# Answer Set Programming: A Primer


Thomas Eiter, Thomas Krennwallner (TU Wien, Austria)
Giovambattista Ianni (Università della Calabria, Italy)

Supported by Austrian Science Fund (FWF) project P20840 & P20841, the EC ICT Integrated Project Ontorule (FP7 231875), and the Italian National Project Interlink II04CG8AGG.

# Unit 1 – Basic Concepts & Properties

Thomas Eiter

KBS Group, Institute of Information Systems, TU Vienna

Reasoning Web Summer School 2009

# Unit Outline

# Introduction

- Answer Set Programming (ASP) is a recent problem solving approach
- The term was coined by Vladimir Lifschitz [1999,2002]
- Proposed by other people at about the same time, e.g. [Marek and Truszczyński, 1999],[Niemelä, 1999]
- It has roots in KR, logic programming, and nonmonotonic reasoning
- At an abstract level, relates to SAT solving and CSP.
- Book: [Baral, 2003]

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- Six people sit at a round table

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- Six people sit at a round table
- Each drinks a different kind of soda

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- Six people sit at a round table
- Each drinks a different kind of soda
- Each plans to visit a different French-speaking country

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- 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.

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- 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.

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- 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 tangelo soda.

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- 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 tangelo soda.
- ...

# French Phrases, Italian Soda (*Dell Logic Puzzles*)

- 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 tangelo 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.

# Wanted!

A general-purpose approach for modeling and solving these and many other problems.

## Issues:

- Diverse domains
- Spatial and temporal reasoning
- Constraints
- Incomplete information
- Preferences and priority

# Wanted!

A general-purpose approach for modeling and solving these and many other problems.

## Issues:

- Diverse domains
- Spatial and temporal reasoning
- Constraints
- Incomplete information
- Preferences and priority

## Proposal:

Answer Set Programming (ASP) paradigm!

# Roots of ASP – Knowledge Representation (KR)

## How to model

- An agent's belief sets
- Commonsense reasoning
- Defeasible inferences
- Preferences and priority

# Roots of ASP – Knowledge Representation (KR)

## How to model

- An agent's belief sets
- Commonsense reasoning
- Defeasible inferences
- Preferences and priority

## Approach

- use a logic-based formalism
- Inherent feature: nonmonotonicity

## Many logic-based KR formalisms have been developed.

# Logic Programming – Prolog revisited

1960s/70s: Logic as a Programming Language (?)

- Breakthrough in Computational Logic by Robinson's discovery of the Resolution Principle (1965)

## Kowalski (1979):

**ALGORITHM = LOGIC + CONTROL**

- Knowledge for problem solving (LOGIC)
- "Processing" of the knowledge (CONTROL)

# Prolog

**Prolog = "Programming in Logic"**

- Basic data structures: terms
- Programs: rules and facts
- Computing: Queries (goals)
  - Proofs provide answers
  - SLD-resolution
  - unification - basic mechanism to manipulate data structures
- Extensive use of recursion

## Example (Recursion)

```
append([],X,X) .
append([X|Y],Z,[X|T]) :- append(Y,Z,T) .

reverse([],[]).
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

- Both relations are defined recursively.
- Terms represent complex objects: lists, sets, ...

## Example (Recursion)

```
append([],X,X) .
append([X|Y],Z,[X|T]) :- append(Y,Z,T) .

reverse([],[]).
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

- Both relations are defined recursively.
- Terms represent complex objects: lists, sets, ...

## Problem:

Reverse the list `[a,b,c]`

## Example (Recursion)

```
append([],X,X) .
append([X|Y],Z,[X|T]) :- append(Y,Z,T) .

reverse([],[]).
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

- Both relations are defined recursively.
- Terms represent complex objects: lists, sets, ...

## Problem:

Reverse the list `[a,b,c]`

## Ask query: `?- reverse([a,b,c],X).`

## Example (Recursion)

```
append([],X,X) .
append([X|Y],Z,[X|T]) :- append(Y,Z,T) .

reverse([],[]).
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

- Both relations are defined recursively.
- Terms represent complex objects: lists, sets, ...

## Problem:

Reverse the list `[a,b,c]`

## Ask query: `?- reverse([a,b,c],X).`

- A proof of the query yields a substitution: `X=[c,b,a]`
- The substitution constitutes an answer

**The key:** Techniques to search for proofs

- Understanding of the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

**The key:** Techniques to search for proofs

- Understanding of the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

## Example

```
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

vs

```
reverse([X|Y],Z) :- reverse(Y,U), append(U,[X],Z) .
```

Query: `?- reverse([a|X],[b,c,d,b])`

**The key:** Techniques to search for proofs

- Understanding of the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

## Example

```
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

vs

```
reverse([X|Y],Z) :- reverse(Y,U), append(U,[X],Z) .
```

Query: `?- reverse([a|X],[b,c,d,b])`

**Is this truly declarative programming?**

# Desiderata

Relieve the programmer from several concerns.

It is desirable that

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;
- termination is not subject to such order.

# Desiderata

Relieve the programmer from several concerns.

It is desirable that

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;
- termination is not subject to such order.

**“Pure” declarative programming**

- Prolog does not satisfy these desiderata
- Satisfied e.g. by the answer set semantics of logic programs

# Positive Logic Programs

## Definition (Positive Logic Program)

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

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

where $a$, $b_1$, ..., $b_m$ are atoms of a first-order language $L$.

- $a$ 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 $a$)

Roughly, (1) can be seen as material implication $b_1 \wedge \cdots \wedge b_m \supset a$.

# Positive Logic Programs

## Definition (Positive Logic Program)

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

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

where $a$, $b_1$, ..., $b_m$ are atoms of a first-order language $L$.

- $a$ 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 $a$)

Roughly, (1) can be seen as material implication $b_1 \wedge \cdots \wedge b_m \supset a$.

## Example

$$connected(cagliari) \leftarrow hub(rome), link(rome, cagliari)$$
$$connected(X) \leftarrow hub(Y), link(Y, X)$$

# Herbrand Semantics

## Definition (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, 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.

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

$HU(P_1)$: { 0, $a$, $b$, $r$, $f(0)$, $f(f(0))$, ... $f^i(0))$, .... $f(a)$, $f(f(a))$, ... }

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

$HU(P_1)$: { 0, $a$, $b$, $r$, $f(0), f(f(0)),$ ... $f^i(0)),$ .... $f(a), f(f(a)),$ ... }

$HB(P_1)$: $\{p(0,0,0), p(a,a,a),$ ... $h(0,0,), h(0,a),$ ... $t(0,0,0), t(a,a,a)$ }

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

$HU(P_1)$: { $0$, $a$, $b$, $r$, $f(0), f(f(0)),$ ... $f^i(0)),$ .... $f(a), f(f(a)),$ ... }

$HB(P_1)$: $\{p(0,0,0), p(a,a,a),$ ... $h(0,0,), h(0,a),$ ... $t(0,0,0), t(a,a,a)$ }

Some Herbrand interpretations:

$I_1 = \emptyset$;

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

$HU(P_1)$: { 0, $a$, $b$, $r$, $f(0), f(f(0)),$ ... $f^i(0))$, .... $f(a), f(f(a)),$ ... }

$HB(P_1)$: $\{p(0,0,0), p(a,a,a),$ ... $h(0,0,), h(0,a),$ ... $t(0,0,0), t(a,a,a)$ }

Some Herbrand interpretations:

$I_1 = \emptyset$; $I_2 = HB(P_1)$;

## Example (Program $P_1$)

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

Constant symbols: $0$, $a$, $b$, $r$; Function symbols: $f$.

$HU(P_1)$: { 0, $a$, $b$, $r$, $f(0), f(f(0)),$ ... $f^i(0))$, .... $f(a), f(f(a)),$ ... }

$HB(P_1)$: $\{p(0,0,0), p(a,a,a),$ ... $h(0,0,), h(0,a),$ ... $t(0,0,0), t(a,a,a)$ }

Some Herbrand interpretations:

$$I_1 = \emptyset; \quad I_2 = HB(P_1); \quad I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}; \ldots$$

# Grounding

The semantics of positive logic programs is defined in terms of grounding.

## Definition (ground instance, grounding)

A *ground instance* of a clause $C$ of the form (1) is any clause $C'$ obtained from $C$ by applying a substitution

$$\theta : Var(C) \rightarrow HU(P)$$

to the variables in $C$, denoted as $Var(C)$.

