Preferred Answer Sets: Comparison of Generating Sets

Alexander Šimko
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava
Mlynská dolina, 824 48 Bratislava, Slovakia

Abstract

Logic programming under answer set semantics is a favourite tool for knowledge representation. If a logic program contains conflicting rules, we want to resolve the conflicts. Preferences are usually used to encode additional principles for conflict resolution. They state which of the conflicting rules are applicable.

Different semantics for preferential reasoning have been proposed. They usually have difficulties to ignore some unnatural preferences: (i) when the preferences are not compatible with an order, in which rules have to be applied, (ii) when the preferences are specified on rules that are not applicable under answer set semantics, (iii) when the preferences are specified on non-conflicting rules. In such situations existing approaches lead to counter-intuitive conclusions. On the other hand, approach of (Šefránek 2008) that is able to ignore preferences on non-applicable rules is too restrictive, and ignores some preferences between conflicting rules.

In this paper, we address this issue by proposing semantics for preferential reasoning that is able to handle aforementioned situations. Our approach is based on the following notions: (i) comparison of generating rules, (ii) detection which rules are conflicting, and (iii) splitting a program into components, in which the comparisons are performed.

Introduction

Knowledge base encoded as an extended logic program under answer set semantics (Gelfond and Lifschitz 1991) is simply a set of "if then" rules. It is quite natural that a knowledge base contains conflicting rules (mutually exclusive rules). In order to resolve conflicts between rules, additional principles are used. In a law domain, for example, a rule from a law with higher authority takes precedence. Such additional principles are traditionally encoded using a preference relation on rules.

Different semantics for logic programs with preferences have been proposed. A common approach to preference handling is to select some of the answer sets to be preferred ones. Otherwise, the approaches usually differ a lot. (Brewka and Eiter 1999), (Delgrande, Schaub, and Tompits 2003) and (Wang, Zhou, and Lin 2000) understand preferences as an order, in which rules have to be applied. A rule is applied if all preferred rules have been applied or have been defeated in a preference aware manner. (Zhang and Foo 1997) uses different approach. Preference handling is understood as a removal of defeated less preferred rules. At each step we remove from a program a less preferred rule that is defeated by the remainder of the program. (Sakama and Inoue 2000) works with preferences on literals. It also provides a way to transform preferences on rules to preferences on literals. Preferred answer set selection is then done via comparison of answer sets. (Šefránek 2008) defines an argumentation framework. The rules of a program are the basic argumentation structures, and the preferences on rules are used to form the basic attacks between argumentation structures. The derivation rules are used to derive argumentation structures and attacks. Special type of argumentation structures, complete argumentation structures, correspond to answer sets. Complete argumentation structures that are not attacked represent preferred answer sets.

None of the approaches we are aware of is able to fully handle the following situations:

- when the preferences are not compatible with an order, in which rules have to be applied (Example 1, Example 2),
- when the preferences are specified on rules that are not applicable under answer set semantics (Example 1, Example 3),
- when the preferences are specified on non-conflicting rules (Example 1, Example 3, Example 4),
- when preferences are specified on conflicting rules (Example 5).

We illustrate these areas on the following examples.

Example 1. *Consider the program from Example 5.5 from (Brewka and Eiter 1999).*

$r_1 : c \leftarrow not\ b$ $\qquad$ r_1 *is preferred over* r_2
$r_2 : b \leftarrow not\ a$

The program is stratified, and its semantics is well defined. First, we need to know whether b *holds to decide whether* c *holds. Therefore the rule* r_1 *can be used only after* r_2*. Moreover, there is no conflict between* r_1 *and* r_2 *as they produce the unique answer set* $\{b\}$*. Hence* $\{b\}$ *should be the unique preferred answer set. According to (Brewka and Eiter 1999), (Delgrande, Schaub, and Tompits 2003) and (Wang, Zhou, and Lin 2000) the program has no preferred answer set.*

Example 2. *Consider the program 5.1 from (Brewka and Eiter 1999).*

$r_1 : b \leftarrow not\ \neg b, a$ $\quad$ *r_1 is preferred over r_2*
$r_2 : \neg b \leftarrow not\ b$ $\quad$ *r_2 is preferred over r_3*
$r_3 : a \leftarrow not\ \neg a$

There is no way to derive $\neg a$, hence r_3 can be applied. r_1 and r_2 depend on each other, and r_1 depends on r_3. Also the only conflict is between rules r_1 and r_2. Therefore we first apply rule r_3 and then we decide whether we use rule r_1 or r_2. Since r_1 is preferred $\{b, a\}$ should be the unique preferred answer set.

(Delgrande, Schaub, and Tompits 2003) and (Wang, Zhou, and Lin 2000) have no preferred answer sets. They require that r_1 is applied before r_3 but that cannot be done, as r_3 is the only rule that derives the prerequisite a of r_1.

Example 3. *Consider the following program*

$r_1 : a \leftarrow not\ b$ $\quad$ *r_1 is preferred over r_2*
$r_2 : b \leftarrow not\ a$

$r_3 : inc \leftarrow a, not\ inc$

The rule r_1 is preferred. However, it cannot be applied, as the answer sets containing literal a are ruled out by the integrity constraint r_3.

According to our view, the unique answer set $\{b\}$ should be the unique preferred answer set. (Brewka and Eiter 1999), (Wang, Zhou, and Lin 2000), and (Delgrande, Schaub, and Tompits 2003) provide no preferred answer set, since they require that r_1 is applied.

Example 4. *Consider the following program*

$r_1 : a \leftarrow not\ b$ $\quad$ *r_3 is preferred over r_1*
$r_2 : b \leftarrow not\ a$
$r_3 : c \leftarrow b$

The only conflict is between the rules r_1 and r_2. The rules r_1 and r_3 are not in conflict. The head of the rule r_3 is not in the body of the rule r_1, and vice versa. Moreover, $not\ b$ is the precondition of r_1, and b is the precondition of r_3. Hence the rules are applicable in different "situations". Hence the preference should not be "effectual". Both $\{a\}$ and $\{b, c\}$ should be preferred answer sets.

(Zhang and Foo 1997) and (Sakama and Inoue 2000) are unable to detect that r_2 and r_3 are not in conflict, and select $\{b, c\}$ as the unique preferred answer set of the program.

Example 5. *Consider the program*

$r_1 : a \leftarrow not\ \neg a$ $\quad$ *r_1 is preferred over r_2*
$r_2 : b \leftarrow not\ \neg b$
$r_3 : \neg b \leftarrow a$
$r_4 : \neg a \leftarrow b$

The rules r_1 and r_2 are in conflict. This conflict is indirect via r_3 and r_4. The rule r_1 being preferred, $\{a, \neg b\}$ should be the unique preferred answer set.

