What's your preference?

And how to express and implement it in logic programming!

Torsten Schaub

University of Potsdam

What's your preference?

And how to express and implement it in logic programming!

Torsten Schaub

University of Potsdam

- Motivation
- Answer set programming
- Answer set programming with preferences
  - ⋆ Syntax
  - ⋆ Semantics
  - ⋆ Implementation
- Conclusion

Motivation

The notion of *preference* in commonsense reasoning is pervasive.

Motivation

The notion of *preference* in commonsense reasoning is pervasive.

For instance,

- in buying a car, one may prefer certain features over others;

Motivation

The notion of *preference* in commonsense reasoning is pervasive.

For instance,

- in buying a car, one may prefer certain features over others;
- in scheduling, meeting some deadlines may be more important than meeting others;

Motivation

The notion of *preference* in commonsense reasoning is pervasive.

For instance,

- in buying a car, one may prefer certain features over others;
- in scheduling, meeting some deadlines may be more important than meeting others;
- in legal reasoning, laws are subject to higher principles, like lex superior or lex posterior, which are themselves subject to "higher higher" principles;

Motivation

The notion of *preference* in commonsense reasoning is pervasive.

For instance,

- in buying a car, one may prefer certain features over others;
- in scheduling, meeting some deadlines may be more important than meeting others;
- in legal reasoning, laws are subject to higher principles, like lex superior or lex posterior, which are themselves subject to "higher higher" principles;
- etc etc . . .

Legal reasoning

The challenge!

"A person wants to find out if her security interest in a certain ship is perfected. She currently has possession of the ship. According to the Uniform Commercial Code (UCC, §9-305) a security interest in goods may be perfected by taking possession of the collateral. However, there is a federal law called the Ship Mortgage Act (SMA) according to which a security interest in a ship may only be perfected by filing a financing statement. Such a statement has not been filed. Now the question is whether the UCC or the SMA takes precedence in this case. There are two known legal principles for resolving conflicts of this kind. The principle of Lex Posterior gives precedence to newer laws. In our case the UCC is newer than the SMA. On the other hand, the principle of Lex Superior gives precedence to laws supported by the higher authority. In our case the SMA has higher authority since it is federal law."

(Gordon, 1993)

Legal reasoning

Our solution in *“ordered logic programming”*

```
    perfected :- name(ucc), possession,               not neg perfected.
neg perfected :- name(sma), ship, neg finstatement, not     perfected.

possession.   ship.   neg finstatement.

(Y < X) :- name(lex_posterior(X,Y)), newer(X,Y), not neg (Y < X).

(X < Y) :- name(lex_superior(X,Y)), state_law(X), federal_law(Y), not neg (X < Y).

newer(ucc,sma).   federal_law(sma).   state_law(ucc).

(lex_posterior(X,Y) < lex_superior(X,Y)).
```

Approaches to preference

(in alphabetical order)

Approaches to preference

(in alphabetical order)

- Baader and Hollunder
- Benferhat
- Brewka and Eiter
- Delgrande, Schaub, and Tompits
- Dimopoulos and Kakas
- Geffner and Pearl
- Gelfond and Son
- Grosof

Approaches to preference

(in alphabetical order)

- Baader and Hollunder
- Benferhat
- Brewka and Eiter
- Delgrande, Schaub, and Tompits
- Dimopoulos and Kakas
- Geffner and Pearl
- Gelfond and Son
- Grosof
- Hunter
- Marek and Truszczyński
- Prakken and Sergot
- Rintanen
- Sakama and Inoue
- Wang, Zhou, and Lin
- Zhang and Foo
- etc etc

Our focus

Our focus

Extended logic programming under *answer sets semantics*

Our focus

Extended logic programming under *answer sets semantics*

- Allows for solving all search problems within NP

Our focus

Extended logic programming under *answer sets semantics*

- Allows for solving all search problems within NP
- Allows for multiple solutions

Our focus

Extended logic programming under *answer sets semantics*

- Allows for solving all search problems within NP
- Allows for multiple solutions
- Allows for using powerful off-the-shelf systems, such as `dlv`, `nomore` and `smodels`

Approaches to preference

(in alphabetical order)

- Baader and Hollunder
- Benferhat
- Brewka and Eiter
- Delgrande, Schaub, and Tompits
- Dimopoulos and Kakas
- Geffner and Pearl
- Gelfond and Son
- Grosof
- Hunter
- Marek and Truszczyński
- Prakken and Sergot
- Rintanen
- Sakama and Inoue
- Wang, Zhou, and Lin
- Zhang and Foo
- etc etc

Why these approaches?

Why these approaches?

- ✔ Extended logic programming

Why these approaches?

- ✔ Extended logic programming
- ✘ Selection function on the set of answer sets

Why these approaches?

- ✔ Extended logic programming
- ✘ Selection function on the set of answer sets
- ✘ Complexity within NP

Extended logic programs

- A *rule*, r, is an ordered pair of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n,$$

  where $n \geq m \geq 0$, and each L_i $(0 \leq i \leq n)$ is a literal.

- An *extended logic program* is a finite *set* of rules.

