# Strong Equivalence for Logic Programs with Preferences*

**Wolfgang Faber**†
Department of Mathematics
University of Calabria
I-87030 Rende (CS), Italy
Email: faber@mat.unical.it

**Kathrin Konczak**
Institut für Informatik
Universität Potsdam
PF 900327, D-14439 Potsdam, Germany
Email: konczak@cs.uni-potsdam.de

## Abstract

Recently, strong equivalence for Answer Set Programming has been studied intensively, and was shown to be beneficial for modular programming and automated optimization. In this paper we define the novel notion of strong equivalence for logic programs with preferences. Based on this definition we give, for several semantics for preference handling, necessary and sufficient conditions for programs to be strongly equivalent. These results provide a clear picture of the relationship of these semantics with respect to strong equivalence, which differs considerably from their relationship with respect to answer sets. Finally, based on these results, we present for the first time simplification methods for logic programs with preferences.

## 1 Introduction

During the last decade, Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988] has become an increasingly acknowledged tool for knowledge representation and reasoning. A main advantage of ASP is that it is based on solid theoretical foundations, while being able to model commonsense reasoning in an arguably satisfactory way. The availability of efficient solvers has furthermore stimulated its use in practical applications in recent years. This development had quite some implications on ASP research. For example, increasingly large applications require features for modular programming. Another requirement is the fact that in applications, ASP code is often generated automatically by so-called frontends, calling for optimization methods which remove redundancies, as also found in database query optimizers. For these purposes the rather recently suggested notion of strong equivalence for ASP [Lifschitz *et al.*, 2001; Turner, 2003] can be used. Indeed, if two ASP programs are strongly equivalent, they can be used interchangeably in any context. This gives a handle on showing the equivalence of ASP modules. If a program is strongly equivalent to a subprogram of itself, then one can always use the subprogram instead of the original program, a technique which serves as an effective optimization method.

On a different line of ASP research, many extensions of the basic formalism have been proposed. Perhaps the most intensively studied one is the modeling of preferences in ASP, cf. [Delgrande *et al.*, 2004]. Strongly rooted in the research of nonmonotonic formalisms, the ability to specify preferences is acknowledged to be particularly beneficial to ASP, since they constitute a very natural and effective way of resolving indeterminate solutions. For example, preferences have been successfully used for timetabling, auctioning, configuration, and other domains. A sophisticated application for information site selection has been described in [Eiter *et al.*, 2002]. The emergence of such applications thus also calls for optimization methods, as for standard ASP.

In this paper we tackle this issue and generalize the notion of strong equivalence to ASP with preferences. Since a plethora of formalisms and semantics has been introduced for extending ASP by preferences, we have to limit ourselves in this paper to a few concrete approaches. A basic classification of preference handling is (i) over which elements the preference is defined and (ii) the way how preferences are interpreted, and in particular how they are aggregated. In this work we focus on ordered programs, where preferences are defined among rules, which semantically act as filters over answer sets. In particular, we consider the three semantics defined in [Delgrande *et al.*, 2003; Wang *et al.*, 2000; Brewka and Eiter, 1999], referred to as $D$-, $W$-, and $B$-semantics, respectively. The reason for this choice is that these semantics seem to be widely accepted and their properties and interrelationships are fairly well-understood.

The contributions of this work are as follows. We define the notion of strong equivalence for ordered programs. It turns out that the definition for standard ASP cannot be straightforwardly extended for two main reasons: (1) the "union" of ordered programs needs to be defined properly, and (2) not any "union" of ordered programs yields a valid ordered program. To overcome these problems, we define the notion of *admissible extension*, which provides a general method for defining strong equivalence for arbitrary structures. We show several properties of strong order equivalence, providing necessary

*The first author was partially supported by the EC under project IST-2002-33570 (INFOMIX). The second author was partially supported by the German Science Foundation (DFG) under grant SCHA 550/6-4, TP C. Both authors were partially supported by the EC under project IST-2001-37004 WASP.

†Funded by APART of the Austrian Academy of Sciences.

and sufficient conditions for programs to be strongly order equivalent. We also explore the relationship of strong order equivalence between the three semantics studied in this paper, and the relationship to strong equivalence of standard logic programs. Based on these results, we study the applicability of program simplification rules, known from standard ASP, in the presence of preferences. We obtain that most of these simplifications can be applied only in particular contexts. The main results of this work are that the relationship between the three preference semantics is changing under strong equivalence and that preferences cannot be "compiled away" in a modular fashion. We omit all proofs due to the page limit. [1]