The only preference is ignored in (Šefránek 2008) as the head of the rule r_1 is not default negated in the body of the rule r_2.

The contribution of the paper is proposed semantics that addresses aforementioned situations, i.e.:

- it ignores preferences between rules that are not in conflict,
- takes into account preferences between conflicting rules,
- it ignores preferences on rules that cannot be applied under answer set semantics.

It also provides a preferred answer set when a standard answer set of a program exists.

We want to note that these ideas are to some extent present in existing approaches. (Brewka and Eiter 1999), (Wang, Zhou, and Lin 2000), (Delgrande, Schaub, and Tompits 2003), (Zhang and Foo 1997) are able to ignore some unnatural preferences. On the other hand (Šefránek 2008) is too restrictive, and ignores preferences between conflicting rules in some situations. (Šefránek 2008) is also able to ignore preferences on rules that are not applicable under answer set semantics. However, none of the approaches satisfactorily handles all the mentioned situations.

Our approach is based on the following notions

- comparison of generating rules,
- detection which rules are conflicting, and
- splitting a program into components, in which the comparisons are performed.

The rest of the paper is organized as follows. First, we give the preliminaries and notation. Second, we informally present our approach on an example. Then, we give the formal presentation of the approach, and we present the properties of the approach. After that we compare our approach to other approaches and conclude the paper.

Preliminaries and Notation

We use extended logic programs. Let At be the set of *atoms*. A *literal* is either a or $\neg a$ where a is an atom. The set of all *literals* is Lit.

A *rule* is an expression of the form $L \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$, where $m, n \in \mathbb{N}$, $m \leq n$, $L, L_i \in Lit$.

If r is a rule of the form as above, then L, the *head of the rule* is denoted by $head(r)$, $\{L_1, \ldots, not\ L_n\}$ is called the *body* of the rule, and the literals in the body of the rule $\{L_1, \ldots, L_n\}$ are denoted by $body^L(r)$.

Also positive part of the rule $\{L_1, \ldots, L_m\}$ is denoted by $body^+(r)$, and negative part $\{L_{m+1}, \ldots, L_n\}$ is denoted by $body^-(r)$.

If r_1, r_2 are rules, we say that r_1 *blocks* r_2 iff $head(r_1) \in body^-(r_2)$.

An *extended logic program* is a finite set of rules. For simplicity, we will use the term *logic program* instead of *extended logic program*.

If P is a logic program and X is a set of literal, we will use $P \cup X$ to denote the program $P \cup \{x \leftarrow : x \in X\}$.

Two literals $a, b \in Lit$ are *contradictory* iff $a = \neg b$. A set $S \subseteq Lit$ of literals is *inconsistent* iff there are contradictory literals $a, b \in S$. Otherwise it is consistent.

A set of literals S *satisfies* a rule r iff $head(r) \in S$ whenever $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$. A set of literals

S is a *model* of a logic program P iff: (i) S satisfies every rule $r \in P$, and (ii) S is consistent or $S = Lit$. We will denote the least model of a logic program P (under $\subset$ ordering) by $\mathcal{M}_0(P)$.

Let r be a rule. Then r^+ is $head(r) \leftarrow body^+(r)$. If P is a logic program then $P^+ = \{r^+ : r \in P\}$. A logic program P is *definite* iff $P = P^+$.

Let R be a set of rules and S be a set of literals. Then Gelfond-Lifschitz transformation $R^S = \{r^+ : r \in R \wedge body^-(r) \cap S = \emptyset\}$.

A set of literals S is an answer set of a logic program P iff $S = \mathcal{M}_0(P^S)$. We will denote the set of all answer sets of a logic program P by $\mathcal{AS}(P)$.

Let A be an answer set of P. Then the set of all generating rules of A is the set $\mathcal{AGR}_P(A) = \{r \in P : body^+(r) \subseteq A \wedge body^-(r) \cap A = \emptyset\}$. Set of all generating rules of a program P is the set $\mathcal{AGR}(P) = \bigcup_{A \in \mathcal{AS}(P)} \mathcal{AGR}_P(A)$.

Definition 1 (Preference relation on rules)**.** *Let R be a set of rules. Relation $<\subseteq R \times R$ is a* preference relation *on rules iff it is (i) irreflexive, (ii) asymmetric, and (iii) transitive. If $r_1 < r_2$ we say that r_2 is preferred over r_1.*

Definition 2 (Program with preferences)**.** Program with preferences *is a pair $(P, <)$ where P is a logic program and $<$ is a preference relation on rules.*

Informal Presentation

In this section we informally present our approach on the following example.

Example 6. *Consider the program*

$r_1 : a \leftarrow not\ b$ $\qquad$ $r_2 < r_1$
$r_2 : b \leftarrow not\ a$ $\qquad$ $r_4 < r_3$
$r_3 : c \leftarrow a, not\ d$
$r_4 : d \leftarrow a, not\ c$

The program has three answer sets: $A_1 = \{a, c\}$ generated by the rules $R_1 = \{r_1, r_3\}$, $A_2 = \{a, d\}$ generated by the rules $R_2 = \{r_1, r_4\}$, and $A_3 = \{b\}$ generated by the rule $R_3 = \{r_2\}$.

We consider the rules r_1 and r_2 to be conflicting as $head(r_1) \in body^-(r_2)$, $head(r_2) \in body^-(r_1)$, and $r_1 \in R_1$ and $r_2 \in R_3$ are used to generate the answer sets. Also the rules r_3 and r_4 are conflicting.

We adopt the view, that the preference handling is a selection between different generating sets. However, we do not want to compare the whole sets R_1, R_2, R_3. Structure of the rules reveals that there are two decision points:

- *whether we use r_1 or r_2, and*
- *whether we use r_3 or r_4 (if we choose r_1).*

These two decision points represent different concerns, and as a good practice, we do not want to mix them. We split the program into components $\Pi_1 = \{r_1, r_2\}$ and $\Pi_2 = \{r_3, r_4\}$. Π_1 has two answer sets: $B_1 = \{a\}$ and $B_2 = \{b\}$, generated by the rules $R_{B_1} = \{r_1\}$, and $R_{B_2} = \{r_2\}$, respectively. Using the preference $r_2 < r_1$ we select B_1 as a preferred answer set of the component $\Pi_1 - R_{B_1}$ contains a preferred rule. Using B_1, the component Π_2 has two answer sets $C_1 = \{c\}$ and $C_2 = \{d\}$ generated by the rules $R_{C_1} = \{r_3\}$ and $R_{C_2} = \{r_4\}$ respectively. R_{C_1} contains a preferred rule, hence we select C_1 as a preferred answer set of the component Π_2. Finally, $A_1 = B_1 \cup C_1$ is the unique preferred answer set of the program.