- $grnd(C)$ denotes the set of all possible ground instances of $C$
- for any program $P$, the *grounding of* $P$ is $grnd(P) = \bigcup_{C \in P} grnd(C)$.

Roughly speaking, $C$ is a shortcut denoting $grnd(C)$, and each variable appearing in $C$ ranges over the Herbrand universe.

## Example (Program $P_2$)

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

## Example (Program $P_2$)

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

The ground instances of the first rule are

$$p(f(0), 0, 0) \leftarrow p(0, 0, 0), h(0, 0), t(0, 0, r). \qquad X = Y = Z = Z' = 0$$
$$\ldots$$
$$p(f(0), r, 0) \leftarrow p(0, r, 0), h(0, r), t(0, 0, r). \qquad X = Z = Z' = 0, Y = r$$
$$\ldots$$
$$p(f(r), r, r) \leftarrow p(r, r, r), h(r, r), t(r, r, r). \qquad X = Y = Z = Z' = r$$
$$\ldots$$
$$p(f(f(0)), 0, 0) \leftarrow p(f(0), 0, 0), h(f(0), 0), t(0, 0, r). \qquad X = Y = Z = Z' = 0$$
$$\ldots$$

## Example (Program $P_2$)

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

The ground instances of the first rule are

$$p(f(0), 0, 0) \leftarrow p(0, 0, 0), h(0, 0), t(0, 0, r). \qquad X = Y = Z = Z' = 0$$
$$\ldots$$
$$p(f(0), r, 0) \leftarrow p(0, r, 0), h(0, r), t(0, 0, r). \qquad X = Z = Z' = 0, Y = r$$
$$\ldots$$
$$p(f(r), r, r) \leftarrow p(r, r, r), h(r, r), t(r, r, r). \qquad X = Y = Z = Z' = r$$
$$\ldots$$
$$p(f(f(0)), 0, 0) \leftarrow p(f(0), 0, 0), h(f(0), 0), t(0, 0, r). \qquad X = Y = Z = Z' = 0$$
$$\ldots$$

The single ground instance of the second rule is

$$h(0, 0).$$

# Herbrand Models

## Definition (Model, satisfaction)

An interpretation $I$ is a (Herbrand) *model* of a

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

## Example (Program $P_2$ cont'd)

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

Which of the following interpretations are models of $P_2$?

- $I_1 = \emptyset$
- $I_2 = HB(P_2)$
- $I_3 = \{h(0, 0), t(a, b, r), p(0, 0, b)\}$

## Example (Program $P_2$ cont'd)

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

Which of the following interpretations are models of $P_2$?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P_2)$
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$

## Example (Program $P_2$ cont'd)

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

Which of the following interpretations are models of $P_2$?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P_2)$ **yes**
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$

## Example (Program $P_2$ cont'd)

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

Which of the following interpretations are models of $P_2$?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P_2)$ **yes**
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$ **no**

## Example (Program $P_2$ cont'd)

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

Which of the following interpretations are models of $P_2$?

- $I_1 = \emptyset$ **no**
- $I_2 = HB(P_2)$ **yes**
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$ **no**

Which of the above interpretations are models of $P_1$?

Note:

## Proposition

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

# 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.

# 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.

## Example

Given

$$P_3 = \{a \leftarrow b. \quad b \leftarrow c. \quad c\},$$

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

# 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.

## Example

Given

$$P_3 = \{a \leftarrow b. \quad b \leftarrow c. \quad c\},$$

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

Given

$$P_4 = \{a \leftarrow b. \quad b \leftarrow a. \quad c\},$$

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

# Minimal Model Semantics (cont'd)

Semantics: Prefer models with true-part as small as possible.

## Definition

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: Prefer models with true-part as small as possible.

## Definition

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

## Theorem

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

# Minimal Model Semantics (cont'd)

Semantics: Prefer models with true-part as small as possible.

## Definition

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

## Theorem

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

This is entailed by the following property:

## Proposition (Intersection closure)

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

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have $LM(P_3) = \{a, b, c\}$.

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have $LM(P_3) = \{a, b, c\}$.
- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have $LM(P_4) = \{c\}$.

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have $LM(P_3) = \{a, b, c\}$.
- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have $LM(P_4) = \{c\}$.
- For $P_2 = \{\ p(f(X), Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \quad h(0, 0)\}$, we have $LM(P_2) = \{h(0, 0)\ \}$.

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have $LM(P_3) = \{a, b, c\}$.
- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have $LM(P_4) = \{c\}$.
- For $P_2 = \{\ p(f(X), Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \quad h(0, 0)\}$, we have $LM(P_2) = \{h(0, 0)\ \}$.
- For $P_1$ above, we have

$$LM(P_1) = \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,a), h(f(0),f(0))\}.$$

# Computation

The minimal model can be computed via fixpoint iteration.

## Definition ($T_P$ Operator)

Let $T_P \colon 2^{HB(P)} \to 2^{HB(P)}$ be defined as

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there exists some } a \leftarrow b_1, \ldots, b_m \\ \text{in } grnd(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \end{array} \right\} .$$

We let denote $\quad T_P^0 = \emptyset, \quad T_P^{i+1} = T_P(T_P^i), \quad i \geq 0.$

# Computation

The minimal model can be computed via fixpoint iteration.

## Definition ($T_P$ Operator)

Let $T_P \colon 2^{HB(P)} \to 2^{HB(P)}$ be defined as

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there exists some } a \leftarrow b_1, \ldots, b_m \\ \text{in } grnd(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \end{array} \right\} .$$

We let denote $\quad T_P^0 = \emptyset, \quad T_P^{i+1} = T_P(T_P^i), \quad i \geq 0.$

Fundamental result:

## Theorem

*$T_P$ has a least fixpoint, $lfp(T_P)$, and the sequence $\langle T_P^i \rangle, \quad i \geq 0$, converges to $lfp(T_P)$.*

Proof: Use the fixpoint theorems of Knaster-Tarski and Kleene.

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T_{P_3}^0 = \{\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T_{P_3}^0 = \{\},\ T_{P_3}^1 = \{c\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

Hence $lfp(T_{P_3}) = \{c, b, a\}$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\}, \quad T^1_{P_4} = \{c\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\}, \quad T^1_{P_4} = \{c\}, \quad T^2_{P_4} = T^1_{P_4}$$

  Hence $lfp(T_{P_4}) = \{c\}$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\}, \quad T^1_{P_4} = \{c\}, \quad T^2_{P_4} = T^1_{P_4}$$

  Hence $lfp(T_{P_4}) = \{c\}$

- For program $P_2$ above, we have

$$T^0_{P_2} = \emptyset,$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\}, \quad T^1_{P_4} = \{c\}, \quad T^2_{P_4} = T^1_{P_4}$$

  Hence $lfp(T_{P_4}) = \{c\}$

- For program $P_2$ above, we have

$$T^0_{P_2} = \emptyset, \quad T^1_{P_2} = \{h(0,0)\},$$

## Example

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \{\},\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}$$

  Hence $lfp(T_{P_3}) = \{c, b, a\}$

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \{\}, \quad T^1_{P_4} = \{c\}, \quad T^2_{P_4} = T^1_{P_4}$$

  Hence $lfp(T_{P_4}) = \{c\}$

- For program $P_2$ above, we have

$$T^0_{P_2} = \emptyset, \quad T^1_{P_2} = \{h(0,0)\}, \quad T^2_{P_2} = T^1_{P_2}.$$

  Hence $lfp(T_{P_2}) = \{h(0,0)\}$.