## 2 Background

**Answer Sets.** A *logic program* (LP) is a finite set of rules such as $p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n$, where $n \geq m \geq 0$, and each $p_i$ $(0 \leq i \leq n)$ is an *atom*. For such a rule $r$, we let $head(r)$ denote the *head*, $p_0$, of $r$ and $body(r)$ the *body*, $\{p_1, \ldots, p_m,\ not\ p_{m+1}, \ldots, not\ p_n\}$, of $r$. Let $body^+(r) = \{p_1, \ldots, p_m\}$ and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$. For a set of rules $\Pi$, we write $head(\Pi) = \{head(r) \mid r \in \Pi\}$. A program is *basic* if $body^-(r) = \emptyset$ for all its rules. The *reduct*, $\Pi^X$, of a program $\Pi$ *relative to* a set $X$ of atoms is defined by $\Pi^X = \{head(r) \leftarrow body^+(r) \mid r \in \Pi, body^-(r) \cap X = \emptyset\}$. A set of atoms $X$ is *closed under* a basic program $\Pi$ if for any $r \in \Pi$, $head(r) \in X$ if $body^+(r) \subseteq X$. The smallest set of atoms being closed under a basic program $\Pi$ is denoted by $Cn(\Pi)$. Then, a set $X$ of atoms is an *answer set* of a program $\Pi$ if $Cn(\Pi^X) = X$. We use $AS(\Pi)$ for denoting the set of all answer sets of $\Pi$. In what follows, an important concept is that of the *generating rules* of an answer set. Given a set $X$ of atoms from $\Pi$, define $R_\Pi(X) = \{r \in \Pi \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset\}$.

**Strong Equivalence.** Under the answer set semantics, two LPs $\Pi_1$ and $\Pi_2$ are regarded as equivalent, denoted $\Pi_1 \equiv \Pi_2$, iff $AS(\Pi_1) = AS(\Pi_2)$. Two LPs $\Pi_1$ and $\Pi_2$ are *strongly equivalent*, denoted $\Pi_1 \equiv_s \Pi_2$, iff, for any LP $\Pi$, the programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent, i.e. $\Pi_1 \cup \Pi \equiv \Pi_2 \cup \Pi$. As shown in [Lifschitz *et al.*, 2001], strong equivalence is closely related to the non-classical logic of here-and-there, which was adapted to logic-programming terms by Turner [Turner, 2003]: Let $\Pi$ be an LP, and let $X, Y \subseteq A$ such that $X \subseteq Y$. The pair $(X, Y)$ is an *SE-model* of $\Pi$ (over $A$), if $Y \models \Pi$ and $X \models \Pi^Y$. By $SE^A(\Pi)$ we denote the set of all SE-models over $A$ of $\Pi$. Usually the set $A$ is left implicit and one just writes $SE(\Pi)$. Then, for any LPs $\Pi_1$ and $\Pi_2$, $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$ [Turner, 2003].

**Preferences.** A logic program $\Pi$ is said to be *ordered* if we have a set $\mathcal{N}$ of terms serving as *names* for rules and a *preference relation* $< \subseteq \mathcal{N} \times \mathcal{N}$ as a strict partial order among rules, where we write $s < t$ for the names $s, t \in \mathcal{N}$ of rules in $\Pi$. Furthermore, we assume a bijective function $n(.)$ assigning to each each rule $r \in \Pi$ a name $n(r) \in \mathcal{N}$. To simplify our notation, we usually write $r_i$ instead of $n(r_i)$ for some $r_i \in \Pi$. Given $r_i, r_j \in \Pi$, $r_i < r_j$ states that $r_j$ has higher priority than $r_i$. Formally, an ordered (logic) program (OLP) can be understood as a quadruple $(\Pi, \mathcal{N}, n, <)$, where $\Pi$ is a logic program, $n$ is a bijective function between $\Pi$ and the set of names $\mathcal{N}$, and $<$ is a set of preference relations over $\Pi$. Whenever possible, we leave $\mathcal{N}$ and $n$ implicit and just write $(\Pi, <)$ for OLPs. Moreover, we write $<= \emptyset$ if no partial order is given that is $(\Pi, \emptyset)$ denotes a logic program. The interpretation that one rule has higher priority than an other rule can be made precise in different ways. In what follows, we consider three such interpretations: $D$– [Delgrande *et al.*, 2003], $B$– [Brewka and Eiter, 1999], and $W$– preference [Schaub and Wang, 2003]. Given $(\Pi, <)$, all of them use $<$ for selecting preferred answer sets among the standard answer sets of $\Pi$. We now recall the definitions of these semantics. [2]

**Definition 2.1** *Let $(\Pi, <)$ be an OLP and $X$ be an answer set of $\Pi$. Then, $X$ is $<^D$–preserving, if an enumeration $\langle r_i \rangle_{i \in I}$ of $\Pi$ exists s.t. for every $i, j \in I$ we have*