In the next sections we formally define the approach. It is defined in the three steps: 1. a program is split into components in order to separate the rules, 2. we detect conflicting rules, 3. we select the preferred answer sets of the components via comparison between generating sets.

Splitting – summary

In order to split a program into components, we use the splitting developed by Lifschitz and Turner. In this section we summarize the splitting presented in (Lifschitz and Turner 1994). However, for the technical reasons we use the modified definitions. The important difference between our and the original version of splitting it that we require a sequence of splitting sets to be total in a certain sense. It allow us to split a program as much as possible. For the formal formulation see Definition 4.

Definition 3 (Splitting set)**.** *A set S of literals is a* splitting set *for a program P iff for every $r \in P$ we have that if $head(r) \in S$ then $body^L(r) \subseteq S$.*

Definition 4 (Splitting)**.** *Let P be a logic program. Then a finite sequence of rules $\langle \Pi_0, \ldots, \Pi_n \rangle$ is called a* splitting *of P iff there is a sequence $\langle S_0, \ldots, S_n \rangle$ of splitting sets such that:*

- *$S_0 = \emptyset$,*
- *$S_i \subset S_{i+1}$ for $0 \leq i < n$,*
- *$S_n = Lit$, and*
- *for each $0 \leq i < n$ there is no splitting set S such that $S_i \subset S \subset S_{i+1}$,*
- *$\Pi_0 = \emptyset$, and*
- *$\Pi_i = \{r \in P : head(r) \in S_i \setminus S_{i-1}\}$ for $i > 0$.*

We call Π_i a component *of the program.*

Definition 5 (Solution)**.** *Let P be a logic program, $\Pi = \langle \Pi_0, \ldots \Pi_n \rangle$ be a splitting of P. A solution to Π is a sequence $\langle X_0, X_1, \ldots, X_n \rangle$ of sets of literals such that:*

- *$X_0 = \emptyset$,*
- *X_i is an answer set of $\Pi_i \cup X_{i-1}$ for $0 < i \leq n$, and*
- *X_n is consistent.*

We call X_i an answer set of the component Π_i.

Note that in our formulation of a solution, $X_i \subseteq X_{i+1}$.

Theorem 1 (Splitting sequence theorem)**.** *Let $\Pi = \langle \Pi_0, \ldots, \Pi_n \rangle$ be a splitting of a logic program P. A set A of literals is a consistent answer set of P iff $A = X_n$ for some solution $\langle X_0, \ldots, X_n \rangle$ to Π.*

Restriction to generating rules

There are the two problems with straightforward use of splitting.

The first problem is that not every answer set of a component is a subset of an answer set of the whole program.

Some candidates to an answer sets are rejected, e.g. by an integrity constraint. If preference handling semantics is unaware of this, then all the preferred answer set candidates can be rejected. Example 7 shows this situation.

Example 7. *Consider the following program*

$r_1 : a \leftarrow not\ b \qquad r_1 < r_2$
$r_2 : b \leftarrow not\ a$

$r_3 : inc \leftarrow b, not\ inc$

We split the program into two components: $\Pi_1 = \{r_1, r_2\}$ *and* $\Pi_2 = \{r_3\}$. $B = \{b\}$ *is an answer set of* Π_1. *However, using* B, $\Pi_2 \cup B$ *has no answer set as the constraint* r_3 *is not satisfied. Hence* B *is not an answer set of the whole program.*

If we choose B *to be the only preferred answer set of* Π_1 *the program has no preferred answer set.*

We want to avoid this kind of behaviour. We want to ensure that there is a preferred answer set whenever a standard one exists.

To fix the first problem we must select a preferred answer set of a component only from the answer sets of the component that are not rejected, i.e. they are subsets of some answer sets of a program.

Example 8. *In Example 7, we will choose a preferred answer set of the component* Π_1 *only from* $\{a\}$ *since* $\{b\}$ *is not a subset of any answer set of the program.*

The second problem is that one can always construct a bad rule that causes a program to have the only one component. In this way, independent rules can be forced to be in the same component. This is caused by the fact that splitting into components works on syntactic level. Example 9 shows this.

Example 9. *Consider the program from Example 6. We split it into two components:* $\Pi_1 = \{r_1, r_2\}$ *and* $\Pi_2 = \{r_3, r_4\}$.

Consider we add the rule $r_5 : a \leftarrow x, c$. *Now, the program has the unique component* $\Pi_1' = \{r_1, r_2, r_3, r_4, r_5\}$.

We can fix this problem by restricting to generating rules. It holds that $\mathcal{AS}(P) \subseteq \mathcal{AS}(\mathcal{AGR}(P))$, but there are programs where $\mathcal{AS}(\mathcal{AGR}(P)) \not\subseteq \mathcal{AS}(P)$. Therefore we need to check that an answer set of a component of $\mathcal{AGR}(P)$ is not rejected in P, i.e. it has a superset in $\mathcal{AS}(P)$. Note that the same check is needed to fix the first problem.

Following definitions formally define aforementioned check.

Definition 6 (Accepted set)**.** *Let* X *be a set of literals, and* O *be a set of sets of literals.* X *is* accepted *with respect to* O *iff there is* $B \in O$ *such that* $X \subseteq B$.

Definition 7 (Accepted solution)**.** *Let* P *be a logic program,* $O \subseteq \mathcal{AS}(P)$, Π *be a splitting of* P, *and* $X = \langle X_i, \ldots, X_n \rangle$ *be a solution to* Π.

X *is* accepted *with respect to* O *iff* X_i *is accepted with respect to* O *for all* $0 \leq i \leq n$.

The following propositions formally state that we will get the answer sets of P by using splitting to $\mathcal{AGR}(P)$ and checking whether the answer sets of the components are accepted with respect to $\mathcal{AS}(P)$. This manifests how to focus to the generating rules of a program.

Proposition 1. *Let* P *be a logic program,* $O \subseteq \mathcal{AS}(P)$, Π *be a splitting of* P, *and* $X = \langle X_0, \ldots, X_n \rangle$ *be a solution to* Π.

X *is accepted with respect to* O *iff* $X_n \in O$.

Proof. $\Rightarrow$ X_i is accepted with respect to O. Hence $A = X_n \subseteq B$ for some $B \in O$. From splitting sequence theorem we have that $A \in \mathcal{AS}(P)$. Also $B \in \mathcal{AS}(P)$. Therefore $A \subseteq B$ implies $A = B$.