Extended logic programs

- A *rule*, r, is an ordered pair of the form

$$L_0 \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n,$$

  where $n \geq m \geq 0$, and each L_i $(0 \leq i \leq n)$ is a literal.

- An *extended logic program* is a finite *set* of rules.

- Notations

$$\begin{array}{rcl} \mathit{head}(r) & = & L_0 \\ \mathit{body}(r) & = & \{L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n\} \\ \mathit{body}^+(r) & = & \{L_1, \ldots, L_m\} \\ \mathit{body}^-(r) & = & \{L_{m+1}, \ldots, L_n\} \\ r^+ & = & \mathit{head}(r) \leftarrow \mathit{body}^+(r) \end{array}$$

Extended logic programs

- A *rule*, r, is an ordered pair of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n,$$

  where $n \geq m \geq 0$, and each L_i $(0 \leq i \leq n)$ is a literal.

- An *extended logic program* is a finite *set* of rules.

- Notations

$$\begin{aligned} head(r) &= L_0 \\ body(r) &= \{L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\} \\ body^+(r) &= \{L_1, \ldots, L_m\} \\ body^-(r) &= \{L_{m+1}, \ldots, L_n\} \\ r^+ &= head(r) \leftarrow body^+(r) \end{aligned}$$

- A rule r is *defeated* by a set of literals X iff $body^-(r) \cap X \neq \emptyset$.

Answer sets

- The *reduct*, Π^X, of a program Π relative to a set X of literals is defined by

$$\Pi^X = \{r^+ \mid r \in \Pi \text{ and } body^-(r) \cap X = \emptyset\}.$$

Answer sets

- The *reduct*, Π^X, of a program Π relative to a set X of literals is defined by

$$\Pi^X = \{r^+ \mid r \in \Pi \text{ and } body^-(r) \cap X = \emptyset\}.$$

In other words, Π^X is obtained from Π by

1. deleting any rule in Π which is defeated by X and
2. deleting each literal of the form $not\ L$ occurring in the bodies of the remaining rules.

Answer sets

- The *reduct*, Π^X, of a program Π relative to a set X of literals is defined by

$$\Pi^X = \{r^+ \mid r \in \Pi \text{ and } \mathit{body}^-(r) \cap X = \emptyset\}.$$

  In other words, Π^X is obtained from Π by

  1. deleting any rule in Π which is defeated by X and
  2. deleting each literal of the form $not\ L$ occurring in the bodies of the remaining rules.

- A set X of literals is an *answer set* of a program Π iff $\mathsf{Cn}(\Pi^X) = X$ (where $\mathsf{Cn}(\cdot)$ is the usual consequence operator of basic logic programs).

Answer sets

- The *reduct*, Π^X, of a program Π relative to a set X of literals is defined by

  $$\Pi^X = \{r^+ \mid r \in \Pi \text{ and } \mathit{body}^-(r) \cap X = \emptyset\}.$$

  In other words, Π^X is obtained from Π by

  1. deleting any rule in Π which is defeated by X and
  2. deleting each literal of the form $\mathit{not}\ L$ occurring in the bodies of the remaining rules.

- A set X of literals is an *answer set* of a program Π iff $\mathrm{Cn}(\Pi^X) = X$ (where $\mathrm{Cn}(\cdot)$ is the usual consequence operator of basic logic programs).

☞ For the talk, we consider *consistent* answer sets only!

An example: n-Queens

An example: n-Queens

For $n = 4$, we get:

	Q		
			Q
Q			
		Q	

n-Queens in answer set programming

$q(X, Y)$ gives the legal positions of the queens

n-Queens in answer set programming

$q(X, Y)$ gives the legal positions of the queens

$$
\begin{aligned}
q(X, Y) &\leftarrow not \neg q(X, Y) \\
\neg q(X, Y) &\leftarrow not \;\; q(X, Y)
\end{aligned}
$$

n-Queens in answer set programming

$q(X,Y)$ gives the legal positions of the queens

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & not \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & not \;\; q(X,Y) \\
\\
 & \leftarrow & q(X,Y), q(X',Y) \\
 & \leftarrow & q(X,Y), q(X,Y') \\
 & \leftarrow & q(X,Y), q(X',Y'), |X-X'| = |Y-Y'|
\end{array}
$$

n-Queens in answer set programming

$q(X, Y)$ gives the legal positions of the queens

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & not \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & not \ \ q(X,Y) \\
\\
 & \leftarrow & q(X,Y), q(X',Y) \\
 & \leftarrow & q(X,Y), q(X,Y') \\
 & \leftarrow & q(X,Y), q(X',Y'), |X - X'| = |Y - Y'| \\
\\
 & \leftarrow & not \ \ hasq(X) \\
hasq(X) & \leftarrow & q(X,Y)
\end{array}
$$

n-Queens

(in the `smodels` language)

```
q(X,Y) :- d(X), d(Y), not negq(X,Y).
negq(X,Y) :- d(X), d(Y), not q(X,Y).

:- d(X), d(Y), d(X1), q(X,Y), q(X1,Y),  X1 != X.
:- d(X), d(Y), d(Y1), q(X,Y), q(X,Y1), Y1 != Y.
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1),
        X != X1,  Y != Y1, abs(X - X1) == abs(Y - Y1).

:- d(X), not hasq(X).
hasq(X) :-  d(X), d(Y), q(X,Y).

d(1..queens).
```

