Knowledge Representation for the Semantic Web

Answer Set Programming – Unit 1

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
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?
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 Truszczynski [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 $\text{NP}$ (and $\text{NP}^\text{NP}$) 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 $\text{NP}$ (and $\text{NP}^{\text{NP}}$) in a uniform way
Declarative Programming

**Traditional programming**: describe how to solve the problem

**Declarative programming**: describe what is the problem

- **MODELING**
  - PROBLEM
  - ANSWER SET PROGRAM
- **SOLVING**
  - ASP solvers
  - Solving
- **INTERPRETING**
  - SOLUTION
  - ANSWER SET
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, **SAT**isfiability testing)

• **ASP** allows for solving all **search problems** in **NP** (and **NP**$^N$P) 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:

$$\text{if } T \models \phi \text{ and } T \subseteq T', \text{ then } T' \not\models \phi.$$ 

- Note: inference in description logics is monotonic.

Example: Monotonicity of description logics

- $T = \{\text{Bird} \sqsubseteq \text{Flier}, \text{Bird(tweety)}\}$
- $T \models \text{Flier(tweety)}$
- $T' = T \cup \{\neg \text{Flier(tweety)}\}$
- $T' \models \text{Flier(tweety)}$ (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:

  \[
  \text{if } T \models \phi \text{ and } T \subseteq T', \text{ then } T' \models \phi.
  \]

• 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, ...

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Variables</th>
<th>Function symbols</th>
<th>Explicit sets</th>
<th>Explicit lists</th>
<th>Disjunctive (choice rules) support</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPeRX</td>
<td>Linux/GPL</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>on-the-fly grounding</td>
<td>on-the-fly grounding</td>
</tr>
<tr>
<td>ASSAT</td>
<td>Solaris/Freeware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SAT-solver based</td>
<td>SAT-solver based</td>
</tr>
<tr>
<td>Clasp Answer Set Solver</td>
<td>Linux/macOS, Windows/GPL</td>
<td>Yes, in Clingo</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>incremental, SAT-solver inspired (nogood, conflict-driven)</td>
</tr>
<tr>
<td>Cmodels</td>
<td>Linux/Solaris</td>
<td>Requires grounding</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>incremental, SAT-solver inspired (nogood, conflict-driven)</td>
</tr>
<tr>
<td>DLV</td>
<td>Linux/macOS, Windows/GPL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>not transparent compatible</td>
</tr>
<tr>
<td>DLV-Complex</td>
<td>Linux/macOS, Windows/GPL</td>
<td>Requires grounding</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>built on top of DLV — not Lparse compatible</td>
</tr>
<tr>
<td>Gt</td>
<td>Linux/GPL</td>
<td>Requires grounding</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>built on top of smodels</td>
</tr>
<tr>
<td>Nmore++</td>
<td>Linux/GPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>combined literal+rule-based</td>
</tr>
<tr>
<td>Platypus</td>
<td>Linux/Solaris, Windows/GPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>distributed, multi-threaded</td>
</tr>
<tr>
<td>Pbmodels</td>
<td>Linux/Unknown</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pseudo-boolean solver based</td>
</tr>
<tr>
<td>Smodels</td>
<td>Linux/macOS, Windows/GPL</td>
<td>Requires grounding</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>SAT-solver based; smodels w/conflict clauses</td>
</tr>
<tr>
<td>Smodels-cc</td>
<td>Linux/Unknown</td>
<td>Requires grounding</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>SAT-solver based</td>
</tr>
<tr>
<td>Sup</td>
<td>Linux/Unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $\Pi$ and an instance $I$, reduce it to the problem of computing intended models of a logic program:

  1. Encode $(\Pi, I)$ as a logic program $P$ such that the solutions of $\Pi$ 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

Modeling

SOLUTION

ASP solvers, e.g. clingo, dlv, dlvhex...

Interpreting

ANSWER SET PROGRAM

Solving

ANSWER SET
Introduction

Horn Logic Programming

Negation in Logic Programs

---

Example

Graph 3-colorability

Modeling

\[
\begin{align*}
\text{node}(1 \ldots 6); & \quad \text{edge}(1, 2); \\
\text{col}(V, \text{red}) \leftarrow \neg \text{col}(V, \text{blue}), \neg \text{col}(V, \text{green}), \text{node}(V); & \\
\text{col}(V, \text{green}) \leftarrow \neg \text{col}(V, \text{blue}), \neg \text{col}(V, \text{red}), \text{node}(V); & \\
\text{col}(V, \text{blue}) \leftarrow \neg \text{col}(V, \text{green}), \neg \text{col}(V, \text{red}), \text{node}(V); & \\
\bot \leftarrow \text{col}(V, \text{C}), \text{col}(V', \text{C}'), \text{C} \neq \text{C'}; & \\
\bot \leftarrow \text{col}(V, \text{C}), \text{col}(V', \text{C}), \text{edge}(V, V') &
\end{align*}
\]