$\Leftarrow$ We have $X_n = B$ for some $B \in O$. Therefore $X_i \subseteq B$ as $X_i \subseteq X_n$. X_i is accepted with respect to O. Hence X is accepted with respect to O. □

Proposition 2. *Let* P *be a logic program,* Π *be a splitting of the program* $\mathcal{AGR}(P)$, *and* A *be a consistent set of literals.*

A *is an answer set of* P *iff there is a solution* $X = \langle X_0, \ldots, X_n \rangle$ *to* Π *such that* $X_n = A$, *and* X *is accepted with respect to* $\mathcal{AS}(P)$.

Proof. $\Rightarrow$ $A \in \mathcal{AS}(P)$. Therefore $A \in \mathcal{AS}(\mathcal{AGR}(P))$. From splitting sequence theorem we have that there is a solution $X = \langle X_0, \ldots, X_n \rangle$ to Π such that $X_n = A$. Since $A \in \mathcal{AS}(P)$ we have that X is accepted with respect to $\mathcal{AS}(P)$.

$\Leftarrow$ X is accepted with respect to $\mathcal{AS}(P)$. From Proposition 1 follows that $X_n \in \mathcal{AS}(P)$. Hence we have that $A \in \mathcal{AS}(P)$. □

Whenever there is an answer set of a component that is accepted with respect to a subset of $\mathcal{AS}(P)$, then there is also an answer set of the next component that is accepted with respect to the subset. In other words, an already selected answer set of a component is not rejected in any later component.

Proposition 3. *Let* P *be a consistent logic program,* $O \subseteq \mathcal{AS}(P)$. *Let* $\Pi = \langle \Pi_0, \ldots, \Pi_n \rangle$ *be a splitting of* P.

If there are $X_0, X_1, \ldots, X_k$, *for* $0 \leq k < n$ *such that*

- $X_0 = \emptyset$,
- X_i *is an answer set of* $\Pi_i \cup X_{i-1}$, *for* $0 < i \leq k$, *and*
- $X_k \subseteq A$ *for some answer set* $A \in O$

then there is X_{k+1} *such that*

- X_{k+1} *is an answer set of* $\Pi_{k+1} \cup X_k$, *and*
- $X_{k+1} \subseteq B$ *for some answer set* $B \in O$.

Proof. Since A is a consistent answer set of P, there is a solution $Y = \langle Y_0, \ldots, Y_n \rangle$ to Π. Let $\langle S_0, \ldots, S_n \rangle$ be a sequence of splitting sets associated with Π.

Since $X_k \subseteq A$, then $X_k \cap S_j \subseteq A \cap S_j$ for all $0 \leq j \leq k$. $A \cap S_j = Y_j$, and $X_k \cap S_j = X_j$. Then $X_j \subseteq Y_j$.

Now we prove by induction that $X_i = Y_i$ for all $0 \leq i \leq k$.

We have that $X_0 = \emptyset = Y_0$. Assume $X_i = Y_i$ for all $0 \leq i < j$ where $0 < j \leq k$. We have that both X_j and Y_j are answer sets of $\Pi_j \cup X_{j-1}$. Since one answer set cannot be a proper subset of a second answer set of the same program, $X_j = Y_j$.

Now we simply take $X_{k+1} = Y_{k+1}$, and $B = A$. The second condition for X_{k+1} is satisfied because $Y_i \subseteq A$. $\square$

Selection of Preferred Answer Sets

After splitting a program into components, we select the preferred answer sets of the components.

Recall Definition 5. In each step of the construction of a solution to a splitting we have to compute an answer set of the component Π_i. If Π_i has the multiple answer sets we understand this as a decision point in which we select an answer set from all acceptable answer sets of Π_i. We understand the answer set selection as the comparison of the answer sets. Moreover, since the answer sets are generated by rules, we understand the comparison of answer sets as a comparison between generating rules.

Comparison of Generating Rules

In the first step, we transfer a preference relation on rules to a preference relation on sets of rules (generating sets). We start by defining a generating set of an answer set of a component. Every rule in a generating set must be applicable in an answer set it generates, and the generating set must generate the whole answer set.

Definition 8 (Generating set of an answer set of a component)**.** *Let Π be a component of a program P, X a set of literals, and A be an answer set of $\Pi \cup X$.*

A generating set *of A with respect to (Π, X) is a set $R \subseteq \Pi$ of rules such that:*

- *for every $r \in R$ it holds that $body^+(r) \subseteq A$ and $body^-(r) \cap A = \emptyset$,*
- *A is an answer set of $R \cup X$.*

The set of all generating sets of an answer set A with respect to (Π, X) is denoted by $\mathcal{GS}_{\Pi,X}(A)$.

When comparing two generating sets, we do not use all preferences. Only preferences between the conflicting rules are used. Our intuition is that preferences are used in order to help to resolve conflicts between rules, and therefore the preferences on non-conflicting rules should have no effect on the program.

We consider two rules to be conflicting if they are part of the sequences of rules that block each other. The next example illustrates the intuition behind this notion, and the next definition formalizes it.

Example 10. *Consider the program (a simplified version of a program from (Brewka and Eiter 1999))*

$r_1 : buy(a) \leftarrow safe(a), not\ \neg buy(a)$
$r_2 : buy(b) \leftarrow nice(b), not\ \neg buy(b)$
$r_3 : \neg buy(a) \leftarrow buy(b)$
$r_4 : \neg buy(b) \leftarrow buy(a)$
$r_5 : buy(c) \leftarrow buy(a)$

It has two answer sets, one generated by the rules $R_1 = \{r_1, r_4, r_5\}$, and the second by the rules $R_2 = \{r_2, r_3\}$. The rule r_4 depends on the rule r_1 and blocks the rule r_2. The rule r_3 depends on r_2 and blocks r_1. The rule r_5 blocks no rule. Hence there are only four conflicts, between the rules (i) r_1, r_2, (ii) r_1, r_3, (iii) r_4, r_2, (iv) r_4, r_3.

Definition 9. *Let R, P be finite sets of rules. Let $\langle r_0, r_1, \ldots, r_n \rangle$, $\langle p_0, p_1, \ldots, p_m \rangle$ be sequences of rules such that:*

- *$r_i \in R$,*
- *$p_i \in P$,*
- *$head(r_i) \in body^+(r_{i+1})$ for $0 \leq i < n$,*
- *$head(p_i) \in body^+(p_{i+1})$ for $0 \leq i < m$,*
- *$head(r_n) \in body^-(p_0)$, and*
- *$head(p_m) \in body^-(r_0)$.*

Then r_i and p_i are conflicting in (R, P).