And the performance . . . ?

```
torsten@belle-ile 506 > lparse -c queens=20 queens2.lp | smodels
smodels version 2.25. Reading...done
Answer: 1
Stable Model: d(1) d(2) d(3) d(4) d(5) d(6) d(7) d(8) d(9) d(10) d(11) d(12)
d(13) d(14) d(15) d(16) d(17) d(18) d(19) d(20) q(1,16) q(2,13) q(3,6) q(4,3)
q(5,15) q(6,19) q(7,1) q(8,4) q(9,9) q(10,11) q(11,8) q(12,10) q(13,17)
q(14,2) q(15,20) q(16,18) q(17,7) q(18,5) q(19,14) q(20,12)
True
Duration: 37.810
Number of choice points: 1471
Number of wrong choices: 1464
Number of atoms: 501
Number of rules: 10100
Number of picked atoms: 304305
Number of forced atoms: 14604
Number of truth assignments: 3111768
Size of searchspace (removed): 400 (0)
```

How to express preferences?

How to express preferences?

Two options:

How to express preferences?

Two options:

Static preferences: Use an external order $<$.

Ordered logic program: $(\Pi, <)$

where Π is a logic program over $\mathcal{L}$ and

$<$ is a strict partial order over Π ;

How to express preferences?

Two options:

Static preferences: Use an external order $<$.

Ordered logic program: $(\Pi, <)$

where Π is a logic program over $\mathcal{L}$ and
$<$ is a strict partial order over Π ;

Dynamic preferences: Use a special-purpose predicate $\prec$.

Ordered logic program: Π

where Π is a logic program over $\mathcal{L} \cup \{\prec\}$ containing
rules expressing that $\prec$ is a strict partial order.

An example

Consider the following ordered logic program $(\Pi, <)$ with $\Pi = \{r_1, r_2, r_3\}$

$$\begin{array}{rrcl} r_1 : & \neg a & \leftarrow & \\ r_2 : & b & \leftarrow & \neg a, not\ c \\ r_3 : & c & \leftarrow & not\ b \end{array} \qquad \text{and} \qquad r_3 < r_2\ .$$

An example

Consider the following ordered logic program $(\Pi, <)$ with $\Pi = \{r_1, r_2, r_3\}$

$$\begin{array}{rrcl} r_1: & \neg a & \leftarrow & \\ r_2: & b & \leftarrow & \neg a, \mathit{not}\ c \\ r_3: & c & \leftarrow & \mathit{not}\ b \end{array} \qquad \text{and} \qquad r_3 < r_2\ .$$

This program has two standard answer sets,

$$\{\neg a, b\} \qquad \text{and} \qquad \{\neg a, c\}$$

An example

Consider the following ordered logic program $(\Pi, <)$ with $\Pi = \{r_1, r_2, r_3\}$

$$\begin{array}{rrcl} r_1: & \neg a & \leftarrow & \\ r_2: & b & \leftarrow & \neg a, \mathit{not}\ c \\ r_3: & c & \leftarrow & \mathit{not}\ b \end{array} \qquad \text{and} \qquad r_3 < r_2\ .$$

This program has two standard answer sets,

$$\{\neg a, b\} \qquad \text{and} \qquad \{\neg a, c\}$$

among which the green one is (usually) preferred.

Three types of preference

Three types of preference

- W-preference (Wang, Zhou, and Lin)
  - ⋆ (alternating) fixed point theory

Three types of preference

- W-preference (Wang, Zhou, and Lin)
  - ⋆ (alternating) fixed point theory
- D-preference (Delgrande, Schaub, and Tompits)
  - ⋆ order preservation (of generating rules)
  - ⋆ translation into standards programs

Three types of preference

- W-preference (Wang, Zhou, and Lin)
  - ⋆ (alternating) fixed point theory
- D-preference (Delgrande, Schaub, and Tompits)
  - ⋆ order preservation (of generating rules)
  - ⋆ translation into standards programs
- B-preference (Brewka and Eiter)
  - ⋆ dual GL-reduction (eliminating prerequisites)
  - ⋆ fixed point operator

How to define "preferred" answer sets?

Claim Standard approach, ie. "$\mathrm{Cn}(\Pi^X) = X$", doesn't work!

How to define “preferred” answer sets?

Claim Standard approach, ie. “$\mathrm{Cn}(\Pi^X) = X$”, doesn't work!

Common intuitions

How to define “preferred” answer sets?

Claim Standard approach, ie. “$\mathrm{Cn}(\Pi^X) = X$”, doesn’t work!

Common intuitions

“$<$” induces some order on rule application

How to define “preferred” answer sets?

Claim Standard approach, ie. “$\mathrm{Cn}(\Pi^X) = X$”, doesn't work!

Common intuitions

“$<$” induces some order on rule application

$\hookrightarrow$ *iterative specification* (option)

How to define "preferred" answer sets?

Claim Standard approach, ie. "$\mathrm{Cn}(\Pi^X) = X$", doesn't work!