Interpreting

Solving

\[
\begin{align*}
\text{node}(1 \ldots 6); & \quad \text{edge}(1, 2); \ldots \\
\text{col}(1, \text{red}), \text{col}(2, \text{blue}), & \\
\text{col}(3, \text{red}), \text{col}(4, \text{green}), & \\
\text{col}(6, \text{green}), \text{col}(5, \text{blue})
\end{align*}
\]
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/](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 $\Phi$ comprised of nonempty finite sets of
  - constants (e.g., $\text{frankfurt}$)
  - variables (e.g., $X$)
  - predicate symbols (e.g., $\text{connected}$)
Syntax

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

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

- Assume a vocabulary $\Phi$ 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.

- 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 $\Phi$, and
  - $t_1, \ldots, t_n$ are terms.

  (e.g., $connected(frankfurt)$)
Syntax

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

• A \textit{term} is either a variable, a constant, or inductively built from other terms using function symbols.

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

  (e.g., \textit{connected(frankfurt)})

• A \textit{term} or an \textit{atom} is \textit{ground} if it contains no variable.
  (e.g., \textit{connected(frankfurt)} is ground, \textit{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

$$ a \leftarrow b_1, \ldots, b_m, \quad (1) $$

where $a, b_1, \ldots, b_m$ 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

$$ \forall \bar{x} \ b_1 \land \cdots \land b_m \rightarrow a, \text{ where } \bar{x} $$

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

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

  \[
  \text{connected}(\text{srb}) \leftarrow \text{hub\_airport}(\text{frankfurt}), \text{link}(\text{frankfurt, srb})
  \]
Example

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

  \[
  \text{connected}(srb) \leftarrow \text{hub\_airport}(frankfurt), \text{link}(frankfurt, srb)
  \]

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

  \[
  \text{connected}(X) \leftarrow \text{hub\_airport}(Y), \text{link}(Y, X)
  \]

  can be read as a universally quantified clause

  \[
  \forall X, Y \text{ hub\_airport}(Y) \land \text{link}(Y, X) \rightarrow \text{connected}(X).
  \]
Herbrand Semantics

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

- Given a logic program \( P \), the **Herbrand universe** of \( P \), \( \text{HU}(P) \), is the set of all terms which can be formed from constants and functions symbols in \( P \) (resp., the vocabulary \( \Phi \) of \( P \), if explicitly known).

- The **Herbrand base** of \( P \), \( \text{HB}(P) \), is the set of all ground atoms which can be formed from predicates and terms \( t \in \text{HU}(P) \).
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 $\Phi$ 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., $t^I = t$. 

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.
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 $\Phi$ 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.
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 $\Phi$ 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 $\{ p(t_1, \ldots, t_n) \in HB(P) \mid \langle t^I_1, \ldots, t^I_n \rangle \in p^I \}$.

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 \( \Phi \) 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 \( \{ p(t_1, \ldots, t_n) \in HB(P) \mid \langle t^I_1, \ldots, t^I_n \rangle \in p^I \} \).

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 $P$:

\[
p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\]
Example

Program $P$:

\[
\begin{align*}
p(X, Y, Z) & \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
h(X, Z') & \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
p(0, 0, b). & \quad h(0, 0). \quad t(a, b, r).
\end{align*}
\]

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

Program $P$:

\[
\begin{align*}
p(X, Y, Z) & \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
h(X, Z') & \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
p(0, 0, b). & \quad h(0, 0). \quad t(a, b, r).
\end{align*}
\]

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

Program $P$:

\[
p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\]

- 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), \quad t(0, 0, 0), t(0, 0, a), \ldots, t(r, r, r)\}\}
**Example**

**Program** $P$:

\[ p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \]
\[ h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \]
\[ p(0, 0, b). \quad h(0, 0). \quad t(a, b, r). \]

- 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:
  \[ I_1 = \emptyset; \quad I_2 = HB(P); \quad I_3 = \{ h(0, 0), t(a, b, r), p(0, 0, b) \}. \]
Grounding Example

Program $P$: 

\[
p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\]
Grounding Example

Program $P$:

\begin{align*}
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). & \quad h(0, 0). \quad t(a, b, r). 
\end{align*}