The next definition transfers preferences on rules to preferences on set of rules. Only preferences on conflicting rules are used. We also allow preferences to defeat each other. We motivate the definition with the following example.

Example 11. *Consider the rules of a simple ski recommender*

$r_1 : \neg rec(slope) \leftarrow no\_snow(slope), not\ rec(slope)$
$r_2 : rec(slope) \leftarrow user\_likes(slope), not\ \neg rec(slope)$
$r_3 : \neg rec(slope) \leftarrow difficult(slope), not\ rec(slope)$
$r_4 : no\_show(slope) \leftarrow$
$r_5 : user\_likes(slope) \leftarrow$
$r_6 : difficult(slope) \leftarrow$ $\qquad r_3 < r_2 < r_1$

The rule r_3 represents a general requirement of a user. He/she does not like difficult slopes. However, the rule r_2 express that the system should recommend a slope when a user likes it. Since information in the rule r_2 is more specific than in the rule r_3, r_2 is preferred over r_3. The rule r_1 on the other hand operates with the most important information. It makes no sense to recommend a slope with no snow even if a user likes the slope. Hence the rule r_1 is preferred over r_2.

The program has two answer sets A_1, A_2 such that $rec(slope) \in A_1$, and $\neg rec(slope) \in A_2$. A_1 is generated by the rules $R_1 = R \cup \{r_2\}$ and A_2 is generated by the rules $R_2 = \cup\{r_1, r_3\}$ where $R = \{r_4, r_5, r_6\}$. We prefer R_2 over R_1 even though $r_2 \in R_1$ is preferred over $r_3 \in R_2$. The reason is that $r_1 \in R_2$ is even more preferred than r_2.

Definition 10 (Preference relation on sets of rules)**.** *Let P be a logic program and $<$ be a preference relation on rules.* Preference relation $\leq \subseteq 2^P \times 2^P$ *on sets of rules is defined as follows. $A \leq B$ iff*

- *there are $r_A \in A$, $r_B \in B$ such that r_A and r_B are conflicting in (A, B) and $r_A < r_B$, and*
- *there is no $r_C \in B$ such that r_B and r_C are conflicting in (B, A) and $r_B < r_C$.*

Preferred Answer Sets of a Component

Now, we are ready to define the selection of the preferred answer sets of a component. A preference relation on the answer sets of a component is constructed via comparison of the generating rules. Then, the "maximal" answer sets are selected as preferred.

Our definition is based on the view that one can choose one particular generating set to generate an answer set and

view the others generating sets as inactive. If there is a generating set of an answer set that can be selected without violating any preferences then also an answer set can be selected without violating any preferences (by using exactly the rules from that generating set).

Definition 11 (Preference relation on answer sets of a component)**.** *Let P be a logic program, and $<$ be a preference relation on P. Let Π be a component of the program P. Let X be a set of literals. Let Q be the set of all answer sets of $\Pi \cup X$.*

Then preference relation *on the answer sets of the component Π for $<$, denoted $\leq_{\Pi,X} \subseteq Q \times Q$ is defined as follows:*

- $A \leq_{\Pi,X} A$,
- $A \leq_{\Pi,X} B$ *if for each* $R_A \in \mathcal{GS}_{\Pi,X}(A)$ *there is* $R_B \in \mathcal{GS}_{\Pi,X}(B)$ *such that* $R_A \leq R_B$*, and*
- *if* $A \leq_{\Pi,X} B$ *and* $B \leq_{\Pi,X} C$*, then* $A \leq_{\Pi,X} C$.

Definition 12 (Preferred answer set of a component)**.** *Let P a logic program , and $<$ be a preference relation on P. Let Π be a component of the program P. Let X be a set of literals. Let Q be the set of all answer sets of $\Pi \cup X$, and $O \subseteq Q$.*

$A \in O$ is preferred among *O with respect to $<$ iff for every $B \in O$ if $A \leq_{\Pi,X} B$ then $B \leq_{\Pi,X} A$.*

Proposition 4. *Let P be a logic program, and $<$ be a preference relation on P. Let Π be a component of the program P. Let X be a set of literals. Let Q be the set of all answer sets of $\Pi \cup X$, and $\emptyset \subset O \subseteq Q$.*

Then there is a preferred answer set A among O with respect to $<$.

Proof. Assume there is no preferred answer set among O with respect to $<$.

We can create sequence $A_1, A_2, \ldots, A_{n+1}$ where:

- $n = |O|$,
- $A_i \leq_{\Pi,X} A_{i+1}$ for $1 \leq i < n$,
- $A_{i+1} \not\leq_{\Pi,X} A_i$ for $1 \leq i < n$.

Since there is a finite number of elements in O we have that $A_{n+1} = A_i$ for some $i < n+1$. Hence $A_{n+1} \leq_{\Pi,X} A_n$. A contradiction. □

Preferred Answer Sets

Preferred answer sets of a whole program are defined via preferred solutions. A solution to a program is preferred iff an answer set of each component is preferred.

Definition 13 (Preferred solution)**.** *Let P be a logic program, $<$ be a preference relation on P, $O \subseteq \mathcal{AS}(P)$, Π be a splitting of P, and $X = \langle X_0, \ldots, X_n \rangle$ be a solution to Π.*

X is preferred *with respect to $<$ and O iff in Definition 5 for all $i > 0$ $X_i \in Q_i$, and X_i is preferred among Q_i with respect to $<$, where Q_i is the set of all those answer sets of $\Pi_i \cup X_{i-1}$, which are accepted with respect to O.*

An answer set is preferred if there is a preferred solution that generates it. A solution is always bound to a splitting. In Definition 14 we do not require a preferred solution to be bound to any particular splitting. We motivate the definition on the following example.

Example 12. *Consider the program*

$r_1 : a \leftarrow not\ b$ $\quad$ $r_3 : c \leftarrow not\ d$ $\quad$ $r_1 < r_2$
$r_2 : b \leftarrow not\ a$ $\quad$ $r_4 : d \leftarrow not\ c$ $\quad$ $r_4 < r_3$

$r_5 : inc \leftarrow b, c, not\ inc$

It has three components $\Pi_1 = \{r_1, r_2\}$, $\Pi_2 = \{r_3, r_4\}$, and $\Pi_3 = \{r_5\}$. There is no dependency between Π_1 and Π_2 so we can consider them in arbitrary order.

When we consider Π_1 first, we select $\{b\}$ as a preferred answer set of Π_1. In the next step, we cannot select $\{b, c\}$ even though r_3 is preferred. Thus $\{b, d\}$ is preferred answer set.