Common intuitions

- **"$<$"** induces some order on rule application
  $\hookrightarrow$ *iterative specification* (option)
- **"$<$"** induces additional dependencies between rules

How to define “preferred” answer sets?

Claim Standard approach, ie. “$\mathrm{Cn}(\Pi^X) = X$”, doesn't work!

Common intuitions

“$<$” induces some order on rule application

$\hookrightarrow$ *iterative specification* (option)

“$<$” induces additional dependencies between rules

$\hookrightarrow$ *keep original rules*

Fixpoint definition of standard answer sets

(unfolding iterated applications of "immediate consequence operations")

Let Π be a logic program and let X be a (consistent) set of literals.

We define

$$\begin{array}{rcl} X_0 & = & \emptyset \qquad\qquad \text{and for } i \geq 0 \\ X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; r \in \Pi,\; body^+(r) \subseteq X,\; body^-(r) \cap Y = \emptyset \right\} \end{array}$$

Then, X is an *answer set* of Π if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of standard answer sets

(unfolding iterated applications of "immediate consequence operations")

Let Π be a logic program and let X be a (consistent) set of literals.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ \mathit{head}(r) \;\middle|\; r \in \Pi,\ \mathit{body}^+(r) \subseteq X,\ \mathit{body}^-(r) \cap Y = \emptyset \right\}
\end{array}
$$

Then, X is an *answer set* of Π if $X = \bigcup_{i \geq 0} X_i$.

☞ A rule $r \in \Pi$ is *active* wrt the pair (X, Y) of sets of literals,
if $\mathit{body}^+(r) \subseteq X$ and $\mathit{body}^-(r) \cap Y = \emptyset$.

Fixpoint definition of W–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be a set of literals.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X_i, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ head(r') \notin X_i \end{array} \right\}
\end{array}
$$

Then, X is a *W–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of W–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be a set of literals.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X_i, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\; r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\; head(r') \notin X_i \end{array} \right\}
\end{array}
$$

Then, X is a *W–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of W–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be a set of literals.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X_i, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ head(r') \notin X_i \end{array} \right\}
\end{array}
$$

Then, X is a *W–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of D–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be a set of literals.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X_i, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ r' \notin rule(X_i) \end{array} \right\}
\end{array}
$$

Then, X is a *D–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of D–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be a set of literals.

We define

$$X_0 = \emptyset \qquad \text{and for } i \geq 0$$

$$X_{i+1} = X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{ll} I. & r \in \Pi \text{ is active wrt } (X_i, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ r' \notin rule(X_i) \end{array} \right\}$$

Then, X is a *D–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Example

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{rcll}
r_1: & a & \leftarrow & not\ b \qquad\qquad r_2 < r_1 \\
r_2: & b & \leftarrow & \\
r_3: & a & \leftarrow &
\end{array}
$$

Π has one standard answer set: $X = \{a, b\}$.

Example

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{llcll}
r_1 : & a & \leftarrow & not\ b & \qquad\qquad r_2 < r_1 \\
r_2 : & b & \leftarrow & & \\
r_3 : & a & \leftarrow & &
\end{array}
$$

Π has one standard answer set: $X = \{a, b\}$.

- X is a W-preferred answer set.

Example

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{lcll}
r_1: & a & \leftarrow & not\ b \qquad\qquad r_2 < r_1 \\
r_2: & b & \leftarrow & \\
r_3: & a & \leftarrow &
\end{array}
$$

Π has one standard answer set: $X = \{a, b\}$.

- X is a W-preferred answer set.
- Π has no D-preferred answer sets.

Fixpoint definition of B–preferred answer sets

Fixpoint definition of B–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be an answer set of Π.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ \mathit{head}(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ \mathit{head}(r') \notin X_i \end{array} \right\}
\end{array}
$$

Then, X is a *B–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Fixpoint definition of B–preferred answer sets

Let $(\Pi, <)$ be an ordered logic program and let X be an answer set of Π.

We define

$$
\begin{array}{rcl}
X_0 & = & \emptyset \qquad \text{and for } i \geq 0 \\
X_{i+1} & = & X_i \cup \left\{ head(r) \;\middle|\; \begin{array}{rl} I. & r \in \Pi \text{ is active wrt } (X, X) \text{ and} \\ II. & \text{there is no rule } r' \in \Pi \text{ with } r < r' \\ & \text{such that} \\ & (a)\ r' \text{ is active wrt } (X, X_i) \text{ and} \\ & (b)\ head(r') \notin X_i \end{array} \right\}
\end{array}
$$

Then, X is a *B–preferred* answer set if $X = \bigcup_{i \geq 0} X_i$.

Example

(Brewka and Eiter, 1999)

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{rrcl}
r_1: & b & \leftarrow & a, not\ \neg b \\
r_2: & \neg b & \leftarrow & not\ b \\
r_3: & a & \leftarrow & not\ \neg a
\end{array}
\qquad\qquad r_3 < r_2 < r_1
$$

Π has two standard answer sets: $X = \{a, b\}$ and $X' = \{a, \neg b\}$.