**1.** *if $r_i < r_j$, then $j < i$, and*

**2.** *if $r_i \in R_\Pi(X)$ then $body^+(r_i) \subseteq \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$, and*

**3.** *if $r_i \in \Pi \setminus R_\Pi(X)$ then*

**(a)** *$body^+(r_i) \not\subseteq X$ or*

**(b)** *$body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$.*

Cond.1 stipulates that the enumeration of $\Pi$ is compatible with $<$. Cond.2 makes the property of supportedness explicit. Although any answer set is generated by a supported sequence of rules, in $D$–preferences, rules cannot be supported by lower-ranked ones. Cond.3a separates the handling of unsupported rules from preference handling. Cond.3b guarantees that rules can never be blocked by lower-ranked ones.

**Definition 2.2** *Let $(\Pi, <)$ be an OLP and $X$ be an answer set of $\Pi$. Then, $X$ is $<^W$–preserving, if an enumeration $\langle r_i \rangle_{i \in I}$ of $\Pi$ exists s.t. for every $i, j \in I$ we have*

**1.** *if $r_i < r_j$, then $j < i$, and*

**2.** *if $r_i \in R_\Pi(X)$ then*

**(a)** *$body^+(r_i) \subseteq \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$ or*

**(b)** *$head(r_i) \in \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$, and*

**3.** *if $r_i \in \Pi \setminus R_\Pi(X)$ then*

**(a)** *$body^+(r_i) \not\subseteq X$ or*

**(b)** *$body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$ or*

**(c)** *$head(r_i) \in \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$.*

For $W$–preference, the concept of order preservation from $D$–preferences is weakened in Condition 2 and 3 for suspending both conditions, whenever the head of a preferred rule is derivable in an alternative way.

**Definition 2.3** *Let $(\Pi, <)$ be an OLP and $X$ be an answer set of $\Pi$. Then, $X$ is $<^B$–preserving, if an enumeration $\langle r_i \rangle_{i \in I}$ of $\Pi$ exists s.t. for every $i, j \in I$ we have*

**1.** *if $r_i < r_j$, then $j < i$, and*

[1] A version of the paper including proofs can be found at `http://www.wfaber.com/research/#publications`

[2] In the following, $I$ is an index set $I = \{1, \ldots, n\}$ for $|\Pi| = n$.

**2.** *if* $r_i \in \Pi \setminus R_\Pi(X)$ *then*

**(a)** $body^+(r_i) \not\subseteq X$ *or*

**(b)** $body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$ *or*

**(c)** $head(r_i) \in X$.

$B$–preference drops Cond.2; thus decoupling preference handling from the order induced by consecutive rule applications.

For $\sigma \in \{D, W, B\}$, we define $AS^\sigma((\Pi, <))$ as the set of all $<^\sigma$-preserving answer sets of OLP $(\Pi, <)$. As shown in [Schaub and Wang, 2003], the three strategies yield an increasing number of preferred answer sets. That is, $D$–preference is stronger than $W$–preference, which is stronger than $B$–preference, which is stronger than no preference. More precisely, we have $AS^D((\Pi, <)) \subseteq AS^W((\Pi, <)) \subseteq AS^B((\Pi, <)) \subseteq AS(\Pi)$ and, in addition, $AS^\sigma((\Pi, \emptyset)) = AS(\Pi)$ for $\sigma \in \{D, W, B\}$.

## 3 Strong Order Equivalence

In this section we define variants of equivalences for OLPs. Defining "standard" equivalence (two programs permit precisely the same answer sets) is straightforward by considering the respective preferred answer sets.

However, concerning the notion of strong equivalence, there are some caveats. For standard programs, strong equivalence for $\Pi_1$ and $\Pi_2$ holds if for any program $\Pi$, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent. If we now take preferences into account, for any $(\Pi, <)$, $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ would be strongly equivalent if $(\Pi_1 \cup \Pi, <_1 \cup <)$ and $(\Pi_2 \cup \Pi, <_2 \cup <)$ are equivalent. The problem is that $(\Pi_1 \cup \Pi, <_1 \cup <)$ is not necessarily an OLP, because of two issues: 1. Rule label clashes, and 2. partial order clashes.

Rule label clashes, i.e. two different rules have the same name or one rule has two different names, can be resolved by a *renaming*. We say that $(\Pi, \mathcal{N}^*, n^*, <^*)$ is a *renaming* of $(\Pi, \mathcal{N}, n, <)$ if $\mathcal{N}^*$ is a new set of names of rules, $n^*(.)$ is a bijective function assigning each rule of $\Pi$ an unique name of $\mathcal{N}^*$, and $n^*(r) <^* n^*(r')$ holds iff $n(r) < n(r')$, for all $r, r' \in \Pi$. The union of two OLPs is then defined as follows.