Considering Π_2 first, yields dual situation. $\{c, a\}$ is a preferred answer set.

Definition 14 (Preferred answer set)**.** *Let $\mathcal{P} = (P, <)$ be a logic program with preferences.*

A consistent set A of literals is a preferred answer set *of $\mathcal{P}$ iff there is a solution $X = \langle X_0, \ldots, X_n \rangle$ to some splitting Π of the program $\mathcal{AGR}(P)$ such that $X_n = A$, and X is preferred with respect to $<$ and $\mathcal{AS}(P)$.*

Lit is a preferred answer set of $\mathcal{P}$ iff Lit is an answer set of P.

Set of all preferred answer sets of $\mathcal{P}$ is denoted $\mathcal{PAS}(\mathcal{P})$.

Proposition 5. *Let P be a consistent logic program, $<$ be a preference relation on P, $\emptyset \subset O \subseteq \mathcal{AS}(P)$, and Π be a splitting of P.*

There exists a preferred solution to Π with respect to $<$ and O.

Proof. Recall Definition 13. From Proposition 3 we have that $Q_{i+1} \neq \emptyset$ for all $i \geq 0$ whatever X_i we choose. From Proposition 4 we have that there is a preferred answer set of the component Π_i among Q_i with respect to $<$ for all $i > 0$. □

Properties

In this section we present properties of preferred answer sets.

The presented approach to preference handling is selective, i.e. every preferred answer set is an answer set.

Theorem 2. *Let $(P, <)$ be a logic program with preferences.*

If A is an preferred answer set of $(P, <)$ then it is an answer set of P.

Proof. Follows from Definition 14 using Proposition 2. □

Brewka and Eiter have presented in their paper (Brewka and Eiter 1999) principles for preference handling, which try to characterize reasonable preference handling semantics. Neither Principle I nor Principle II is satisfied in our approach. Reasons are discussed below each principle. However our semantics does not satisfy principles I and II, it is able to correctly solve problematic programs from literature.

Principle I. *Let $(P, <)$ be a logic program with preferences, A_1, A_2 be two answer sets of P. Let $R \subset P$ be a set of rules and $d_1, d_2 \notin R$ are rules. Let $\mathcal{AGR}(A_1) = R \cup \{d_1\}$ and $\mathcal{AGR}(A_2) = R \cup \{d_2\}$. If d_1 is preferred over d_2 then A_2 is not a preferred answer set of $(P, <)$.*

Theorem 3. *Preferred answer sets defined in Definition 14 do not satisfy Principle I.*

Proof. Consider the following program:

$r_1 : a \leftarrow x, y, not\ b, not\ z$ $\quad r_2 < r_1$
$r_2 : b \leftarrow x, y, not\ a, not\ z$ $\quad r_5 < r_3$
$r_3 : x \leftarrow not\ z, not\ w$ $\quad r_4 < r_6$
$r_4 : y \leftarrow not\ z, not\ w$
$r_5 : z \leftarrow not\ x, not\ y, not\ a, not\ b$
$r_6 : w \leftarrow not\ x, not\ y, not\ a, not\ b$

Whole program forms the only component. It has three answer sets: $A_1 = \{x, y, a\}$ generated by $R_1 = \{r_3, r_4, r_1\}$, $A_2 = \{x, y, b\}$ generated by $R_2 = \{r_3, r_4, r_2\}$, and $A_3 = \{z, w\}$ generated by $R_3 = \{r_5, r_6\}$. We have $A_2 \leq A_1$, $A_1 \leq A_3$, and $A_3 \leq A_2$. Thus A_1, A_2, and A_3 are preferred answer sets.

$\mathcal{AGR}(A_1) = \emptyset \cup \{r_1\}$, and $\mathcal{AGR}(A_2) = \emptyset \cup \{r_2\}$. Since $r_2 < r_1$, Principle I requires that A_2 is not preferred. Principle I is thus violated. □

Principle I uses a local view. It is only concerned how the answer sets A_1 and A_2 are generated. However, a logic program can have more answer sets with contrary preferences. Such preferences cause an existence of a cycle in a preference relation on answer sets in our approach. The answer sets that form a cycle are equally preferred. Our view is that we should either select all answer sets in a cycle as preferred or do not select any of them. Since we want to always select a preferred answer set when a standard one exists, we select all the answer sets in a cycle.

Principle II. *Let A be a preferred answer set of a logic program $(P, <)$ with preferences and r be a rule such that $body^+(r) \not\subseteq A$. Then A is a preferred answer set of a logic program $(P \cup \{r\}, <')$ with preferences such that $<' \cap (P \times P) = <$.*

Theorem 4. *Preferred answer sets defined in Definition 14 do not satisfy Principle II.*

Proof. Consider the situation from (Brewka and Eiter 1999). Consider the following program:

$r_1 : c \leftarrow not\ b$ $\quad r_2 < r_1$
$r_2 : b \leftarrow not\ a$

It has the only answer set $B = \{b\}$, which is also the only preferred answer set.

Let extend the program with the rule $r_3 : a \leftarrow c$ and the preference $r_1 < r_3$. Now, it has the two answer sets $A = \{c, a\}$ and $B = \{b\}$. $body^+(r_3) \not\subseteq B$, but B is not a preferred answer set in our approach. A is the only preferred answer set. Principle II is thus violated. □

As Brewka and Eiter showed in (Brewka and Eiter 1999), Principle II and III are incompatible (under the condition that semantics handles program from previous proof as described). We favour Principle III under these circumstances.

Principle III. *Let $\mathcal{P} = (P, <)$ be a logic program with preferences. If $\mathcal{AS}(P) \neq \emptyset$ then $\mathcal{PAS}(\mathcal{P}) \neq \emptyset$.*

Theorem 5. *Preferred answer sets defined in Definition 14 satisfy Principle III.*

Proof. For a inconsistent program it follows directly from Definition 14. For a consistent program it follows from Proposition 5. □

We add the following principle. It says that preferences on non-generating rules are irrelevant.

Principle IV. *Let $\mathcal{P}_1 = (P, <)$ and $\mathcal{P}_2 = (P, <')$ be logic programs with preferences such that $< \cap D =<' \cap D$ where $D = \mathcal{AGR}(P) \times \mathcal{AGR}(P)$.*
Then $\mathcal{PAS}(\mathcal{P}_1) = \mathcal{PAS}(\mathcal{P}_2)$.

Theorem 6. *Preferred answer sets defined in Definition 14 satisfy Principle IV.*

Proof. Follows directly from Definition 14. Only the rules from $\mathcal{AGR}(P)$ are used in the definition. □

When there are no preferences on the generating rules of a program, all the answer sets are preferred.