## Example (cont'd)

- For program $P_1$ above, we have

$$\begin{aligned} T_{P_1}^0 &= \emptyset, \\ T_{P_1}^1 &= \{h(0,0), t(a,b,r), p(0,0,b)\}, \\ T_{P_1}^2 &= \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,b), h(f(0),f(0))\}, \\ T_{P_1}^2 &= T_{P_1}^3. \end{aligned}$$

Hence
$lfp(T_{P_1}) = \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,b), h(f(0),f(0))\}.$

## Example (cont'd)

- For program $P_1$ above, we have

$$\begin{aligned} T_{P_1}^0 &= \emptyset, \\ T_{P_1}^1 &= \{h(0,0), t(a,b,r), p(0,0,b)\}, \\ T_{P_1}^2 &= \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,b), h(f(0),f(0))\}, \\ T_{P_1}^2 &= T_{P_1}^3. \end{aligned}$$

  Hence
  $lfp(T_{P_1}) = \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,b), h(f(0),f(0))\}.$

- For program $P = \{p(0). \quad p(f(X)) \leftarrow p(X)\}$, we have

$$T_P^0 = \emptyset, \quad T_P^1 = \{p(0)\}, \ldots, T_P^i = \{p(0), \ldots p(f^{i-1}(0))\}, \quad i \geq 0;$$

  hence $lfp(T_P) = \{p(f^i(0)) \mid i \geq 0\}$ is infinite.

# Negation in Logic Programs

Why negation?

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

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

# Negation in Logic Programs

Why negation?

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

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

## Definition

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 in a first-order language $L$.

$not$ is called “negation as failure”, “default negation”, or “weak negation”

Things get more complex!

# Programs with Negation

Prolog: “$not\ \langle X \rangle$” means “Negation as Failure (to prove to $\langle X \rangle$)”

**Different from negation in classical logic!**

# Programs with Negation

Prolog: "$not\ \langle X \rangle$" means "Negation as Failure (to prove to $\langle X \rangle$)"

**Different from negation in classical logic!**

## Example (Program $P_5$)

$$man(dilbert).$$

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

$$husband(X) \leftarrow fail. \quad \text{\% fail = "false" in Prolog}$$

# Programs with Negation

Prolog: "$not\ \langle X \rangle$" means "Negation as Failure (to prove to $\langle X \rangle$)"

**Different from negation in classical logic!**

## Example (Program $P_5$)

$$man(dilbert).$$

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

$$husband(X) \leftarrow fail. \quad \text{\% fail = "false" in Prolog}$$

Query:

$$?- single(X).$$

# Programs with Negation

Prolog: "$not\ \langle X \rangle$" means "Negation as Failure (to prove to $\langle X \rangle$)"

**Different from negation in classical logic!**

## Example (Program $P_5$)

$$man(dilbert).$$

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

$$husband(X) \leftarrow fail. \quad \text{\% fail = "false" in Prolog}$$

Query:

$$?- single(X).$$

Answer:

$$X = dilbert \ .$$

## Example (cont'd)

Modifying the last rule of $P_5$, we get $P_6$:

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

Result in Prolog ????

## Example (cont'd)

Modifying the last rule of $P_5$, we get $P_6$:

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

Result in Prolog ????

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

## Example (cont'd)

Modifying the last rule of $P_5$, we get $P_6$:

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

Result in Prolog ????

**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 Logic Programs With Negation

- “War of Semantics” in Logic Programming (1980/90ies):

  Meaning of programs like the Dilbert example above

# Semantics of Logic Programs With Negation

- “War of Semantics” in Logic Programming (1980/90ies):

  Meaning of programs like the Dilbert example above
- Great Schism: Single model vs. multiple model semantics

# Semantics of Logic Programs With Negation

- "War of Semantics" in Logic Programming (1980/90ies):

  Meaning of programs like the Dilbert example above

- Great Schism: Single model vs. multiple model semantics

- To date:
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]

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

# Semantics of Logic Programs With Negation

- "War of Semantics" in Logic Programming (1980/90ies):

  Meaning of programs like the Dilbert example above

- Great Schism: Single model vs. multiple model semantics

- To date:
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]

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

  - *Answer Set (alias Stable Model) Semantics* by Gelfond and Lifschitz [1988,1991].

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

# Semantics of Logic Programs With Negation

- "War of Semantics" in Logic Programming (1980/90ies):

  Meaning of programs like the Dilbert example above

- Great Schism: Single model vs. multiple model semantics

- To date:
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]

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

  - *Answer Set (alias Stable Model) Semantics* by Gelfond and Lifschitz [1988,1991].

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

- Agreement for so-called "stratified programs"

  Different selection principles for non-stratified programs

# Stratified Negation

**Intuition**: To evaluate a rule $r$: $a \leftarrow \cdots, not\, p(\vec{t}), \cdots$, the value of $p(\vec{t})$ should be known.

1. Evaluate first $p(\vec{t})$.
2. if $p(\vec{t})$ is $\begin{cases} \text{false}, & \text{then } not\, p(\vec{t}) \text{ is true,} \\ \text{true}, & \text{then } not\, p(\vec{t}) \text{ is false and } r \text{ is not applicable.} \end{cases}$

# Stratified Negation

**Intuition**: To evaluate a rule $r$: $a \leftarrow \cdots, not\, p(\vec{t}), \cdots$, the value of $p(\vec{t})$ should be known.

1. Evaluate first $p(\vec{t})$.
2. if $p(\vec{t})$ is $\begin{cases} \text{false}, & \text{then } not\, p(\vec{t}) \text{ is true,} \\ \text{true}, & \text{then } not\, p(\vec{t}) \text{ is false and } r \text{ is not applicable.} \end{cases}$

## Example

$$P = \{\, boring(chess) \leftarrow not\, interesting(chess) \,\}$$

- $interesting(chess)$ is false $\Rightarrow$ $not\, interesting(chess)$ is true.
- hence, $r$ is applied and $boring(chess)$ is true.
- This leads to the Herbrand model $H = \{boring(chess)\}$ of $P$.

# Stratified Negation

**Intuition**: To evaluate a rule $r$: $a \leftarrow \cdots, not\, p(\vec{t}), \cdots$, the value of $p(\vec{t})$ should be known.

1. Evaluate first $p(\vec{t})$.
2. if $p(\vec{t})$ is $\begin{cases} \text{false}, & \text{then } not\, p(\vec{t}) \text{ is true,} \\ \text{true}, & \text{then } not\, p(\vec{t}) \text{ is false and } r \text{ is not applicable.} \end{cases}$

## Example

$$P = \{\, boring(chess) \leftarrow not\, interesting(chess) \,\}$$

- $interesting(chess)$ is false $\Rightarrow$ $not\, interesting(chess)$ is true.
- hence, $r$ is applied and $boring(chess)$ is true.
- This leads to the Herbrand model $H = \{boring(chess)\}$ of $P$.

**Note**: this introduces *procedurality* (violates declarativity)!

# Dependency Graph

**Restriction:**

- The method works if there is no cyclic negation.
- Need a syntactic criterion to ensure this property.

## Definition (Dependency graph)

The *dependency graph* of a set $P$ of rules, is a directed graph $dep(P) = \langle N, E \rangle$ where

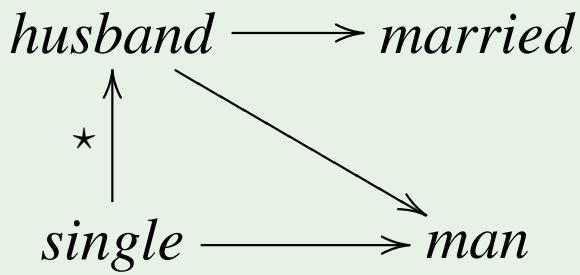
- $N$ = { predicate $p$ | $p$ occurs in $P$, $p$ is not a built-in } (=: $pred(S)$)
- $E$ contains $p \to q$, iff $P$ contains some rule $a \leftarrow \cdots, \ell, \cdots$ with $a = p(\cdots)$ and $\ell = q(\cdots)$ or $\ell = not\, q(\cdots)$

  Label $p \to q$ with "*", if $\ell = not\, q(\cdots)$

## Example (Program $P_7$)

$$man(dilbert).$$
$$husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X), not\ husband(X).$$

$dep(P_7)$:



# Stratification

## Definition (Stratification)

A *stratification* of a set $P$ of rules is a partitioning

$$\Sigma = \{S_1 \ldots, S_n\}$$

of $pred(P)$ into $n$ nonempty, pairwise disjoint sets such that

(a) if $p \in S_i$, $q \in S_j$, and $p \rightarrow q$ is in $dep(P)$, then $i \geq j$; and

(b) if $p \in S_i$, $q \in S_j$, and $p \rightarrow^\star q$ is in $dep(P)$ then $i > j$.

The sets $S_1, \ldots, S_n$ are the *strata* of $P$ w.r.t. $\Sigma$.

$P$ is *stratified*, if it has some stratification $\Sigma$.