Example

(Brewka and Eiter, 1999)

Consider ordered logic program $(\Pi, <)$:

$$\begin{array}{rrcl} r_1: & b & \leftarrow & a, not\ \neg b \\ r_2: & \neg b & \leftarrow & not\ b \\ r_3: & a & \leftarrow & not\ \neg a \end{array} \qquad r_3 < r_2 < r_1$$

Π has two standard answer sets: $X = \{a, b\}$ and $X' = \{a, \neg b\}$.

- X is the unique B-preferred answer set.

Example

(Brewka and Eiter, 1999)

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{rrcl}
r_1: & b & \leftarrow & a, not\ \neg b \\
r_2: & \neg b & \leftarrow & not\ b \\
r_3: & a & \leftarrow & not\ \neg a
\end{array}
\qquad\qquad r_3 < r_2 < r_1
$$

Π has two standard answer sets: $X = \{a, b\}$ and $X' = \{a, \neg b\}$.

- X is the unique B-preferred answer set.
- Π has no W- and D-preferred answer sets.

Example

(Baader and Hollunder, 1993)

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{lrcl}
r_1 : & \neg f & \leftarrow & p, not\ f \\
r_2 : & w & \leftarrow & b, not\ \neg w \\
r_3 : & f & \leftarrow & w, not\ \neg f \\
r_4 : & b & \leftarrow & p \\
r_5 : & p & \leftarrow &
\end{array}
\qquad r_2 < r_1
$$

Π has two standard answer sets:

$$X = \{p, b, \neg f, w\} \qquad \text{and} \qquad X' = \{p, b, f, w\} \ .$$

Example

(Baader and Hollunder, 1993)

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{rrcll}
r_1: & \neg f & \leftarrow & p, not\ f & \qquad\qquad r_2 < r_1 \\
r_2: & w & \leftarrow & b, not\ \neg w & \\
r_3: & f & \leftarrow & w, not\ \neg f & \\
r_4: & b & \leftarrow & p & \\
r_5: & p & \leftarrow & &
\end{array}
$$

Π has two standard answer sets:

$$X = \{p, b, \neg f, w\} \qquad \text{and} \qquad X' = \{p, b, f, w\} \ .$$

- X is the unique W- and D-preferred answer set.

Example

(Baader and Hollunder, 1993)

Consider ordered logic program $(\Pi, <)$:

$$\begin{array}{rrcl}
r_1 : & \neg f & \leftarrow & p, not\ f \\
r_2 : & w & \leftarrow & b, not\ \neg w \\
r_3 : & f & \leftarrow & w, not\ \neg f \\
r_4 : & b & \leftarrow & p \\
r_5 : & p & \leftarrow &
\end{array} \qquad r_2 < r_1$$

Π has two standard answer sets:

$$X = \{p, b, \neg f, w\} \qquad \text{and} \qquad X' = \{p, b, f, w\}\ .$$

- X is the unique W- and D-preferred answer set.
- X and X' are both B-preferred answer sets.

Is there a lesson to be learned. . . ?

Is there a lesson to be learned. . . ?

Let $(\Pi, <)$ be an ordered logic program.

Then, we have:

$$\mathcal{AS}_{\mathsf{D}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{W}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{B}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}(\Pi)$$

where $\mathcal{AS}(\Pi)$ — set of standard answer sets

$\mathcal{AS}_P(\Pi, <)$ — set of "P–preferred answer sets"

Is there a lesson to be learned. . . ?

Let $(\Pi, <)$ be an ordered logic program.

Then, we have:

$$\mathcal{AS}_{\mathsf{D}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{W}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{B}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}(\Pi)$$

where $\mathcal{AS}(\Pi)$ — set of standard answer sets

$\mathcal{AS}_P(\Pi, <)$ — set of "P–preferred answer sets"

Roughly, the hierarchy is induced by a decreasing interaction between *groundedness* and *preference*:

Is there a lesson to be learned. . . ?

Let $(\Pi, <)$ be an ordered logic program.

Then, we have:

$$\mathcal{AS}_{\mathsf{D}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{W}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}_{\mathsf{B}}(\Pi, <) \quad \subseteq \quad \mathcal{AS}(\Pi)$$

where $\mathcal{AS}(\Pi)$ — set of standard answer sets

$\mathcal{AS}_P(\Pi, <)$ — set of "P–preferred answer sets"

Roughly, the hierarchy is induced by a decreasing interaction between *groundedness* and *preference*:

D-preference	full compatibility
W-preference	weak compatibility
B-preference	no compatibility

Implementation

for *dynamically* ordered logic programs

Implementation

for *dynamically* ordered logic programs

Idea Translate a logic program Π with preference information into a standard logic program $\mathcal{T}(\Pi)$ such that answers to $\mathcal{T}(\Pi)$ respect the preferences in Π.

Implementation

for *dynamically* ordered logic programs

Idea Translate a logic program Π with preference information into a standard logic program $\mathcal{T}(\Pi)$ such that answers to $\mathcal{T}(\Pi)$ respect the preferences in Π.

Plan
1. Extend the language for expressing preference
2. Add axioms encoding specific preference handling strategies

(Dynamically) ordered logic programs

An *ordered logic program* is an extended logic program over a propositional language $\mathcal{L}$,

containing the following pairwise disjoint categories:

- a set $\mathcal{N}$ of terms serving as *names* for rules;
- a set $\mathbf{A}$ of regular (propositional) atoms of a program; and
- a set $\mathbf{A}_{\prec}$ of *preference atoms* $s \prec t$, where $s, t \in \mathcal{N}$ are names.