- The ground instances of the first rule are

\begin{align*}
p(0, 0, 0) & \leftarrow p(0, 0, 0), h(0, 0), t(0, 0, r). \quad X = Y = Z = Z' = 0 \\
\vdots \\
p(0, r, 0) & \leftarrow p(0, r, 0), h(0, r), t(0, 0, r). \quad X = Z = Z' = 0, Y = r \\
\vdots \\
p(r, r, r) & \leftarrow p(r, r, r), h(r, r), t(r, r, r). \quad X = Y = Z = Z' = r 
\end{align*}

- The single ground instance of the last rule is

$t(a, b, r)$
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 $C' \in \text{grnd}(C)$;

- a program $P$, symbolically $I \models P$, if $I \models C$ for every clause $C$ in $P$. 

Proposition
For every positive logic program $P$, $\text{HB}(P)$ is a model of $P$. 

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 $C' \in \text{grnd}(C)$;

- 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 $P$:

$p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r)$.

$h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r)$.

$p(0, 0, b)$. $h(0, 0)$. $t(a, b, r)$.

Which of the following interpretations are models of $P$?

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

\[
p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]

\[
h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]

\[
p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\]

Which of the following interpretations are models of $P$?

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

Reconsider program $P$:

\[ p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \]
\[ h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \]
\[ p(0, 0, b). \quad h(0, 0). \quad t(a, b, r). \]

Which of the following interpretations are models of $P$?

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

Reconsider program $P$:

\[
p(X, Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
h(X, Z') \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).
\]
\[
p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\]

Which of the following interpretations are models of $P$?

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

Horn Logic Programming

Negation in Logic Programs

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:**

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

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

**Given:**

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

truth of $a$ in the model $I = \{a, b, c\}$ is not founded.
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 $P_1 = \{ a \leftarrow b. \ b \leftarrow c. \ c \}$, we have $LM(P_1) = \{a, b, c\}$.

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

- For $P$ from above,

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

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

\[ a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \quad (n, m \geq 0) \quad (2) \]

where \( a \) and all \( b_i, c_j \) are atoms.

The symbol “\( \text{not} \)” 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: “\( \text{not} \langle X \rangle \)” 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) \( \text{not} \)**

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

\[
\text{walk} \leftarrow \text{at}(X), \text{crossing}(X), \text{not train_approaches}(X)
\]

**Classical negation \( \neg \)**

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

\[
\text{walk} \leftarrow \text{at}(X), \text{crossing}(X), \neg \text{train_approaches}(X)
\]
Example:

\[ \text{man}(\text{dilbert}). \]
\[ \text{single}(X) \leftarrow \text{man}(X), \text{not husband}(X). \]

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

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

$$\text{man} (\text{dilbert}).$$

$$\text{single}(X) \leftarrow \text{man}(X), \text{not husband}(X).$$

$$\text{husband}(X) \leftarrow \text{man}(X), \text{not single}(X).$$

Semantics???

**Problem**: not a single intuitive model!
Example:

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

\[
\text{man(}d\text{ilbert)}.
\]
\[
\text{single}(X) \leftarrow \text{man}(X), \text{not husband}(X).
\]
\[
\text{husband}(X) \leftarrow \text{man}(X), \text{not single}(X).
\]

Semantics???

**Problem**: not a single intuitive model!

Two intuitive Herbrand models:

\[
M_1 = \{ \text{man(}d\text{ilbert)}, \text{single(}d\text{ilbert)} \}, \quad \text{and}
\]
\[
M_2 = \{ \text{man(}d\text{ilbert)}, \text{husband(}d\text{ilbert)} \}.
\]

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: \( \text{man}(\text{dilbert}) \) is true,
    - \( \text{single}(\text{dilbert}), \text{husband}(\text{dilbert}) \) 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:  
    \[
    \text{man}(\text{dilbert}) \text{ is true,} \\
    \text{single}(\text{dilbert}), \text{husband}(\text{dilbert}) \text{ are unknown}
    \]
  
  - **Stable Model (alias Answer Set) Semantics**
    by Gelfond and Lifschitz (1990)
    Alternative models:  
    \[
    M_1 = \{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}) \}, \\
    M_2 = \{ \text{man}(\text{dilbert}), \text{husband}(\text{dilbert}) \}.
    \]
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: \( \text{man}(\text{dilbert}) \) is true, \( \text{single}(\text{dilbert}), \text{husband}(\text{dilbert}) \) are unknown