- Informally, $\Sigma$ specifies an *order of evaluation* for the predicates in $P$
- The sequential evaluation of $S_1$, $S_2$,. . . ,$S_n$ can be done by computing a series of *iterative least models*.

# Semantics

## Definition (iterative least model)

Suppose $P$ is a logic program with stratification $\Sigma = \{S_1, \ldots, S_k\}$, $k \geq 1$. Then

- $P_{S_i} = \{a \leftarrow b_1, \ldots, b_n \in P \mid a = p(\cdots), p \in S_i\}$, and
- $HB^\star(P_{S_i}) = \bigcup_{j \leq i} \{p(\mathbf{t}) \in HB(P) \mid p \in S_j\}$.

The *iterative least models* $M_i \subseteq HB(P)$, $1 \leq i \leq k$, are such that

(i) $M_1$ is the least model of $P_{S_1}$;

(ii) if $i > 1$, then $M_i$ is the least subset $M$ of $HB(P)$ such that
- $M$ is a model of $P_{S_i}$, and
- $M \cap HB^\star(P_{S_{i-1}}) = M_{i-1} \cap HB^\star(P_{S_{i-1}})$.

The *iterative least model* of $P$ is $M_{P,\Sigma} = M_k$.

## Example ($P_7$ cont'd)

$$man(dilbert).$$
$$husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X),\ not\ husband(X).$$

Stratification: $\Sigma = \{S_1 = \{man, married\},\ S_2 = \{husband\},\ S_3 = \{single\}\}$

## Example ($P_7$ cont'd)

$$man(dilbert).$$
$$husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X),\ not\ husband(X).$$

Stratification: $\Sigma = \{S_1 = \{man, married\},\ S_2 = \{husband\},\ S_3 = \{single\}\}$

- $P_{S_1} = \{man(dilbert)\}$ and $M_1 = LM(P_{S_1}) = \{man(dilbert)\}$.

## Example ($P_7$ cont'd)

$$man(dilbert).$$
$$husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X),\ not\ husband(X).$$

Stratification: $\Sigma = \{S_1 = \{man, married\},\ S_2 = \{husband\},\ S_3 = \{single\}\}$

- $P_{S_1} = \{man(dilbert)\}$ and $M_1 = LM(P_{S_1}) = \{man(dilbert)\}$.
- $P_{S_2} = \{husband(X) \leftarrow man(X), married(X)\}$.

  $HB^\star(P_{S_1}) = \{man(dilbert), married(dilbert)\}$.

  Then, $M_2 = \{man(dilbert)\}$ is a model of $P_{S_2}$ and
  $M_2 \cap HB^\star(P_{S_1}) = M_1 \cap HB^\star(P_{S_1})$ (no smaller such model exists)

## Example ($P_7$ cont'd)

$$man(dilbert).$$
$$husband(X) \leftarrow man(X),\ married(X).$$
$$single(X) \leftarrow man(X),\ not\ husband(X).$$

Stratification: $\Sigma = \{S_1 = \{man, married\},\ S_2 = \{husband\},\ S_3 = \{single\}\}$

- $P_{S_1} = \{man(dilbert)\}$ and $M_1 = LM(P_{S_1}) = \{man(dilbert)\}$.
- $P_{S_2} = \{husband(X) \leftarrow man(X), married(X)\}$.
  $HB^\star(P_{S_1}) = \{man(dilbert), married(dilbert)\}$.
  Then, $M_2 = \{man(dilbert)\}$ is a model of $P_{S_2}$ and
  $M_2 \cap HB^\star(P_{S_1}) = M_1 \cap HB^\star(P_{S_1})$ (no smaller such model exists)
- $P_{S_3} = \{single(X) \leftarrow man(X),\ not\ husband(X)\}$.
  Thus $M_3 = \{single(dilbert)\} \cup M_2$ is the least model of $P_{S_3}$ such that
  $M_3 \cap HB^\star(P_{S_2}) = M_2 \cap HB^\star(P_{S_2})$.

# Stratification Theorem

Note: stratifications are not unique.

## Example ($P_7$ cont'd)

Other stratification: $\Sigma' = \{S'_1 = \{man, married, husband\},\ S'_2 = \{single\}\}$.

Evaluation with $\Sigma'$ yields same result!

# Stratification Theorem

Note: stratifications are not unique.

## Example ($P_7$ cont'd)

Other stratification: $\Sigma' = \{S'_1 = \{man, married, husband\},\ S'_2 = \{single\}\}$.

Evaluation with $\Sigma'$ yields same result!

This is not accidental:

## Theorem (Apt *et al.* [1988])

*Let $P$ be a stratified program. Then for every stratifications $\Sigma$ and $\Sigma'$ of $P$, it holds that $M_{P,\Sigma} = M_{P,\Sigma'}$.*

Hence, we can simplify $M_{P,\Sigma}$ to $M_P = M_{P,\Sigma}$ (for arbitrary $\Sigma$ of choice)

# Stratification Theorem

Note: stratifications are not unique.

## Example ($P_7$ cont'd)

Other stratification: $\Sigma' = \{S'_1 = \{man, married, husband\},\ S'_2 = \{single\}\}$.

Evaluation with $\Sigma'$ yields same result!

This is not accidental:

## Theorem (Apt *et al.* [1988])

*Let $P$ be a stratified program. Then for every stratifications $\Sigma$ and $\Sigma'$ of $P$, it holds that $M_{P,\Sigma} = M_{P,\Sigma'}$.*

Hence, we can simplify $M_{P,\Sigma}$ to $M_P = M_{P,\Sigma}$ (for arbitrary $\Sigma$ of choice)

## Corollary

*Stratified programs have a canonical model, also called perfect model.*

# Stable model semantics

First, for variable-free (ground) programs $P$

- Treat "$not$" specially
- Intuitively, literals $not\, a$ are a source of "contradiction" or "unstability".

# Stable model semantics

First, for variable-free (ground) programs $P$

- Treat "*not*" specially
- Intuitively, literals $not\, a$ are a source of "contradiction" or "unstability".

## Example ($P_6$ cont'd)

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

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

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

# Stable model semantics

First, for variable-free (ground) programs $P$

- Treat "*not*" specially
- Intuitively, literals $not\, a$ are a source of "contradiction" or "unstability".

## Example ($P_6$ cont'd)

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

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

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

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

  If as in $M'$, $man(dilbert)$ were true and $husband(dilbert)$ false, by $r_1$ also $single(dilbert)$ should be true. This is not coherent.

# Stable model semantics

First, for variable-free (ground) programs $P$

- Treat "*not*" specially
- Intuitively, literals $not\, a$ are a source of "contradiction" or "unstability".

## Example ($P_6$ cont'd)

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

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

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

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

  If as in $M'$, $man(dilbert)$ were true and $husband(dilbert)$ false, by $r_1$ also $single(dilbert)$ should be true. This is not coherent.

- Consider $M'' = \{man(dilbert), single(dilbert), husband(dilbert)\}$.

  The bodies of $r_2$ and $R_2$ are not true wrt $M''$, hence there is no evidence for $single(dilbert)$ and $husband(dilbert)$ being true.

# Stable Models

## Definition (Gelfond-Lifschitz Reduct $P^M$ 1988)

The *GL-reduct* (simply *reduct*) of a ground program $P$ w.r.t. an interpretation $M$, denoted $P^M$, is the program obtained from $P$ by

1. removing rules with $not\, a$ in the body for each $a \in M$; and
2. removing literals $not\, a$ from all other rules.

# Stable Models

## Definition (Gelfond-Lifschitz Reduct $P^M$ 1988)

The *GL-reduct* (simply *reduct*) of a ground program $P$ w.r.t. an interpretation $M$, denoted $P^M$, is the program obtained from $P$ by

1. removing rules with $not\, a$ in the body for each $a \in M$; and
2. removing literals $not\, a$ from all other rules.

**Intuition:**

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

# Stable Models (cont'd)

## Definition (stable model)

An interpretation $M$ of $P$ is a *stable* model of $P$, if

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

# Stable Models (cont'd)

## Definition (stable model)

An interpretation $M$ of $P$ is a *stable* model of $P$, if

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

Observe:

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

## Example ($P_6$ cont'd)

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

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

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

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 ($P_6$ cont'd)

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

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

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

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 ($P_6$ cont'd)

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

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

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

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

## Example ($P_6$ cont'd)

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

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

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

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

  reduct $P_6^{M_1}$:

  $$man(dilbert).$$

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

  The least model of $P_6^{M_1}$ is $\{man(dilbert), single(dilbert)\} = M_1$.