Theorem 7. *Let $Q = (P, <)$ be a logic program with preferences such that $< \cap (\mathcal{AGR}(P) \times \mathcal{AGR}(P)) = \emptyset$.*
Then $\mathcal{PAS}(Q) = \mathcal{AS}(P)$.

Proof. For the inconsistent program it follows directly from Definition 14. Assume P is consistent. Consider Definition 12 of a preferred answer set of a component. We have that $<_{\Pi,X} = \emptyset$. Then every answer set of a component is preferred. Hence each solution to $\mathcal{AGR}(P)$ accepted with respect to $\mathcal{AS}(P)$ is preferred with respect to $<$ and $\mathcal{AS}(P)$. Finally, each answer set of P is a preferred answer set of $\mathcal{P}$. □

Adding the additional preferences to a logic program does not necessary restrict the set of all preferred answer sets. The reason is that the additional preferences can introduce a cycle to a preference relation on the answer sets, and a former non-preferred answer set becomes preferred one.

Theorem 8. *Let $\mathcal{P}_1 = (P, <)$ and $\mathcal{P}_2 = (P, <')$ be a logic programs such that $< \subseteq <'$. Then $\mathcal{PAS}(\mathcal{P}_1) \not\supseteq \mathcal{PAS}(\mathcal{P}_2)$.*

Proof. Consider the program $P_1 = (P, <)$:

$r_1 : x \leftarrow not\ b$ $\quad r_3 : y \leftarrow not\ a$ $\quad r_3 < r_1$
$r_2 : a \leftarrow x$ $\quad r_4 : b \leftarrow y$

It has two answer sets: $A_1 = \{x, a\}$ generated by $R_1 = \{r_1, r_2\}$, and $A_2 = \{y, b\}$ generated by $R_2 = \{r_3, r_4\}$. $\mathcal{PAS}(\mathcal{P}_1) = \{A_1\}$. Now, consider the program $\mathcal{P}_2 = (P, <')$ with preferences $r_3 <' r_1$, $r_2 <' r_4$. We have that $< \subseteq <'$ and $\mathcal{PAS}(\mathcal{P}_2) = \{A_1, A_2\} \not\subseteq \mathcal{PAS}(\mathcal{P}_1)$. □

Related Work

In this section we compare proposed semantics to existing semantics. There are other semantics for preference handling besides those considered here, e.g. (Brewka 2002), (Van Nieuwenborgh and Vermeir 2006), (Gabaldon 2011). Since the exhaustive comparison is beyond the scope of this paper, we focus on the most related approaches.

Brewka and Eiter; Delgrande, Schaub, and Tompits; Wang, Zhou, and Lin

Semantics (Brewka and Eiter 1999), (Delgrande, Schaub, and Tompits 2003), and (Wang, Zhou, and Lin 2000) are defined for the same underlying language as we do: extended logic programs with preferences on rules encoded as a strict order on rules.

Schaub and Wang have shown in (Schaub and Wang 2003) that the three semantics can be characterized in a uniform way. One of the characterization is via order preserving enumeration of generating rules. To illustrate the characterization we recapitulate the definition of preferred answer sets according to (Delgrande, Schaub, and Tompits 2003).

Definition 15. *Let $\mathcal{P} = (P, <)$ be a logic program with preferences and let X be an answer set of P.*

Then X is a preferred answer set of $\mathcal{P}$, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of $\mathcal{AGR}_P(X)$ such that for every $i, j \in I$ we have that:

1. $body^+(r_i) \subseteq \{head(r_j) : j < i\}$;
2. *if* $r_i < r_j$, *then* $j < i$; *and*
3. *if* $r_i < r'$ *and* $r' \in P \setminus \mathcal{AGR}_P(X)$, *then*
   (a) $body^+(r') \subseteq X$ *or*
   (b) $body^-(r') \cap \{head(r_j) : j < i\} \neq \emptyset$.

Definitions of preferred answer sets according to (Brewka and Eiter 1999) and (Wang, Zhou, and Lin 2000) differ in the condition 1 and 3(b). All the approaches use the conditions 2 and 3(a).

The second condition expresses the prescriptive (Delgrande et al. 2004) nature of the approaches. They understand the preferences as an order, in which the rules must be used. If the rules that generate an answer set of a program cannot be used in this order then the answer set is not preferred (e.g. Example 2). This is the main difference between our approaches. We apply the rules of a program in a natural order. Dependencies between rules and answer set semantics induce this natural order. We use preferences only to select between the rules that are responsible for multiple answer sets of a program.

The second difference is how broad view an approach uses. (Brewka and Eiter 1999), (Delgrande, Schaub, and Tompits 2003), and (Wang, Zhou, and Lin 2000) use a local view. When testing whether an answer set is preferred, other answer sets are not used. However, we use a global view. We compare the answer sets, and therefore we need all the answer sets to decide whether an answer set is preferred.

The approaches of preferred answer sets defined in (Brewka and Eiter 1999) and (Delgrande, Schaub, and Tompits 2003) satisfy Principle I and II. However, Principles III and IV are not satisfied in any of the three approaches.

Brewka and Eiter – Weakly Preferred Answer Sets

Approach of weakly preferred answer sets (Brewka and Eiter 1999) is a relaxation of the approach of preferred answer sets from (Brewka and Eiter 1999). The preferred answer sets of a program are used whenever they exist. If the program has no preferred answer set, the weakly preferred answer sets are used. An answer set is weakly preferred if the minimal number of changes to a preference relation is needed in order for the answer set to pass the preferred answer set test. The approach adopts a technical point of view. Consequently, it is less concerned with the properties of defined semantics. It satisfies only Principle III. It also does not consider preferences to be immutable. It changes preference information in order to ensure existence of a weakly preferred answer set. All preferences are treated alike, and have the same importance; both relevant and irrelevant ones. On the other hand, we detect which preferences are irrelevant in our approach.

Example 13. *Consider the program from the proof of Proposition 6.6 from (Brewka and Eiter 1999):*

$r_1 : a \leftarrow not\ \neg a$ $\quad$ $r_5 : \neg a \leftarrow not\ b, a$
$r_2 : \neg a \leftarrow not\ a$ $\quad$ $r_4 : c \leftarrow not\ c$
$r_3 : \neg a \leftarrow not\ c, a$ $\quad$ $r_6 : b \leftarrow not\ \neg b$

$r_6 < r_5 < r_4 < r_3 < r_2 < r_1$

The program has two answer sets, $A_1 = \{a, b, c\}$ generated by $R_1 = \{r_1, r_4, r_6\}$, and $A_2 = \{\neg a, b, c\}$ generated by $R_2 = \{r_2, r_4, r_6\}$. From Principle I it follows that A_2 should not be preferred.

