned}$$

  we have

$$LM(P) = \{h(0,0), t(a,b,r), p(0,0,b), p(0,0,a), h(0,b)\}.$$

Negation in Logic Programs

Negation in Logic Programs

Why negation?

- Natural linguistic concept.
- Facilitates convenient, declarative descriptions (definitions).

  E.g., "Men who are not husbands are singles".

Def: Normal logic program

A normal logic program is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (n, m \geq 0) \tag{2}$$

where a and all b_i, c_j are atoms.

The symbol "not" is called negation as failure (or default negation, weak negation).

Programs with Negation

- Prolog: logic-based programming language (developed in the 1970s), with particular algorithm for proving goals (queries) $\langle X \rangle$
- Negation in Prolog: "$not\ \langle X \rangle$" means "negation as failure (to prove) $\langle X \rangle$".
- Closed World Assumption (CWA): whatever cannot be derived is false.

Programs with Negation

- Prolog: logic-based programming language (developed in the 1970s), with particular algorithm for proving goals (queries) $\langle X \rangle$
- Negation in Prolog: "$not\ \langle X \rangle$" means "negation as failure (to prove) $\langle X \rangle$".
- Closed World Assumption (CWA): whatever cannot be derived is false.

  Different from classical negation in first-order logic!

Negation as failure (default negation) not

At a rail road crossing cross the road if **no train is known** to approach

$walk \leftarrow at(X), crossing(X), \textbf{not}\ train\_approaches(X)$

Classical negation $\neg$

At a rail road crossing cross the road if **no train** approaches

$walk \leftarrow at(X), crossing(X), \neg train\_approaches(X)$

Programs with Negation (cont'd)

Example:

$$
\begin{aligned}
man(dilbert).\\
single(X) \leftarrow man(X), not\, husband(X).
\end{aligned}
$$

- Can not prove $husband(dilbert)$ from rules.
- Single intended minimal model: $\{man(dilbert), single(dilbert)\}$.

Programs with Negation (cont'd)

Example:

Modifying the last rule of P_5, let the result be P_1:

$$\begin{aligned} man(dilbert). \\ single(X) \leftarrow man(X), not\, husband(X). \\ husband(X) \leftarrow man(X), not\, single(X). \end{aligned}$$

Semantics???

Problem: not a single intuitive model!

Programs with Negation (cont'd)

Example:

Modifying the last rule of P_5, let the result be P_1:

$$\begin{aligned} man(dilbert).\\ single(X) \leftarrow man(X), not\, husband(X).\\ husband(X) \leftarrow man(X), not\, single(X).\end{aligned}$$

Semantics???

Problem: not a single intuitive model!

Two intuitive Herbrand models:

$$M_1 = \{man(dilbert), single(dilbert)\}, \text{ and}$$
$$M_2 = \{man(dilbert), husband(dilbert)\}.$$

Which one to choose?

Semantics of Negation in Logic Programs

- "War of Semantics" in LP (1980/90ies):
  Meaning of programs like the Dilbert example above

Semantics of Negation in Logic Programs

- "War of Semantics" in LP (1980/90ies):
  Meaning of programs like the Dilbert example above
- Single model vs. multiple model semantics

Semantics of Negation in Logic Programs

- "War of Semantics" in LP (1980/90ies):
  Meaning of programs like the Dilbert example above
- Single model vs. multiple model semantics
- To date:
  - Well-Founded Semantics by Gelder, Ross & Schlipf (1991)

    Partial model: $man(dilbert)$ is true,
    $single(dilbert)$, $husband(dilbert)$ are unknown

Semantics of Negation in Logic Programs

- "War of Semantics" in LP (1980/90ies):
  Meaning of programs like the Dilbert example above

- Single model vs. multiple model semantics

- To date:

  - Well-Founded Semantics by Gelder, Ross & Schlipf (1991)

    Partial model: $man(dilbert)$ is true,
    $single(dilbert)$, $husband(dilbert)$ are unknown

  - Stable Model (alias Answer Set) Semantics
    by Gelfond and Lifschitz (1990)

    Alternative models: $M_1 = \{man(dilbert), single(dilbert)\}$,
    $M_2 = \{man(dilbert), husband(dilbert)\}$.

Semantics of Negation in Logic Programs

- "War of Semantics" in LP (1980/90ies):
  Meaning of programs like the Dilbert example above
- Single model vs. multiple model semantics
- To date:
  - Well-Founded Semantics by Gelder, Ross & Schlipf (1991)

    Partial model: $man(dilbert)$ is true,
    $single(dilbert)$, $husband(dilbert)$ are unknown

  - **Stable Model (alias Answer Set) Semantics**
    by Gelfond and Lifschitz (1990)

    Alternative models: $M_1 = \{man(dilbert), single(dilbert)\}$,
    $M_2 = \{man(dilbert), husband(dilbert)\}$.

Stable Models: Intuition

Consider program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\, husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\, single(dilbert). \qquad (r_2)$$

Stable Models: Intuition

Consider program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\, husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\, single(dilbert). \qquad (r_2)$$

- Consider $M' = \{man(dilbert)\}$.
  - Assuming that $man(dilbert)$ is true and $husband(dilbert)$ is false, by r_1 also $single(dilbert)$ should be true.
  - M' does not represent a coherent or "stable" view of the information given by P_1.