## Example ($P_6$ cont'd)

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

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

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

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

  reduct $P_6^{M_1}$:

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

  The least model of $P_6^{M_1}$ is $\{man(dilbert), single(dilbert)\} = M_1$.

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

## Example ($P_6$ cont'd)

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

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

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

- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$:

## Example ($P_6$ cont'd)

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

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

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

- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$:

  $P_6^{M_3}$ is

  $$man(dilbert).$$

  $LM(P_6^{M_3}) = \{man(dilbert)\} \neq M_3$.

## Example ($P_6$ cont'd)

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

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

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

- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$:

  $P_6^{M_3}$ is

  $$man(dilbert).$$

  $LM(P_6^{M_3}) = \{man(dilbert)\} \neq M_3$.

- $M_4 = \{man(dilbert)\}$:

## Example ($P_6$ cont'd)

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

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

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

- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$:

  $P_6^{M_3}$ is

  $$man(dilbert).$$

  $LM(P_6^{M_3}) = \{man(dilbert)\} \neq M_3$.

- $M_4 = \{man(dilbert)\}$:

  $P_6^{M_4}$ is

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

  $LM(P_6^{M_4}) = \{man(dilbert), single(dilbert), husband(dilbert)\} \neq M_4$.

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

## Example ($P_\perp$)

$$p \leftarrow not\, p$$

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

## Example ($P_\perp$)

$$p \leftarrow not\, p$$

- Candidate interpretations: $M_1 = \{\}, M_2 = \{p\}$.

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

## Example ($P_\perp$)

$$p \leftarrow not\, p$$

- Candidate interpretations: $M_1 = \{\}, M_2 = \{p\}$.
- $M_1$: $P_\perp^{M_1} = \{p\}$, and $LM(P_\perp) = \{p\} \neq M_1$.

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

## Example ($P_\perp$)

$$p \leftarrow not\, p$$

- Candidate interpretations: $M_1 = \{\}, M_2 = \{p\}$.
- $M_1$: $P_\perp^{M_1} = \{p\}$, and $LM(P_\perp) = \{p\} \neq M_1$.
- $M_2$: $P_\perp^{M_2} = \{\}$, and $LM(P_\perp) = \{\} \neq M_2$.

# Inconsistent Programs

- Each normal logic program has some Herbrand model.
- However, it may have no stable model.

## Example ($P_\perp$)

$$p \leftarrow not\, p$$

- Candidate interpretations: $M_1 = \{\}, M_2 = \{p\}$.
- $M_1$: $P_\perp^{M_1} = \{p\}$, and $LM(P_\perp) = \{p\} \neq M_1$.
- $M_2$: $P_\perp^{M_2} = \{\}$, and $LM(P_\perp) = \{\} \neq M_2$.

Note:

- If $p$ does not occur in $P$, then $P \cup \{p \leftarrow not\, p\}$ has no stable model.
- Adding $p \leftarrow not\, p$ to $P$ "kills" all stable models of $P$ !

# Programs with Variables

- Consider, like in Prolog, only Herbrand interpretations.
- As for positive programs, view a program clause as a shorthand for all its ground instances.
- Recall: $grnd(P)$ is the grounding of program $P$.

## Definition (stable model, general case)

An interpretation $M$ of $P$ is a *stable model* of $P$, if $M$ is a stable model of $grnd(P)$.

# Programs with Variables

- Consider, like in Prolog, only Herbrand interpretations.
- As for positive programs, view a program clause as a shorthand for all its ground instances.
- Recall: $grnd(P)$ is the grounding of program $P$.

## Definition (stable model, general case)

An interpretation $M$ of $P$ is a *stable model* of $P$, if $M$ is a stable model of $grnd(P)$.

- Alternative way: Perform grounding in the GL-reduct, i.e., require $M = LM(P^M)$ where $P^M =_{def} grnd(P)^M$ for non-ground $P$.

## Example (Variant $P_6'$ of $P_6$)

$$man(dilbert). \quad (r_1)$$

$$woman(alice). \quad (r_2)$$

$$single(X) \leftarrow man(X), not\ husband(X). \quad (r_3)$$

$$husband(X) \leftarrow man(X), not\ single(X). \quad (r_4)$$

## Example (Variant $P_6'$ of $P_6$)

$$man(dilbert). \qquad (r_1)$$
$$woman(alice). \qquad (r_2)$$
$$single(X) \leftarrow man(X), not\ husband(X). \qquad (r_3)$$
$$husband(X) \leftarrow man(X), not\ single(X). \qquad (r_4)$$

We have that, for instance,

$$grnd(r_3) = \{\ single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert).$$
$$single(alice) \leftarrow man(alice), not\ husband(alice). \quad \};$$

## Example (Variant $P'_6$ of $P_6$)

$$man(dilbert). \qquad (r_1)$$
$$woman(alice). \qquad (r_2)$$
$$single(X) \leftarrow man(X), not\ husband(X). \qquad (r_3)$$
$$husband(X) \leftarrow man(X), not\ single(X). \qquad (r_4)$$

We have that, for instance,

$$grnd(r_3) = \{\ single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert).$$
$$single(alice) \leftarrow man(alice), not\ husband(alice).\ \};$$

$$grnd(P'_6) = \{\ man(dilbert).$$
$$woman(alice).$$
$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert).$$
$$single(alice) \leftarrow man(alice), not\ husband(alice).$$
$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert).$$
$$husband(alice) \leftarrow man(alice), not\ single(alice).\ \}.$$

## Example ($P'_6$ cont'd)

The program $grnd(P'_6)$, and thus $P'_6$, has the following stable models:

- $M_1 = \{man(dilbert), woman(alice), single(dilbert)\}$
- $M_2 = \{man(dilbert), woman(alice), husband(dilbert)\}$

Indeed,

- the rule instances of $r_3$ and $r_4$ for $dilbert$ generate two possible scenarios;
- the rule instances of $r_3$ and $r_4$ for $alice$ are inapplicable.

# Properties of Stable Models

- Stable model semantics has a strong theoretical basis, many properties are known.
- We consider here some elementary ones.
- See e.g.
  - [Lifschitz, 2008]
  - [Ferraris and Lifschitz, 2005]
  - [Gelfond, 2008]

  for other insights, alternative definitions and properties of stable models.

# Relationship to Classical Models

How do stable models of $P$ relate to classical models of $P$?

## Definition (classical model of normal logic program)

An interpretation $I$ is a *model* of

- a ground clause $C : a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$, if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $\{a, c_1, \ldots, c_n\} \cap I \neq \emptyset$ $(I \models C)$;
- a clause $C$, if $I \models C'$ for every $C' \in grnd(C)$ $(I \models C)$;
- a set $P$ of rules, if $I \models C$ for every clause $C$ in $P$ $(I \models P)$.

This complies with Herbrand models satisfying the clause

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

where $not$ is interpreted as classical negation ("¬").

# Relationship to Classical Models (cont'd)

The following holds:

## Theorem

1. *Every stable model $M$ of $P$ is a model of $P$.*
2. *A stable model $M$ does not contain any model $M'$ of $P$ properly ($M' \not\subset M$), i.e., is a minimal model of $P$ (w.r.t. $\subseteq$).*

# Relationship to Classical Models (cont'd)

The following holds:

## Theorem

1. *Every stable model $M$ of $P$ is a model of $P$.*
2. *A stable model $M$ does not contain any model $M'$ of $P$ properly ($M' \not\subset M$), i.e., is a minimal model of $P$ (w.r.t. $\subseteq$).*

## Corollary

*Stable models are incomparable w.r.t. $\subseteq$, i.e., if $M_1$ and $M_2$ are different stable models of $P$, then $M_1 \not\subseteq M_2$ and $M_2 \not\subseteq M_1$.*

Thus, stable models adhere to minimality of positive information.

# Supportedness

- Note: each atom $a$ in a stable model $M$ must be derived from some rule of $P$.
- Extend the immediate consequence operator $T_P$ to *not* .