**Definition 3.1** *Let* $(\Pi, \mathcal{N}, n, <)$ *and* $(\Pi', \mathcal{N}', n', <')$ *be OLPs where* $\mathcal{N}, \mathcal{N}'$ *are disjoint sets of rule names.*

*We define the union of* $(\Pi, \mathcal{N}, n, <)$ *and* $(\Pi', \mathcal{N}', n', <')$ *as* $(\Pi^*, \mathcal{N}^*, n^*, <^*)$*, where*

- $\Pi^* = \Pi \cup \Pi'$
- $\mathcal{N}^*$ *as new set of names of rules,*
- $n^*(r)$ *for* $r \in \Pi \cup \Pi'$ *as a bijective function assigning each rule of* $\Pi \cup \Pi'$ *an unique name of* $\mathcal{N}^*$*, and*
- $<^*$ *as* $< \cup <'$*, where for* $r, r' \in \Pi \cup \Pi'$*,* $n^*(r) <^* n^*(r')$ *holds whenever* $n(r) < n(r')$ *or* $n'(r) <' n'(r')$ *holds.*

Again, we usually leave $\mathcal{N}^*$ and $n^*$ implicit. Sometimes we write $(\Pi \cup \Pi', < \cup <')$ for the union of $(\Pi, <)$ and $(\Pi', <')$.

We next observe that the union of OLPs is not necessarily an OLP. For this, consider

$$(\Pi_1, <_1) = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow a,\ r_2 <_1 r_1\}$$
$$(\Pi_2, <_2) = \{r_3 : a \leftarrow,\ r_4 : b \leftarrow a,\ r_3 <_2 r_4\}$$

Then, we have

$$(\Pi^*, <^*) = \{r_5 : a \leftarrow,\ r_6 : b \leftarrow a,\ r_5 <^* r_6,\ r_6 <^* r_5\}$$

where $<^*$ is not a strict partial order. Hence, $(\Pi^*, <^*)$ is not an OLP. To avoid this, we define the notion of admissible extensions of an OLP.

**Definition 3.2** *Let* $(\Pi, <)$ *and* $(\Pi', <')$ *be OLPs.*

*Then,* $(\Pi', <')$ *is an admissible extension of* $(\Pi, <)$ *if* $(\Pi \cup \Pi', < \cup <')$ *is an OLP.*

We remark that the notion of compatibility for orderings, as defined in [Brewka and Eiter, 1999], is similar in spirit to admissible extensions, but concerns only the order relation and not the rules in the program.

Analogously to (normal) logic programs, we can define the notion of equivalence of OLPs.

**Definition 3.3** *(Order equivalence) Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs and* $\sigma \in \{D, B, W\}$*. Then,* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are* $<^\sigma$-equivalent*, denoted* $(\Pi_1, <_1) \equiv^\sigma (\Pi_2, <_2)$*, iff* $AS^\sigma((\Pi_1, <_1)) = AS^\sigma((\Pi_2, <_2))$.

Since the definition of strong equivalence for logic programs can be interpreted as having the same answer sets in any extension, we employ the notion of admissible extensions for defining the analogon for OLPs.

**Definition 3.4** *(Strong order equivalence) Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs and* $\sigma \in \{D, B, W\}$.

*Then,* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are* strongly $<^\sigma$-equivalent *iff for all admissible extensions* $(\Pi, <)$ *of* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *it holds that* $(\Pi_1 \cup \Pi, <_1 \cup <)$ *and* $(\Pi_2 \cup \Pi, <_2 \cup <)$ *are* $<^\sigma$*- equivalent.*

Abbreviatory, we write $\equiv_s^\sigma$ whenever two OLPs are strongly $<^\sigma$-equivalent.

## 4 Properties and Relationships

Next, we want to characterize which conditions two OLPs must fulfill s.t. they are strongly order equivalent.

The next theorem states that strong order equivalence implies strong equivalence of the underlying logic programs.

**Theorem 4.1** *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs and* $\sigma \in \{D, W, B\}$*. If* $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ *then* $\Pi_1 \equiv_s \Pi_2$.

Intuitively, if the logic programs are not strongly equivalent, one extension of them admits an answer set $A$, which is no answer set of the extension of the other program. If the corresponding ordered extensions have the same answer sets, this means that $A$ is inhibited by the preferences. But then one can always construct a further extension, which regenerates $A$ for one program, but not for the other. For example, programs

$$(\Pi_1, \emptyset) \quad = \{r_1 : a \leftarrow\}$$
$$(\Pi_2, <_2) \quad = \{r_2 : a \leftarrow not\ b,\ r_3 : b \leftarrow not\ a,\ r_3 <_2 r_2\}$$

have both the preferred answer set $\{a\}$, whereas $\Pi_2$ has additionally the answer set $\{b\}$. Taking $(\Pi, \emptyset) = \{r_4 : b \leftarrow y,\ r_5 : y \leftarrow\}$ yields $\{a, b, y\}$ as preferred answer set of $(\Pi_1 \cup \Pi, \emptyset)$, but $\{b, y\}$ as preferred answer set of $(\Pi_2 \cup \Pi, <_2)$.