However, after changing preference $r_2 < r_1$ to $r_1 < r_2$, answer set A_2 becomes preferred one. Hence A_2 is weakly preferred answer set (Brewka and Eiter 1999).

In other worlds, Principle I states that preference $r_2 < r_1$ is important, and other preferences are irrelevant when checking whether A_2 is preferred. However, approach of weakly preferred answer sets changes the only important preference $r_2 < r_1$ and leaves all irrelevant preferences intact.

Now we show how the program is solved in our approach. The program has three components $\Pi_1 = \{r_4\}$, $\Pi_2 = \{r_6\}$, and $\Pi_3 = \{r_1, r_2, r_3, r_5\}$. $\{c\}$ is the only answer set of Π_1, and $\{b, c\}$ is the only answer set of $\Pi_2 \cup \{c\}$. $\Pi_3 \cup \{b, c\}$ has two answer sets: $B_1 = \{b, c, a\}$ generated by $R_1 = \{r_1\}$ and $B_2 = \{b, c, \neg a\}$ generated by $R_2 = \{r_2\}$. We have that $R_2 \leq R_1$ and hence B_1 is the only preferred answer set of the component Π_3. Finally A_1 is the only preferred answer set of the program according to our approach.

Zhang and Foo

Approach presented in (Zhang and Foo 1997) defines semantics for extended logic programs with preference relation which is strict order. Semantics is defined via sequence of the program reducts. Roughly speaking, having a program P, a rule $r \in P$ is removed from the program if $r < r'$ for some $r' \in P$ and it is defeated by $P' = P \setminus \{r\}$, i.e. every answer set of P' defeats r. The answer sets of a reduced program are the preferred answer sets of the original program. The approach satisfies Principle III (Brewka and Eiter 1999). There are situations in which the approach fails to detect that the rules with preferences are not in conflict.

Example 14. *Consider the program from Example 4*

According to semantics of (Zhang and Foo 1997) r_1 is less preferred that r_3 and is defeated by $\{r_2, r_3\}$:

- *$\{r_2, r_3\}$ has the unique answer set $\{b, c\}$, and*

- $body^{-}(r_1) \cap \{b, c\} \neq \emptyset$.

Hence the preferred answer sets of the original program are exactly the answer sets of $\{r_2, r_3\}$*, i.e.* $\{b, c\}$ *is the only preferred answer set of the program. The approach is unable to detect that* r_3 *is preferred over* r_1 *but the rule* r_2 *defeats* r_1.

In our approach the program is split into two components $\Pi_1 = \{r_1, r_2\}$*, and* $\Pi_2 = \{r_3\}$*. Since there are no preferences between the rules in* Π_1*, both* $\{a\}$ *and* $\{b, c\}$ *are the preferred answer sets of the program.*

Sakama and Inoue

(Sakama and Inoue 2000) defines semantics for general extended disjunctive programs with preferences on literals. It is defined via comparison of answer sets. However (Sakama and Inoue 2000) also gives the recipe how to transform preferences on rules to preferences on literals. Information about structure of rules and dependencies between rules is lost during the transformation. Consequently all preferences, including irrelevant ones, are used to select the preferred answer sets of a program. For example, in Example 4 the transformation is unable to ignore the preference $r_1 < r_3$.

Šefránek and Šimko

We want to note that the approach presented in (Šefránek and Šimko 2011) and the one presented in this paper are two different approaches to preference handling. The former represents a refinement of (Šefránek 2008). Preference handling is viewed as a kind of argumentation.

Both the approach from (Šefránek and Šimko 2011) and the one presented in this paper were designed with the Principle III in mind. Otherwise, they have different designs. For now, we can say that some of the preferred stable models according to (Šefránek and Šimko 2011) are not preferred in this approach. However, we have not studied the connection between the two approaches in depth. It is one of the goals for our future research.

Conclusion

In this paper we have proposed semantics for extended logic programs with preferences on rules. Different semantics have been already proposed in the literature. However, they are not always able to ignore certain unnatural preferences, and lead to counter-intuitive conclusions. Approaches of (Brewka and Eiter 1999; Delgrande, Schaub, and Tompits 2003; Wang, Zhou, and Lin 2000; Zhang and Foo 1997; Sakama and Inoue 2000) have difficulties in the situations: (i) when the preferences are not compatible with natural order, in which rules have to be applied, (ii) when the preferences are specified on rules that are not applicable under answer set semantics, (iii) when the preferences are specified on non-conflicting rules.

On the other hand, approach of (Šefránek 2008) is able to ignore preferences on non-applicable rules. However, it is too restrictive, and ignores some preferences between conflicting rules.

The main contribution of the paper is proposed semantics that addresses aforementioned situations. The approach is based on the following notions: (i) comparison of generating rules, (ii) detection which rules are conflicting, and (iii) splitting a program into components, in which the comparisons are performed.

Acknowledgements

This paper was supported by the grants 1/0689/10 and 1/1333/12 of VEGA.

References

Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence* 109(1-2):297–356.

Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of AAAI-02*, 100–105. AAAI Press.

Delgrande, J.; Schaub, T.; Tompits, H.; and Wang, K. 2004. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence* 20(2):308–334.

Delgrande, J.; Schaub, T.; and Tompits, H. 2003. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming* 3(2):129–187.

Gabaldon, A. 2011. A selective semantics for logic programs with preferences. In *Proceedings of 9th Int'l Workshop on Nonmonotonic Reasoning, Action and Change*.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Principles of Knowledge Representation*, 23–37. MIT Press.

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123(1-2):185–222.

Schaub, T., and Wang, K. 2003. A semantic framework for preference handling in answer set programming. *Theory and Practice of Logic Programming* 3(4):39.

Van Nieuwenborgh, D., and Vermeir, D. 2006. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming* 6(1-2):107–167.

Šefránek, J., and Šimko, A. 2011. Warranted derivations of preferred answer. In *19th International Conference on Applications of Declarative Programming and Knowledge Management and 25th Workshop on Logic Programming.*, 195–207.

Šefránek, J. 2008. Preferred answer sets supported by arguments. In *Proc. of the Workshop NMR*.

Wang, K.; Zhou, L.; and Lin, F. 2000. Alternating fixpoint theory for logic programs with priority. In *Computational Logic*, 164–178.

Zhang, Y., and Foo, N. Y. 1997. Answer sets for prioritized logic programs. In *ILPS*, 69–83.