## Definition ($T_P$ for normal $P$)

Given a normal program $P$ and an interpretation $I$, let

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there is some } r = a \leftarrow b_1, \ldots, b_m, \mathit{not}\, c_1, \ldots, \mathit{not}\, c_n \in grnd(P) \\ \text{such that } \{b_1, \ldots, b_m\} \subseteq I, \{c_1, \ldots c_m\} \cap I = \emptyset \end{array} \right\} .$$

# Supportedness

- Note: each atom $a$ in a stable model $M$ must be derived from some rule of $P$.
- Extend the immediate consequence operator $T_P$ to *not* .

## Definition ($T_P$ for normal $P$)

Given a normal program $P$ and an interpretation $I$, let

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there is some } r = a \leftarrow b_1, \ldots, b_m, \mathit{not}\, c_1, \ldots, \mathit{not}\, c_n \in \mathit{grnd}(P) \\ \text{such that } \{b_1, \ldots, b_m\} \subseteq I, \{c_1, \ldots c_m\} \cap I = \emptyset \end{array} \right\} .$$

An interpretation $I$ of $P$ is a *supported model* of $P$, if $T_P(I) = I$.

# Supportedness

- Note: each atom $a$ in a stable model $M$ must be derived from some rule of $P$.
- Extend the immediate consequence operator $T_P$ to *not* .

## Definition ($T_P$ for normal $P$)

Given a normal program $P$ and an interpretation $I$, let

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there is some } r = a \leftarrow b_1, \ldots, b_m, \mathit{not}\, c_1, \ldots, \mathit{not}\, c_n \in grnd(P) \\ \text{such that } \{b_1, \ldots, b_m\} \subseteq I, \{c_1, \ldots c_m\} \cap I = \emptyset \end{array} \right\} .$$

An interpretation $I$ of $P$ is a *supported model* of $P$, if $T_P(I) = I$.

## Theorem

*Every stable model $M$ of $P$ is a supported model of $P$.*

# Supportedness (cont'd)

- In fact, by minimality of stable models, every stable model is a minimal (w.r.t. $\subseteq$) supported model of $P$.
- The converse is not true.

# Supportedness (cont'd)

- In fact, by minimality of stable models, every stable model is a minimal (w.r.t. $\subseteq$) supported model of $P$.
- The converse is not true.

## Example (Program $P_s$)

$$a \leftarrow not\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

# Supportedness (cont'd)

- In fact, by minimality of stable models, every stable model is a minimal (w.r.t. $\subseteq$) supported model of $P$.
- The converse is not true.

## Example (Program $P_s$)

$$a \leftarrow not\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

- Note that $M_1 = \{a\}$ and $M_2 = \{b, c\}$ are both minimal such that $T_{P_s}(M_1) = M_1$ and $T_{P_s}(M_2) = M_2$.

# Supportedness (cont'd)

- In fact, by minimality of stable models, every stable model is a minimal (w.r.t. $\subseteq$) supported model of $P$.
- The converse is not true.

## Example (Program $P_s$)

$$a \leftarrow not\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

- Note that $M_1 = \{a\}$ and $M_2 = \{b, c\}$ are both minimal such that $T_{P_s}(M_1) = M_1$ and $T_{P_s}(M_2) = M_2$.
- The single stable model of $P_s$ is $M_1$.

# Supportedness (cont'd)

- In fact, by minimality of stable models, every stable model is a minimal (w.r.t. $\subseteq$) supported model of $P$.
- The converse is not true.

## Example (Program $P_s$)

$$a \leftarrow not\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

- Note that $M_1 = \{a\}$ and $M_2 = \{b, c\}$ are both minimal such that $T_{P_s}(M_1) = M_1$ and $T_{P_s}(M_2) = M_2$.
- The single stable model of $P_s$ is $M_1$.
- Problem with $M_2$: Self-supportedness of $b$ (via $c$)

# Unfounded sets

Stable models amount to supported models with no (cyclic) self-support

## Definition (cf. [Van Gelder *et al.*, 1991],[Leone *et al.*, 1997])

A set $U \subseteq HB_P$ is an *unfounded set* of $P$ relative to interpretation $I$, if for every $a \in U$ and $r : a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$ in $\in grnd(P)$, either

1. for some $i \in \{1, \ldots, m\}$, either $b_i \notin I$ or $b_i \in U$, or
2. for some $j \in \{1, \ldots, n\}$, $c_j \in I$.

# Unfounded sets

Stable models amount to supported models with no (cyclic) self-support

## Definition (cf. [Van Gelder *et al.*, 1991],[Leone *et al.*, 1997])

A set $U \subseteq HB_P$ is an *unfounded set* of $P$ relative to interpretation $I$, if for every $a \in U$ and $r : a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$ in $\in grnd(P)$, either

1. for some $i \in \{1, \ldots, m\}$, either $b_i \notin I$ or $b_i \in U$, or
2. for some $j \in \{1, \ldots, n\}$, $c_j \in I$.

- Every $P$ has a *greatest unfounded set* relative to $I$, denoted $U_P(I)$.
- Intuitively, if $I$ is compatible with $P$, all atoms in $U_P(I)$ can be safely switched to false while maintaining compatibilty.

# Unfounded sets

Stable models amount to supported models with no (cyclic) self-support

## Definition (cf. [Van Gelder *et al.*, 1991],[Leone *et al.*, 1997])

A set $U \subseteq HB_P$ is an *unfounded set* of $P$ relative to interpretation $I$, if for every $a \in U$ and $r : a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$ in $\in grnd(P)$, either

1. for some $i \in \{1, \ldots, m\}$, either $b_i \notin I$ or $b_i \in U$, or
2. for some $j \in \{1, \ldots, n\}$, $c_j \in I$.

- Every $P$ has a *greatest unfounded set* relative to $I$, denoted $U_P(I)$.
- Intuitively, if $I$ is compatible with $P$, all atoms in $U_P(I)$ can be safely switched to false while maintaining compatibilty.

## Definition (unfounded-freeness)

$I$ is called *unfounded-free*, if $I \cap U = \emptyset$ for each unfounded set $U$ of $P$ rel. to $I$.

Note: $I$ is *unfounded-free* iff $I \cap U_P(I) = \{\}$.

# Unfounded sets (cont'd)

## Theorem (implicit in [Leone *et al.*, 1997])

*Given a program $P$, a model $M$ of $P$ is stable iff $M$ is* unfounded-free*.*

# Unfounded sets (cont'd)

## Theorem (implicit in [Leone *et al.*, 1997])

*Given a program $P$, a model $M$ of $P$ is stable iff $M$ is* unfounded-free*.*

## Example ($P_s$ cont'd)

$$a \leftarrow not\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

- $M_2 = \{b, c\}$: $U_{P_s}(M_2) = \{b, c\}$, thus $M_2 \cap U_{P_s}(M_2) \neq \emptyset$.
- $M_1 = \{a\}$: $U_{P_s}(M_1) = \emptyset$, thus $M_1 \cap U_{P_s}(M_1) = \emptyset$.

- Unfounded-freeness is exploited for computing stable models (DLV)
- It corresponds to loop formulas [Lin and Zhao, 2002], [Lee, 2005].

# Stratified Programs

Stable model semantics gracefully generalizes stratified semantics:

## Theorem

*If a program $P$ is stratified, then $P$ has a single stable model, which coincides with the perfect (i.e., the iterative least) model model of $P$.*

# Stratified Programs

Stable model semantics gracefully generalizes stratified semantics:

## Theorem

*If a program $P$ is stratified, then $P$ has a single stable model, which coincides with the perfect (i.e., the iterative least) model model of $P$.*

Notes:

- A stratified $P$ may have several minimal models; only one is stable

  E.g., $P = \{boring(chess) \leftarrow not\ interesting(chess)\}$

  has two minimal models:

  $$M_1 = \{boring(chess)\} \text{ and } M_2 = \{interesting(chess)\}.$$

  The perfect model is $M_P = M_1$.

# Stratified Programs

Stable model semantics gracefully generalizes stratified semantics:

## Theorem

*If a program $P$ is stratified, then $P$ has a single stable model, which coincides with the perfect (i.e., the iterative least) model model of $P$.*

Notes:

- A stratified $P$ may have several minimal models; only one is stable

  E.g., $P = \{boring(chess) \leftarrow not\ interesting(chess)\}$

  has two minimal models:

  $$M_1 = \{boring(chess)\} \text{ and } M_2 = \{interesting(chess)\}.$$

  The perfect model is $M_P = M_1$.