(Dynamically) ordered logic programs

An *ordered logic program* is an extended logic program over a propositional language $\mathcal{L}$,

containing the following pairwise disjoint categories:

- a set $\mathcal{N}$ of terms serving as *names* for rules;
- a set $\mathbf{A}$ of regular (propositional) atoms of a program; and
- a set $\mathbf{A}_{\prec}$ of *preference atoms* $s \prec t$, where $s, t \in \mathcal{N}$ are names.

For each ordered program Π, we require a bijective function $n(\cdot)$ assigning to each rule $r \in \Pi$ a name $n(r) \in \mathcal{N}$.

To simplify our notation, we write

- n_r instead of $n(r)$ or n_i instead of n_{r_i} and
- $t : r$ instead of $t = n(r)$.

Towards preferred answer sets

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

   $\mathsf{bl}(n_r)$ signifies that rule r is blocked wrt X, that is, either

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

   $\mathsf{bl}(n_r)$ signifies that rule r is blocked wrt X, that is, either

   - $body^+(r) \not\subseteq X$ or
   - $body^-(r) \cap X \neq \emptyset$.

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

   $\mathsf{bl}(n_r)$ signifies that rule r is blocked wrt X, that is, either

   - $body^+(r) \not\subseteq X$ or
   - $body^-(r) \cap X \neq \emptyset$.

   $\mathsf{ok}(n_r)$ signifies that it is "ok" to consider rule r

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

   $\mathsf{bl}(n_r)$ signifies that rule r is blocked wrt X, that is, either

   - $body^+(r) \not\subseteq X$ or
   - $body^-(r) \cap X \neq \emptyset$.

   $\mathsf{ok}(n_r)$ signifies that it is "ok" to consider rule r

2. Provide axioms that guarantee a consideration of rules that is in accord with the underlying preference information, that is,

   $n_r \prec n_{r'}$ enforces that $\mathsf{ok}(n_{r'})$ is derivable "before" $\mathsf{ok}(n_r)$

Towards preferred answer sets

1. Introduce special purpose predicates controlling rule application:

   $\mathsf{ap}(n_r)$ signifies that rule r is applicable wrt X, that is,

   - $body^+(r) \subseteq X$ and
   - $body^-(r) \cap X = \emptyset$.

   $\mathsf{bl}(n_r)$ signifies that rule r is blocked wrt X, that is, either

   - $body^+(r) \not\subseteq X$ or
   - $body^-(r) \cap X \neq \emptyset$.

   $\mathsf{ok}(n_r)$ signifies that it is "ok" to consider rule r

2. Provide axioms that guarantee a consideration of rules that is in accord with the underlying preference information, that is,

   $n_r \prec n_{r'}$ enforces that $\mathsf{ok}(n_{r'})$ is derivable "before" $\mathsf{ok}(n_r)$

3. Specify what it means that a rule "has been considered"

Translating ordered logic programs

according to *D-preference*

Let $\Pi = \{r_1, \ldots, r_k\}$ be an ordered logic program over $\mathcal{L}$.

Let $\mathcal{L}^\star$ be the language obtained from $\mathcal{L}$ by adding, for each $r, r' \in \Pi$, new pairwise distinct propositional atoms $\mathsf{ap}(n_r)$, $\mathsf{bl}(n_r)$, $\mathsf{ok}(n_r)$, and $\mathsf{rdy}(n_r, n_{r'})$.

Then, the logic program $\mathcal{T}(\Pi)$ over $\mathcal{L}^\star$ contains the following rules, for each $r \in \Pi$, where $L^+ \in body^+(r)$, $L^- \in body^-(r)$, and $r', r'' \in \Pi$:

Then, the logic program $\mathcal{T}(\Pi)$ over $\mathcal{L}^\star$ contains the following rules, for each $r \in \Pi$, where $L^+ \in \mathit{body}^+(r)$, $L^- \in \mathit{body}^-(r)$, and $r', r'' \in \Pi$:

$$
\begin{array}{lrcl}
a_1(r): & \mathit{head}(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), \mathit{body}(r)
\end{array}
$$

Then, the logic program $\mathcal{T}(\Pi)$ over $\mathcal{L}^\star$ contains the following rules, for each $r \in \Pi$, where $L^+ \in \mathit{body}^+(r)$, $L^- \in \mathit{body}^-(r)$, and $r', r'' \in \Pi$:

$$
\begin{array}{rrcl}
a_1(r): & \mathit{head}(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), \mathit{body}(r) \\
b_1(r, L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), \mathit{not}\ L^+ \\
b_2(r, L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), L^-
\end{array}
$$

Then, the logic program $\mathcal{T}(\Pi)$ over $\mathcal{L}^\star$ contains the following rules, for each $r \in \Pi$, where $L^+ \in body^+(r)$, $L^- \in body^-(r)$, and $r', r'' \in \Pi$:

$$\begin{array}{rrcl}
a_1(r): & head(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body(r) \\
b_1(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), not\ L^+ \\
b_2(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), L^- \\
\\
c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \\
c_2(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & not\ (n_r \prec n_{r'}) \\
c_3(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{ap}(n_{r'}) \\
c_4(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{bl}(n_{r'})
\end{array}$$