Note that one can create such a "compensating" program for any answer set which is inhibited by the preference relation. For this, we use new symbols and new rules, which are

not involved in the preference relation, to "repair" this answer set. Therefore strong equivalence is a necessary condition for strong order equivalence.

Next, we turn our attention to the set of generation rules. For this, consider the following OLPs:

$$(\Pi_1, \emptyset) = \{r_1 : a \leftarrow,\ r_3 : b \leftarrow\}$$
$$(\Pi_2, \emptyset) = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow a\}$$

We observe that $\Pi_1 \equiv_s \Pi_2$, but taking

$$(\Pi^*, <^*) = \{r_2 : b \leftarrow a,\ r_4 : y \leftarrow not\ b,\ r_2 <^* r_4\}$$

yields $\{a, b\} \in AS^\sigma((\Pi_1 \cup \Pi^*, <^*))$ but $AS^\sigma((\Pi_2 \cup \Pi^*, <^*)) = \emptyset$ for $\sigma \in \{D, W, B\}$. They are not strongly order equivalent because they do not coincide on their set of generating rules. $(\Pi_1, \emptyset)$ contains $r_3$ and $(\Pi_2, \emptyset)$ rule $r_2$ to derive the atom $b$. Adding $r_4$ and the preference that $r_4$ is higher preferred to $r_2$ entails that $(\Pi_2 \cup \Pi^*, <^*)$ has no preferred answer set since $r_4$ is blocked by the lower preferred rule $r_2$, while $(\Pi_1 \cup \Pi^*, <^*)$ has a preferred answer set since $r_3$ allows to block $r_4$ in an alternative way. This allows us to suppose that strong order equivalence enforces that the OLPs must coincide on all rules which can become applicable. For this, we say that a rule $r \in \Pi$ *contributes* to an answer set $X$, if there exists an $\Pi^*$ such that $X \in AS(\Pi \cup \Pi^*)$ and $r \in R_{\Pi \cup \Pi^*}(X)$. Rules which are not contributing to answer sets, are never applicable, e.g. $body^+(r) \cap body^-(r) \neq \emptyset$. For $B$–preferences, we obtain that two strongly order equivalent programs must coincide on their rules contributing to answer sets.

**Theorem 4.2** *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be OLPs. If $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$, then for all $\Pi$ and for all $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ we have $R_{\Pi_1 \cup \Pi}(X) = R_{\Pi_2 \cup \Pi}(X)$.*

Interestingly, for $D$– and $W$–preferences, these conditions have to be weakened.

**Theorem 4.3** *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be OLPs and $\sigma \in \{D, W\}$. If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$, then for all $\Pi$ and for all $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ we have $R_{\Pi_1 \cup \Pi}(X) \setminus \{r \in \Pi_1 \cup \Pi \mid head(r) \in body^+(r)\} = R_{\Pi_2 \cup \Pi}(X) \setminus \{r \in \Pi_2 \cup \Pi \mid head(r) \in body^+(r)\}$.*

Since $D$- and $W$- preferences couple rule application with preference handling, answer sets can be inhibited by preferences because applied rules are not enumerable in an order preserving way (see Condition 2 in Definition 2.1 and 2.2). But one can always find an extension such that those answer sets are regenerated. For applied "loop" rules, i.e. $head(r) \in body^+(r)$, another rule must always exist which derives the head in an order preserving way. Hence, such loop rules are redundant wrt. $D$- and $W$- preferences as long as they are not involved in the preference relations. In contrast, $B$- preferences decouple preference handling from rule application and hence, one can always find an extension, where those loop rules regenerate a non-preferred answer set to a preferred one in one program but not in the other one. For example,

$$(\Pi_1, \emptyset) = \{r_1 : a \leftarrow\} \quad (\Pi_2, \emptyset) = \{r_1 : a \leftarrow,\ r_2 : a \leftarrow a\}$$

are not strongly $<^B$- equivalent. Taking

$$(\Pi, <) = \{r_1 : a \leftarrow,\ r_3 : y \leftarrow not\ a,\ r_1 < r_3\}$$

yields $AS^B((\Pi_1 \cup \Pi, <)) = \emptyset$ but $AS^B((\Pi_2 \cup \Pi, <)) = \{a\}$. Here, rule $r_2$ can be used to block $r_3$. For $D$- and $W$-preferences, however, $r_2$ has to be ordered after $r_1$, hence it can not be used to block $r_3$.