- Stratified programs can only express deterministic scenarios, no "alternatives" are possible!

# Non-Cumulativity

- In classical logic, adding consequences of a theory $T$ to $T$ preserves its semantics.
- This property is known as cumulativity (or lemma support).
- For stable model semantics, this property does not hold.

# Non-Cumulativity

- In classical logic, adding consequences of a theory $T$ to $T$ preserves its semantics.
- This property is known as cumulativity (or lemma support).
- For stable model semantics, this property does not hold.

## Proposition

*Suppose $P$ and atom $a$ fulfill $M \models a$, for each stable model $M$ of $P$. Then $P$ and $P \cup \{ a \}$ need not have the same stable models (even if $P$ is consistent).*

# Non-Cumulativity

- In classical logic, adding consequences of a theory $T$ to $T$ preserves its semantics.
- This property is known as cumulativity (or lemma support).
- For stable model semantics, this property does not hold.

## Proposition

*Suppose $P$ and atom $a$ fulfill $M \models a$, for each stable model $M$ of $P$. Then $P$ and $P \cup \{ a \}$ need not have the same stable models (even if $P$ is consistent).*

## Example

$$b \leftarrow not\, c. \quad c \leftarrow not\, b.$$
$$a \leftarrow b. \quad a \leftarrow not\, a.$$

$P$ has the stable model $M = \{a, b\}$; $P \cup \{a\}$ has in addition $N = \{a, c\}$.

Note: the property holds for stratified programs.

# Computational Properties

How difficult is it to compute some stable model?

## Decision problem CONS:

Given a program $P$, does $P$ have some stable model?

# Computational Properties

How difficult is it to compute some stable model?

## Decision problem CONS:

Given a program $P$, does $P$ have some stable model?

## Theorem

*For normal logic programs $P$, problem* CONS *is*

- NP*-complete in the propositional and ground case;*
- NEXPTIME*-complete in the datalog (function-free) case;*
- $\Sigma_1^1$*-complete in the general first-order case.*

Recall: NP (NEXPTIME) = class of problems solvable in polynomial (exponential) time on a non-deterministic Turing machine.

$\Sigma_1^1$ is a class in the Analytic Hierarchy

# Computational Properties (cont'd)

Lower complexity holds for fragments:

- For positive and stratified propositional programs, CONS is polynomial (in fact, trivial).
  - Still solvable in linear time if constraints and strong negation are allowed (P-complete).
  - For datalog programs, complexity increases to EXPTIME.
- For programs with function symbols, several decidable program classes are known (up to 3-EXPTIME).

More on basic complexity: [Dantsin *et al.*, 2001].

# Extensions

- Many extensions exist, partly motivated by applications
- Some are syntactic sugar, other strictly add expressiveness
- Incomplete list:
  - constraints
  - strong negation
  - disjunction
  - nested expressions
  - cardinality constraints (Smodels)
  - optimization: weight constraints, *minimize* (Smodels); weak constraints (DLV)
  - aggregates (Smodels, DLV)
  - templates (for macros), external functions (DLVHEX)
  - Frame Logic syntax (for Semantic Web)
  - preferences: e.g., PLP
  - KR frontends (diagnosis, inheritance, planning,...) in DLV
- Comprehensive survey: [Niemelä (ed.), 2005]

# Extensions

- Many extensions exist, partly motivated by applications
- Some are syntactic sugar, other strictly add expressiveness
- Incomplete list:
  - constraints
  - strong negation
  - disjunction
  - nested expressions
  - cardinality constraints (Smodels)
  - optimization: weight constraints, *minimize* (Smodels); weak constraints (DLV)
  - aggregates (Smodels, DLV)
  - templates (for macros), external functions (DLVHEX)
  - Frame Logic syntax (for Semantic Web)
  - preferences: e.g., PLP
  - KR frontends (diagnosis, inheritance, planning,...) in DLV
- Comprehensive survey: [Niemelä (ed.), 2005]

# Constraints

- Adding

$$p \leftarrow q_1, \ldots, q_m, not\ r_1, \ldots, not\ r_n, not\ p.$$

to $P$ "kills" all stable models of $P$ that

  - contain $q_1, \ldots, q_m$, and
  - do not contain $r_1, \ldots, r_n$

- This is convenient to eliminate scenarios which does not satisfy integrity constraints.

# Constraints

- Adding

$$p \leftarrow q_1, \ldots, q_m, not\ r_1, \ldots, not\ r_n, not\ p.$$

to $P$ "kills" all stable models of $P$ that

  - contain $q_1, \ldots, q_m$, and
  - do not contain $r_1, \ldots, r_n$

- This is convenient to eliminate scenarios which does not satisfy integrity constraints.
- Short:

**Constraint**

$$\leftarrow q_1, \ldots, q_m, not\ r_1, \ldots, not\ r_n.$$

## Example (Dilbert $P_6$ cont'd)

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

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

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

$$\leftarrow husband(X), not\ wedding\_ring(X). \qquad (c_1)$$

- The constraint $c_1$ eliminates models in which there is no evidence for a husband having a wedding ring.
- Single stable model: $M_1 = \{man(dilbert), single(dilbert)\}$

# Strong Negation

- Weak negation "$not\, a$" means "$a$ can not be proved (derived) using rules," and that $a$ is false by default (believed to be false).
- This is different from *knowing* (provably) that $a$ is false; this is expressed by $-a$ (sometimes $\neg a$).
- This is called *strong negation* and may make an important difference.

# Strong Negation

- Weak negation "$not\, a$" means "$a$ can not be proved (derived) using rules," and that $a$ is false by default (believed to be false).
- This is different from *knowing* (provably) that $a$ is false; this is expressed by $-a$ (sometimes $\neg a$).
- This is called *strong negation* and may make an important difference.

## Example (due to John McCarthy)

Consider an agent $A$ with the following task:

- "At a railroad crossing, cross the rails if no train approaches."

We may encode this scenario using one of the following two rules:

$$walk \leftarrow at(A, L), crossing(L), not\ train\_approaches(L). \qquad (r_1)$$

$$walk \leftarrow at(A, L), crossing(L), -train\_approaches(L). \qquad (r_2)$$

# Extended Logic Programs

## Definition

An *extended logic program (ELP)* is a finite set of rules

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (n, m \geq 0) \tag{3}$$

where $a$, $b_i$, $c_j$ are atoms or strongly negated atoms in a f.o. language $L$.

# Extended Logic Programs

## Definition

An *extended logic program (ELP)* is a finite set of rules

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (n, m \geq 0) \tag{3}$$

where $a$, $b_i$, $c_j$ are atoms or strongly negated atoms in a f.o. language $L$.

The semantics of ELPs can be defined by program transformation:

- view literals "$-p(\vec{X})$" as atoms with fresh predicate symbols "$-p$";
- add clauses $falsity \leftarrow not\, falsity, p(\vec{X}), -p(\vec{X})$ to $P$ ($p(\vec{X})$ and $-p(\vec{X})$ are not simultaneously true); and
- select the stable models of the resulting program (called *answer sets* of $P$).

# Extended Logic Programs

## Definition

An *extended logic program (ELP)* is a finite set of rules

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (n, m \geq 0) \tag{3}$$

where $a$, $b_i$, $c_j$ are atoms or strongly negated atoms in a f.o. language $L$.

The semantics of ELPs can be defined by program transformation:

- view literals "$-p(\vec{X})$" as atoms with fresh predicate symbols "$-p$";
- add clauses $falsity \leftarrow not\, falsity, p(\vec{X}), -p(\vec{X})$
  to $P$ ($p(\vec{X})$ and $-p(\vec{X})$ are not simultaneously true); and
- select the stable models of the resulting program (called *answer sets* of $P$).

## Answer sets $M$: *three-valued view*

Atom $a$ may be true ($a \in M$), false ($-a \in M$), or *unknown* ($a, -a \notin M$).

Strong negation (combined with weak negation) is e.g. helpful to express *default rules*.

## Example (French speaking)

$$french(luc). \qquad (f_1)$$

$$speaks(X,french) \leftarrow french(X), not\ -speaks(X,french). \qquad (r_1)$$

$$-speaks(X,french) \leftarrow thumb(X). \qquad (r_2)$$

- $r_1$ expresses that by default, French can speak French.
- Single answer set $M = \{french(luc), speaks(luc,french)\}$.