Then, the logic program $\mathcal{T}(\Pi)$ over $\mathcal{L}^\star$ contains the following rules, for each $r \in \Pi$, where $L^+ \in \mathit{body}^+(r)$, $L^- \in \mathit{body}^-(r)$, and $r', r'' \in \Pi$:

$$
\begin{array}{rrcl}
a_1(r): & \mathit{head}(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), \mathit{body}(r) \\
b_1(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), \mathit{not}\ L^+ \\
b_2(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), L^- \\[1ex]
c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \\
c_2(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & \mathit{not}\ (n_r \prec n_{r'}) \\
c_3(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{ap}(n_{r'}) \\
c_4(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{bl}(n_{r'}) \\[1ex]
t(r,r',r''): & n_r \prec n_{r''} & \leftarrow & n_r \prec n_{r'}, n_{r'} \prec n_{r''} \\
as(r,r'): & \neg(n_{r'} \prec n_r) & \leftarrow & n_r \prec n_{r'}
\end{array}
$$

Translating ordered logic programs

according to *W-preference*

Translating ordered logic programs

according to *W-preference*

$$
\begin{array}{rrcl}
a_1(r): & head(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body(r) \\
b_1(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), not\ L^+ \\
b_2(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), L^- \\
\\
c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \dots, \mathsf{rdy}(n_r, n_{r_k}) \\
c_2(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & not\ (n_r \prec n_{r'}) \\
c_3(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{ap}(n_{r'}) \\
c_4(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{bl}(n_{r'}) \\
c_5(r,r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), head(r') \\
\\
t(r,r',r''): & n_r \prec n_{r''} & \leftarrow & n_r \prec n_{r'}, n_{r'} \prec n_{r''} \\
as(r,r'): & \neg(n_{r'} \prec n_r) & \leftarrow & n_r \prec n_{r'}
\end{array}
$$

Translating ordered logic programs

according to *B-preference*

Translating ordered logic programs

according to *B-preference*

$$
\begin{array}{lllrcl}
\Pi & + & a_1(r): & head(r') & \leftarrow & \mathsf{ap}(n_r) \\
 & & a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body(r), not\ body^-(r') \\
 & & b_1(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), not\ L, not\ L' \\
 & & b_2(r,K): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), K, K' \\
 & & c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \\
 & & c_2(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & not\ (n_r \prec n_s) \\
 & & c_3(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & (n_r \prec n_s), \mathsf{ap}(n_s) \\
 & & c_4(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & (n_r \prec n_s), \mathsf{bl}(n_s) \\
 & & c_5(r,s,J): & \mathsf{rdy}(n_r, n_s) & \leftarrow & head(s), J \\
 & & d(r): & & \leftarrow & not\ \mathsf{ok}(n_r) \\
 & & t(r,s,t): & n_r \prec n_t & \leftarrow & n_r \prec n_s, n_s \prec n_t \\
 & & as(r,s): & \neg(n_s \prec n_r) & \leftarrow & n_r \prec n_s
\end{array}
$$

An(other) example

Consider the following ordered logic program $\Pi = \{r_1, r_2, r_3, r_4\}$:

$$\begin{array}{rcrcl} r_1 & = & \neg a & \leftarrow & \\ r_2 & = & b & \leftarrow & \neg a, not\ c \\ r_3 & = & c & \leftarrow & not\ b \\ r_4 & = & n_3 \prec n_2 & \leftarrow & not\ d \end{array}$$

where n_i denotes the name of rule r_i $(i = 1, \ldots, 4)$.

This program has two answer sets, $\{\neg a, b, n_3 \prec n_2\}$ and $\{\neg a, c, n_3 \prec n_2\}$.

An(other) example

Consider the following ordered logic program $\Pi = \{r_1, r_2, r_3, r_4\}$:

$$\begin{array}{rcrcl} r_1 & = & \neg a & \leftarrow & \\ r_2 & = & b & \leftarrow & \neg a, not\ c \\ r_3 & = & c & \leftarrow & not\ b \\ r_4 & = & n_3 \prec n_2 & \leftarrow & not\ d \end{array}$$

where n_i denotes the name of rule r_i $(i = 1, \ldots, 4)$.

This program has two answer sets, $\{\neg a, b, n_3 \prec n_2\}$ and $\{\neg a, c, n_3 \prec n_2\}$.