In the following we will concentrate on the preference relations. For this consider the program

$$\Pi = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow a\}$$

and the preference relation $r_1 > r_2$. We obtain that $(\Pi, \emptyset) \equiv^\sigma (\Pi, <)$ holds for all $\sigma \in \{D, W, B\}$ and $(\Pi, \emptyset)$ and $(\Pi, <)$ coincide on their set of rules. For $(\Pi, \emptyset)$, rule $r_1$ must be considered before rule $r_2$ in $D$- and $W$-preference, since the derivation of the positive body of $r_2$ depends from $r_1$. Thus, adding the preference relation $r_1 > r_2$ to $(\Pi, \emptyset)$ has no effect on $<^D$- and $W$- preferred answer sets. Surprisingly, this does not hold when considering strong order equivalence. Taking

$$(\Pi', <') = \left\{ \begin{array}{lll} r_1 : a \leftarrow & r_3 : z \leftarrow & r_5 : a \leftarrow x \\ r_2 : b \leftarrow a & r_4 : y \leftarrow not\ z & r_6 : x \leftarrow \\ r_3 <' r_2 & r_1 <' r_4 & \end{array} \right\}$$

yields $AS^\sigma((\Pi \cup \Pi', <')) = \{\{a, b, x, z\}\}$ while $AS^\sigma((\Pi \cup \Pi', < \cup <')) = \emptyset$. For $(\Pi, <) \cup (\Pi', <')$ we get the transitive preference $r_4 > r_3$, implying that $r_4$ is blocked by a lower preferred rule, which we don't obtain for $(\Pi, \emptyset) \cup (\Pi', <')$. Since one can always construct such a program (observe that the relevant rules contain new symbols), it follows that two OLPs can never be strongly equivalent if the preference relations do not coincide.

**Theorem 4.4** *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be OLPs and $\sigma \in \{D, W, B\}$. If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ then $<_1 = <_2$.*

As a special case we obtain that $(\Pi, \emptyset) \not\equiv_s^\sigma (\Pi, <)$ holds for any program $\Pi$, any non-empty preference relation on $\Pi$, and any $\sigma \in \{D, W, B\}$. Moreover, for every OLP $(\Pi, <)$, $< \neq \emptyset$, there exists no strongly order equivalent program $(\Pi', \emptyset)$.

**Corollary 4.5** *Let $(\Pi, <)$ be an OLP such that $< \neq \emptyset$ and $\sigma \in \{D, W, B\}$. Then, there exists no logic program $\Pi'$ such that $(\Pi, <) \equiv_s^\sigma (\Pi', \emptyset)$.*

Interestingly, this tells us that no transformation from an OLP with a non-empty preference relation to a logic program exists such that the transformed program is strongly order equivalent to the original one. The results given above are sufficient for characterizing strong order equivalence.

**Theorem 4.6** *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be OLPs. If $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and for all $\Pi$ and for all $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ we have $R_{\Pi_1 \cup \Pi}(X) = R_{\Pi_2 \cup \Pi}(X)$, then $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$.*

**Theorem 4.7** *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be OLPs and $\sigma \in \{D, W\}$. If $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and for all $\Pi$ and for all $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ we have $R_{\Pi_1 \cup \Pi}(X) \setminus \{r \in \Pi_1 \cup \Pi \mid head(r) \in body^+(r)\} = R_{\Pi_2 \cup \Pi}(X) \setminus \{r \in \Pi_2 \cup \Pi \mid head(r) \in body^+(r)\}$, then $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.*

For example,

$$(\Pi_1, \emptyset) = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow not\ a\}$$
$$(\Pi_2, \emptyset) = \{r_1 : a \leftarrow,\ r_3 : c \leftarrow not\ a\}$$

are strongly order equivalent in all three semantics, whereas

$$(\Pi_3, \emptyset) = \{r_1 : a \leftarrow\} \quad (\Pi_4, \emptyset) = \{r_1 : a \leftarrow,\ r_2 : a \leftarrow a\}$$

are only strongly $<^D$- (and $<^W$-) equivalent, but not strongly $<^B$- equivalent. With these results we can formalize the following characterization of strong order equivalence.

**Corollary 4.8** *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs. Then,* $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$ *iff* $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, *and for all* $\Pi$ *and for all* $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ *we have* $R_{\Pi_1 \cup \Pi}(X) = R_{\Pi_2 \cup \Pi}(X)$.