Strong negation (combined with weak negation) is e.g. helpful to express *default rules*.

## Example (French speaking)

$$french(luc). \qquad (f_1)$$

$$speaks(X,french) \leftarrow french(X), not\ -speaks(X,french). \qquad (r_1)$$

$$-speaks(X,french) \leftarrow thumb(X). \qquad (r_2)$$

- $r_1$ expresses that by default, French can speak French.
- Single answer set $M = \{french(luc), speaks(luc,french)\}$.

Note:

- ELPs are closely related to Default Logic [Reiter, 1980]
- The answer sets of $P$ correspond 1-1 to the extensions of the default theory $T = (\emptyset, \{d(C) \mid C \in P\})$ ($d(C)$ casts $C$ into a default rule).

# Disjunction

The use of disjunction is natural to express indefinite knowledge.

# Disjunction

The use of disjunction is natural to express indefinite knowledge.

## Example

- $female(X) \vee male(X) \leftarrow person(X).$
- $broken(left\_hand, tom) \vee broken(right\_hand, tom).$

# Disjunction

The use of disjunction is natural to express indefinite knowledge.

## Example

- $female(X) \vee male(X) \leftarrow person(X).$
- $broken(left\_hand, tom) \vee broken(right\_hand, tom).$

Disjunction is natural for expressing a “guess” and to create non-determinism

# Disjunction

The use of disjunction is natural to express indefinite knowledge.

## Example

- $female(X) \vee male(X) \leftarrow person(X).$
- $broken(left\_hand, tom) \vee broken(right\_hand, tom).$

Disjunction is natural for expressing a “guess” and to create non-determinism

## Example

- $ok(C) \vee -ok(C) \leftarrow component(C).$

# Minimality

- Semantics: disjunction is *minimal* (different from classical logic):

$$a \vee b \vee c.$$

Minimal models: $\{a\}$, $\{b\}$, and $\{c\}$.

# Minimality

- Semantics: disjunction is *minimal* (different from classical logic):

$$a \vee b \vee c.$$

  Minimal models: $\{a\}$, $\{b\}$, and $\{c\}$.

- actually *subset minimal*:

$$a \vee b. \qquad a \vee c.$$

  Minimal models: $\{a\}$ and $\{b, c\}$.

# Minimality

- Semantics: disjunction is *minimal* (different from classical logic):

$$a \vee b \vee c.$$

  Minimal models: $\{a\}$, $\{b\}$, and $\{c\}$.

- actually *subset minimal*:

$$a \vee b. \qquad a \vee c.$$

  Minimal models: $\{a\}$ and $\{b, c\}$.

$$a \vee b. \qquad a \leftarrow b$$

  Models $\{a\}$ and $\{a, b\}$, but only $\{a\}$ is minimal.

# Minimality

- Semantics: disjunction is *minimal* (different from classical logic):

$$a \vee b \vee c.$$

  Minimal models: $\{a\}$, $\{b\}$, and $\{c\}$.

- actually *subset minimal*:

$$a \vee b. \qquad a \vee c.$$

  Minimal models: $\{a\}$ and $\{b, c\}$.

$$a \vee b. \qquad a \leftarrow b$$

  Models $\{a\}$ and $\{a, b\}$, but only $\{a\}$ is minimal.

- but minimality is *not necessarily exclusive*:

$$a \vee b. \qquad b \vee c. \qquad a \vee c.$$

  Minimal models: $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$.

# Disjunction vs. Unstratified Negation

Reconsider the Dilbert Program $P_6$:

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

# Disjunction vs. Unstratified Negation

Reconsider the Dilbert Program $P_6$:

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

is under stable semantics equivalent to the program $P_{dd}$:

$$man(dilbert).$$
$$single(X) \vee husband(X) \leftarrow man(X).$$

The use of disjunction is more intuitive!

# Extended Logic Programs with Disjunctions

## Definition

A *extended disjunctive logic program* (EDLP) is a finite set of rules

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (k, m, n \geq 0) \qquad (4)$$

where all $a_i$, $b_j$, $c_l$ are atoms or strongly negated atoms in f.o. language $L$.

# Extended Logic Programs with Disjunctions

## Definition

A *extended disjunctive logic program* (EDLP) is a finite set of rules

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad (k, m, n \geq 0) \qquad (4)$$

where all $a_i$, $b_j$, $c_l$ are atoms or strongly negated atoms in f.o. language $L$.

Semantics:

- Answer sets of $P$ are defined similarly as for an ELP
- Differences:
  - $I$ is a model of ground (4), if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $\{a_1, \ldots, a_k, c_1, \ldots, c_n\} \cap I \neq \emptyset$
  - "$M$ is the least model of $P^M$" $\rightsquigarrow$ "$M$ is a minimal model of $P^M$" ($P^M$ may have multiple minimal models).

## Example (Disjunctive Dilbert $P_{dd}$, cont'd)

$$man(dilbert).$$
$$single(X) \vee husband(X) \leftarrow man(X).$$

As $P_{dd}$ is "*not*"-free, $grnd(P_{dd})^M = grnd(P_{dd})$ for every $M$.

Answer sets:

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

# Some Properties of EDLPs

- Every answer set of an EDLP $P$ is a minimal model of $P$ (models analogous as for ELPs)
- Different answer sets of an EDLP $P$ are incomparable
- An EDLP may have no, a single or multiple answer sets
- For EDLPs without strong negation, answer sets are models that are unfounded-free [Leone *et al.*, 1997]
- Deciding whether a propositional EDLP $P$ has some answer set is $\Sigma_2^p$-complete. $(\Sigma_2^p = \mathsf{NP}^{\mathsf{NP}})$

Disjunction adds higher problem solving capacity, it is not just syntactic sugar!

- **But:** EDLPs **can not** be regarded as a fragment of Reiter's Default Logic.

# References I


K.R. Apt, H.A. Blair, and A. Walker.

Towards a Theory of Declarative Knowledge.

In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufman, Washington DC, 1988.

Chitta Baral.

*Knowledge Representation, Reasoning and Declarative Problem Solving*.

Cambridge University Press, 2003.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov.

Complexity and Expressive Power of Logic Programming.

*ACM Computing Surveys*, 33(3):374–425, 2001.

Paolo Ferraris and Vladimir Lifschitz.

Mathematical foundations of answer set programming.

In *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, pages 615–664. College Publications, 2005.

# References II


Michael Gelfond and Vladimir Lifschitz.

The Stable Model Semantics for Logic Programming.

In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

Michael Gelfond and Vladimir Lifschitz.

Classical Negation in Logic Programs and Disjunctive Databases.

*New Generation Computing*, 9:365–385, 1991.

M. Gelfond.

Answer sets.

In B. Porter F. van Harmelen, V. Lifschitz, editor, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier, 2008.

# References III


Joohyung Lee.

A model-theoretic counterpart of loop formulas.

In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 503–508. Professional Book Center, 2005.

Nicola Leone, Pasquale Rullo, and Francesco Scarcello.

Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation.

*Information and Computation*, 135(2):69–112, June 1997.

Vladimir Lifschitz.

Answer set planning.

In *ICLP*, pages 23–37, 1999.

Vladimir Lifschitz.

Answer Set Programming and Plan Generation.

*Artificial Intelligence*, 138:39–54, 2002.

# References IV


Vladimir Lifschitz.

Twelve definitions of a stable model.

In *ICLP*, pages 37–51, 2008.

Fangzhen Lin and Yuting Zhao.

ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers.

In *AAAI/IAAI*, pages 112–, 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 V


Ilkka Niemelä (ed.).

Language Extensions and Software Engineering for ASP.

Technical Report WP3, Working Group on Answer Set Programming (WASP), IST-FET-2001-37004, September 2005.

Available at `http://www.tcs.hut.fi/Research/Logic/wasp/wp3/wasp-wp3-web/`.

Ilkka Niemelä.

Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.

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

Raymond Reiter.

A Logic for Default Reasoning.

*Artificial Intelligence*, 13(1–2):81–132, 1980.

# References VI

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf.

The Well-Founded Semantics for General Logic Programs.

*Journal of the ACM*, 38(3):620–650, 1991.