Stable Models: Intuition

Consider program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\, husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\, single(dilbert). \qquad (r_2)$$

- Consider $M' = \{man(dilbert)\}$.
  - Assuming that $man(dilbert)$ is true and $husband(dilbert)$ is false, by r_1 also $single(dilbert)$ should be true.
  - M' does not represent a coherent or "stable" view of the information given by P_1.
- Consider $M'' = \{man(dilbert), single(dilbert), husband(dilbert)\}$.
  - The bodies of r_1 and r_2 are not true w.r.t. M'', hence there is no evidence for $single(dilbert)$ and $husband(dilbert)$ being true.
  - M'' is not "stable" either.

Stable Models

Def: **Gelfond-Lifschitz reduct, stable models, answer sets**

- The GL-reduct (or simply reduct) of a ground program P w.r.t. an interpretation M, denoted P^M, is the program obtained from P by performing the following two steps:

  1. remove all rules with some $not\, a$ in its body s.t. $a \in M$; and
  2. remove all default negated literals from the remaining rules.

- An interpretation M of P is a stable model (or answer set) of P if

$$M = LM(P^M).$$

Stable Models (cont'd)

Intuition behind GL-reduct:

- M makes an assumption about what is true and what is false.
- The GL-reduct P^M incorporates this assumption.
- As a "not"-free program, P^M derives positive facts, given by the least model $LM(P^M)$.
- If this coincides with M, then the assumption of M is "stable".

Observe:

- $P^M = P$ for any "not"-free program P.
- For any positive program P, $LM(P)$ ($=LM(P^M)$) is its single stable model.

Example

Consider again the grounding of P_1:

$$man(dilbert). \qquad (f_1)$$
$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$
$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Candidate interpretations:

- $M_1 = \{man(dilbert), single(dilbert)\}$,
- $M_2 = \{man(dilbert), husband(dilbert)\}$,
- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$,
- $M_4 = \{man(dilbert)\}$.

Example

Consider again the grounding of P_1:

$$man(dilbert). \qquad (f_1)$$
$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$
$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Candidate interpretations:

- $M_1 = \{man(dilbert), single(dilbert)\}$,
- $M_2 = \{man(dilbert), husband(dilbert)\}$,
- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$,
- $M_4 = \{man(dilbert)\}$.

M_1 and M_2 are stable models.

Example (cont'd)

Recall the program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Consider $M_1 = \{man(dilbert), single(dilbert)\}$:

Example (cont'd)

Recall the program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Consider $M_1 = \{man(dilbert), single(dilbert)\}$:

GL-reduct $P_1^{M_1}$ of M_1 is as follows:

$$man(dilbert).$$

$$single(dilbert) \leftarrow man(dilbert).$$

Example (cont'd)

Recall the program P_1:

$$man(dilbert). \qquad (f_1)$$
$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$
$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Consider $M_1 = \{man(dilbert), single(dilbert)\}$:

GL-reduct $P_1^{M_1}$ of M_1 is as follows:

$$man(dilbert).$$
$$single(dilbert) \leftarrow man(dilbert).$$

The least model of $P_1^{M_1}$ is $\{man(dilbert), single(dilbert)\} = M_1$.

Example (cont'd)

Recall the program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\, husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\, single(dilbert). \qquad (r_2)$$

Consider $M_1 = \{man(dilbert), single(dilbert)\}$:

GL-reduct $P_1^{M_1}$ of M_1 is as follows:

$$man(dilbert).$$

$$single(dilbert) \leftarrow man(dilbert).$$

The least model of $P_1^{M_1}$ is $\{man(dilbert), single(dilbert)\} = M_1$.

By symmetry of $husband$ and $single$, also $M_2 = \{man(dilbert), husband(dilbert)\}$ is stable.

Summary

1. Introduction and background
2. Horn logic programming
   - Positive logic programs
   - Minimal model semantics
3. Negation in logic programs
   - Negation in prolog
   - Semantics of negation in logic programs
4. Answer-Set semantics
   - Semantic properties of stable models
   - Computational properties

References I

Georg Boenn, Martin Brain, Marina De Vos, and John ffitch.
Anton - A rule-based composition system.
In *Proceedings of the 2011 International Computer Music Conference, ICMC 2011, Huddersfield, UK, July 31 - August 5, 2011*. Michigan Publishing, 2011.

Esra Erdem, Michael Gelfond, and Nicola Leone.
Applications of answer set programming.
AI Magazine, 37(3):53–68, 2016.

Roland Kaminski, Torsten Schaub, Anne Siegel, and Santiago Videla.
Minimal intervention strategies in logical signaling networks with ASP.
TPLP, 13(4-5):675–690, 2013.

References II

Vladimir Lifschitz.
Answer set planning.
In *Proc. 16th International Conference on Logic Programming (ICLP),*, pages 23–37, 1999.

Vladimir Lifschitz.
Answer Set Programming and Plan Generation.
Artificial Intelligence, 138:39–54, 2002.

Victor W. Marek and Mirosław Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
In K. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.

References III

Ilkka Niemelä.

Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.

Annals of Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry.

An a-prolog decision support system for the space shuttle.

In I. V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001, Las Vegas, Nevada, March 11-12, 2001, Proceedings*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001.