**Corollary 4.9** *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs and* $\sigma \in \{D, W\}$. *Then,* $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$, *iff* $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, *and for all* $\Pi$ *and for all* $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ *we have* $R_{\Pi_1 \cup \Pi}(X) \setminus \{r \in \Pi_1 \cup \Pi \mid head(r) \in body^+(r)\} = R_{\Pi_2 \cup \Pi}(X) \setminus \{r \in \Pi_2 \cup \Pi \mid head(r) \in body^+(r)\}$.

Regarding preferred answer sets, the different strategies for preference handling yield an increasing number of preferred answer sets. Interestingly, the above given theorems show that there is no difference in the behavior of $D$– and $W$–preferences when considering strong order equivalence. In contrast, fewer programs are strongly order equivalent in the $B$–semantics, since preference handling is decoupled from rule application.

**Theorem 4.10** *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be OLPs. Then (a) if* $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$ *then* $(\Pi_1, <_1) \equiv_s^W (\Pi_2, <_2)$, *(b)* $(\Pi_1, <_1) \equiv_s^W (\Pi_2, <_2)$ *iff* $(\Pi_1, <_1) \equiv_s^D (\Pi_2, <_2)$.

Considering answer sets, each $<^D$-preferred answer set is $<^W$-preferred and each $<^W$-preferred is $<^B$-preferred. Interestingly, regarding strong order equivalence we obtain that $\equiv_s^D$ iff $\equiv_s^W$, so the differences between the $D$- and $W$- semantics can be compensated by extending the program in a suitable way. Moreover, since $\equiv_s^B$ implies $\equiv_s^W$ and $\equiv_s^B$ implies $\equiv_s^D$, the $B$-semantics imposes a strictly stronger criterion for programs to be strongly order equivalent. The reason for this is that $B$-preference decouples preference handling from rule application, and in addition, these differences cannot be compensated by adding suitable extensions.

## 5 Program Simplifications

Corollaries 4.9 and 4.8 show that two OLPs which are strongly order equivalent, differ in rules which are not involved in the preference relation and which are never applicable. Additionally, for $D$- and $W$- semantics, strongly order equivalent logic programs may differ in loop rules, in which the head is contained in the positive body. These results allow us to formalize the following results concerning transformations on OLPs. In the following, we say that a rule $r$ is involved in a preference relation $<$ if there exists an $r'$ such that $r < r'$ or $r' < r$ holds.

**Corollary 5.1** *Let* $(\Pi, <)$ *be an OLP such that for* $r \in \Pi$, *which is not involved in* $<$, *for all* $\Pi'$ *and* $X \in AS(\Pi \cup \Pi')$ *we have* $r \notin R_{\Pi \cup \Pi'}(X)$. *Then,* $(\Pi, <) \equiv_s^\sigma (\Pi \setminus \{r\}, <)$ *holds for* $\sigma \in \{D, W, B\}$.

This corollary expresses that a rule, which is never applicable and not involved in $<$, is redundant regarding strong order equivalence. The next corollary concentrates on loop rules.

**Corollary 5.2** *Let* $(\Pi, <)$ *be an OLP and* $\sigma \in \{D, W\}$. *Furthermore, let* $r \in \Pi$ *s.t.* $head(r) \in body^+(r)$ *and* $r$ *is not involved in* $<$. *Then,* $(\Pi, <) \equiv_s^\sigma (\Pi \setminus \{r\}, <)$.

In [Eiter *et al.*, 2004; Osorio *et al.*, 2001] transformations on logic programs are reported, which can be used for simplifying a program. For those modular transformations a program must be strongly equivalent to the transformed one. Considering OLPs, we observe that these transformations in general do not guarantee strong order equivalence. Transformation *TAUT* expresses that for all logic programs $\Pi$ and all $r \in \Pi$ where $head(r) \in body^+(r)$ we have $\Pi \equiv_s \Pi \setminus \{r\}$. Corollary 5.2 implies that the analogon for OLPs only holds for the $D$- and $W$- semantics as long as $r$ is not involved in $<$. Transformation *RED*$^-$ expresses that for a logic program $\Pi$ and for rules $r_1, r_2$, where $head(r_2) \in body^-(r_1)$ and $body(r_2) = \emptyset$, we have $\Pi \equiv_s \Pi \setminus \{r_1\}$. Since $r_2$ is always a generating rule, regardless which rules we add to $\Pi$, $r_1$ never contributes to an answer set. Hence, *RED*$^-$ is feasible for OLPs unless $r_1$ is involved in $<$. For $r \in \Pi$, where $body^+(r) \cap body^-(r) \neq \emptyset$ we have that $\Pi \equiv_s \Pi \setminus \{r\}$ (transformation *CONTRA*). Since such a rule is never applicable, this result can be carried over to OLPs whenever this rule is not involved in $<$. Transformation *NONMIN* states that for $r_1, r_2 \in \Pi$, where $head(r_2) = head(r_1)$ and $body(r_2) \subseteq body(r_1)$, we have that $\Pi \equiv_s \Pi \setminus \{r_1\}$ holds. Since there is no information whether $r_1$ can become applicable or not, this transformation can only be made on OLPs whenever $r_1$ is not involved in $<$ and never becomes applicable. For example, consider the program:

$$(\Pi_1, \emptyset) = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow,\ r_3 : c \leftarrow a,\ r_4 : c \leftarrow a, b\}$$