The example ran through our implementation

Ordered logic program $\Pi = \{r_1, r_2, r_3, r_4\}$:

$$
\begin{array}{rcrcl}
r_1 & = & \neg a & \leftarrow & \\
r_2 & = & b & \leftarrow & \neg a, not\ c \\
r_3 & = & c & \leftarrow & not\ b \\
r_4 & = & n_3 \prec n_2 & \leftarrow & not\ d
\end{array}
$$

becomes

```
neg a.
b :- name(n2), neg a, not c.
c :- name(n3), not b.
(n3 < n2) :- not d.
```

The outcome

```
neg_a.
b :- ap(n2).
ap(n2) :- ok(n2), neg_a, not c.
bl(n2) :- ok(n2), not neg_a.
bl(n2) :- ok(n2), c.
c :- ap(n3).
ap(n3) :- ok(n3), not b.
bl(n3) :- ok(n3), b.
prec(n3, n2) :- not d.
ok(N) :- name(N), oko(N, n2), oko(N, n3).
oko(N, M) :- name(N), name(M), not prec(N, M).
oko(N, M) :- name(N), name(M), prec(N, M), ap(M).
oko(N, M) :- name(N), name(M), prec(N, M), bl(M).
neg_prec(M, N) :- name(N), name(M), prec(N, M).
prec(N, M) :- name(N), name(M), name(O),
              prec(N, O), prec(O, M).
false :- a, neg_a. false :- b, neg_b. false :- c, neg_c. false :- d, neg_d.
false :- name(N), name(M), prec(N, M), neg_prec(N, M).
name(n3). name(n2).
```

Computing preferred answer sets

```
?- lp2dlv('Examples/example').

yes
?- dlv('Examples/example').
dlv [build BEN/Apr  5 2000   gcc 2.95.2 19991024 (release)]

{name(n2), name(n3), neg_a, ok(n2), oko(n2,n2), oko(n2,n3), oko(n3,n3),
 prec(n3,n2), neg_prec(n2,n3), ap(n2), b, oko(n3,n2), ok(n3), bl(n3)}

yes
?- dlv('Examples/example',nice).
dlv [build BEN/Apr  5 2000   gcc 2.95.2 19991024 (release)]

{neg_a, b}

yes
?-
```

`dlv` is an off-the-shelf logic programming/deductive database system

Implementation

plp `http://www.cs.uni-potsdam.de/~torsten/plp`

- Front-end to `dlv` and `smodels`
  1. plp: OLP $\mapsto$ LP
  2. `dlv`/`smodels`: LP $\mapsto$ Answer sets
- Ordered logic programs

  eg. $n_{17} \prec n_{42} \leftarrow \; n_{17} \prec n_{34}, \; not \; (n_{42} \prec n_{34})$
- Ordered logic programs with variables

  eg. $n_1(x) \prec n_2(y) \leftarrow \; p(y), \; not \; (x = c)$
- Disjunctive logic programming
  - ⋆ preferences about disjunctive rules
  - ⋆ disjunctive preferences,

    eg. $(r_2 \prec r_{42}) \vee (r_4 \prec r_{42}) \leftarrow \neg a$

Conclusion

Conclusion

- Methodology for
  - ⋆ specifying and
  - ⋆ implementing

  preferences within answer set programming

Conclusion

- Methodology for
  - specifying and
  - implementing

  preferences within answer set programming
- Elaboration upon three different approaches that were originally defined in rather different ways

Conclusion

- Methodology for
  - specifying and
  - implementing

  preferences within answer set programming
- Elaboration upon three different approaches that were originally defined in rather different ways
- Uniformity provides us with a deeper understanding of how and which answer sets are preferred in each approach

Conclusion

- Methodology for
  - ⋆ specifying and
  - ⋆ implementing

  preferences within answer set programming
- Elaboration upon three different approaches that were originally defined in rather different ways
- Uniformity provides us with a deeper understanding of how and which answer sets are preferred in each approach
- In particular, this is reflected in the compilation framework used for implementing preferences

Acknowledgements

(in alphabetical order)

Acknowledgements

(in alphabetical order)

Farid Benhammadi	Philippe Besnard
Jim Delgrande	Pascal Nicolas
Hans Tompits	Kewen Wang

Acknowledgements

(in alphabetical order)

Farid Benhammadi	Philippe Besnard
Jim Delgrande	Pascal Nicolas
Hans Tompits	Kewen Wang

Thank you all very much!

Example

Consider ordered logic program $(\Pi, <)$:

$$\begin{array}{llll} r_1: & a & \leftarrow & not\ b \\ r_2: & b & \leftarrow & \end{array} \qquad r_2 < r_1$$

Π has one standard answer set: $X = \{a, b\}$.

Example

Consider ordered logic program $(\Pi, <)$:

$$\begin{array}{llll} r_1: & a & \leftarrow & not\ b \\ r_2: & b & \leftarrow & \end{array} \qquad\qquad r_2 < r_1$$

Π has one standard answer set: $X = \{a, b\}$.

- Π has no W-, D- and B–preferred answer sets.

Example

Consider ordered logic program $(\Pi, <)$:

$$\begin{array}{llll} r_1: & a & \leftarrow & b \\ r_2: & b & \leftarrow & \end{array} \qquad\qquad r_2 < r_1$$

Π has one standard answer set: $X = \{a, b\}$.

Example

Consider ordered logic program $(\Pi, <)$:

$$
\begin{array}{llll}
r_1: & a & \leftarrow & b \qquad\qquad\qquad r_2 < r_1 \\
r_2: & b & \leftarrow &
\end{array}
$$

Π has one standard answer set: $X = \{a, b\}$.

- X is B–preferred.
- Π has no W– and D–preferred answer sets.