  - **Stable Model (alias Answer Set) Semantics** by Gelfond and Lifschitz (1990)
    Alternative models: \( M_1 = \{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}) \} \), \( M_2 = \{ \text{man}(\text{dilbert}), \text{husband}(\text{dilbert}) \} \).
Consider program $P_1$:

$$\begin{align*}
\text{man(dilbert).} & \quad (f_1) \\
\text{single(dilbert) ← man(dilbert), not husband(dilbert).} & \quad (r_1) \\
\text{husband(dilbert) ← man(dilbert), not single(dilbert).} & \quad (r_2)
\end{align*}$$
Stable Models: Intuition

Consider program $P_1$:

\[
\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{husband}(\text{dilbert}). & \quad (r_1) \\
\text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{single}(\text{dilbert}). & \quad (r_2)
\end{align*}
\]

- Consider $M' = \{\text{man}(\text{dilbert})\}$.

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

Stable Models: Intuition

Consider program $P_1$:

\[
\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{husband}(\text{dilbert}). & \quad (r_1) \\
\text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{single}(\text{dilbert}). & \quad (r_2)
\end{align*}
\]

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

- Consider $M'' = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert}), \text{husband}(\text{dilbert})\}$.
  - The bodies of $r_1$ and $r_2$ are not true w.r.t. $M''$, hence there is no evidence for $\text{single}(\text{dilbert})$ and $\text{husband}(\text{dilbert})$ being true.
  - $M''$ is not “stable” either.
Stable Models

**Def:** Gelfond-Lifschitz reduct, stable models 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 $\textit{not } a$ 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 $P_1$:

$$\text{man}(\text{dilbert}). \quad (f_1)$$
$$\text{single}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}). \quad (r_1)$$
$$\text{husband}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}). \quad (r_2)$$

Candidate interpretations:

- $M_1 = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert})\}$,
- $M_2 = \{\text{man}(\text{dilbert}), \text{husband}(\text{dilbert})\}$,
- $M_3 = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert}), \text{husband}(\text{dilbert})\}$,
- $M_4 = \{\text{man}(\text{dilbert})\}$. 
Consider again the grounding of $P_1$:

\begin{align*}
  \text{man}(\text{dilbert}). & \quad (f_1) \\
  \text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}). & \quad (r_1) \\
  \text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}). & \quad (r_2)
\end{align*}

Candidate interpretations:

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

$M_1$ and $M_2$ are stable models.
Example (cont’d)

Recall the program $P_1$:

\[
\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}). & \quad (r_1) \\
\text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}). & \quad (r_2)
\end{align*}
\]

Consider $M_1 = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert})\}$:
Example (cont’d)

Recall the program $P_1$:

\[
\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}). & \quad (r_1) \\
\text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}). & \quad (r_2)
\end{align*}
\]

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

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

\[
\begin{align*}
\text{man}(\text{dilbert}). \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}).
\end{align*}
\]
Example (cont’d)

Recall the program $P_1$:

\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{husband}(\text{dilbert}). & (r_1) \\
\text{husband}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}), \neg \text{single}(\text{dilbert}). & (r_2)
\end{align*}

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

GL-reduct $P_{1}^{M_1}$ of $M_1$ is as follows:

\begin{align*}
\text{man}(\text{dilbert}). \\
\text{single}(\text{dilbert}) & \leftarrow \text{man}(\text{dilbert}).
\end{align*}

The least model of $P_{1}^{M_1}$ is $\{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}) \} = M_1$. 
Example (cont’d)

Recall the program $P_1$:

\[
\begin{align*}
\text{man}(\text{dilbert}). & \quad (f_1) \\
\text{single}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}). & \quad (r_1) \\
\text{husband}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}). & \quad (r_2)
\end{align*}
\]

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

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

\[
\begin{align*}
\text{man}(\text{dilbert}). \\
\text{single}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}).
\end{align*}
\]

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

By symmetry of $\text{husband}$ and $\text{single}$, also $M_2 = \{\text{man}(\text{dilbert}), \text{husband}(\text{dilbert})\}$ is stable.
Georg Boenn, Martin Brain, Marina De Vos, and John ffitch.  
Anton - A rule-based composition system.  

Esra Erdem, Michael Gelfond, and Nicola Leone.  
Applications of answer set programming.  

Roland Kaminski, Torsten Schaub, Anne Siegel, and Santiago Videla.  
Minimal intervention strategies in logical signaling networks with ASP.  
Vladimir Lifschitz.

Answer set planning.


Vladimir Lifschitz.

Answer Set Programming and Plan Generation.


Victor W. Marek and Mirosław Truszczynski.

Stable Models and an Alternative Logic Programming Paradigm.

Ilkka Niemelä.

Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.


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

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