Here we have that $head(r_3) = head(r_4)$ and $body(r_3) \subseteq body(r_4)$, hence, $\Pi_1 \equiv_s \Pi_1 \setminus \{r_4\}$. Since $r_4$ is applicable, the simplified program is not strongly order equivalent. Taking

$$(\Pi^*, <^*) = \{r_3 : c \leftarrow a,\ r_5 : y \leftarrow not\ c,\ r_3 <^* r_5\}$$

yields $\{a, b, c\} \in AS^\sigma((\Pi_1 \cup \Pi^*, <^*))$ but $AS^\sigma(((\Pi_1 \setminus \{r_4\}) \cup \Pi^*, <^*)) = \emptyset$, for $\sigma \in \{D, W, B\}$. Thus, $(\Pi_1, \emptyset) \not\equiv_s^\sigma (\Pi_1 \setminus \{r_4\}, \emptyset)$. Similar considerations apply for the transformation *S-IMP*, where for all $r, r' \in \Pi$ such that there exists an $A \subseteq body^-(r')$ such that $head(r) \in head(r') \cup A$, $body^-(r) \subseteq body^-(r') \setminus A$ and $body^+(r) \subseteq body^+(r')$ we have $\Pi \equiv_s \Pi \setminus \{r'\}$. This transformation can only be carried over to OLPs whenever $r'$ is not involved in $<$ and never becomes applicable. For example, taking

$$\Pi = \{r' : a \leftarrow b, not\ c,\ r : a \leftarrow\}$$

and $A = \{c\}$ yields $\Pi \equiv_s \{r\}$, where $r'$ is applicable wrt $\{a, b\} \in AS(\Pi \cup \{b \leftarrow\})$. But $(\Pi, \emptyset) \not\equiv_s^\sigma (\{r\}, \emptyset)$, since for

$$(\Pi^*, <^*) = \{r : a \leftarrow,\ r^y : y \leftarrow not\ a,\ r^b : b \leftarrow,\ r <^* r^y\}$$

we get $\{a, b\} \in AS^\sigma((\Pi \cup \Pi^*, <^*))$ but $\emptyset = AS^\sigma((\{r\} \cup \Pi^*, <^*))$ for $\sigma \in \{D, W, B\}$. Transformation *WGPPE* states for a rule $r_1 \in \Pi$, where $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid$

$head(r_2) = a\}$ and $G_a \neq \emptyset$ that $\Pi \equiv_s \Pi \cup G'_a$ holds where $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$. As with the other transformations obtained for strong equivalence, this one is transferable to strong order equivalence if the rules from $G_a$ become never applicable and are not involved in the preference relation. E.g., for

$$\Pi = \{r_1 : b \leftarrow a,\ r_2 : a \leftarrow\}$$

we have that $\Pi \equiv_s \Pi \cup \{b \leftarrow\}$ holds. By taking

$$(\Pi^*, <^*) = \{r_1 : b \leftarrow a,\ r^y : y \leftarrow not\ b,\ r^y >^* r_1\}$$

we obtain $(\Pi, \emptyset) \not\equiv_s^\sigma (\Pi \cup \{b \leftarrow\}, \emptyset)$ since $(\Pi \cup \Pi^*, <^*)$ has no $<^\sigma$– preferred answer set while $(\Pi \cup \{b \leftarrow\} \cup \Pi^*, <^*)$ has $<^\sigma$– preferred answer set $\{a, b\}$, for $\sigma \in \{D, W, B\}$.

## 6 Conclusions and Future Work

We have presented the notion of strong order equivalence for ASP with preferences. We have provided an extensive analysis of this novel notion, and studied the relationship between different preference semantics for it. We have also analyzed the conditions for certain simplification methods to guarantee strong order equivalence (hence applicability to modules).

We could show in Corollaries 4.8 and 4.9 that two programs are strongly $W$- and $D$-equivalent, if (and only if) the preference relation is equal, their standard ASP programs are strongly equivalent, and their "non-looping" generating rules are identical for all answer sets of any extension of them. For $B$-preference, also the "looping" generating rules must be identical in any answer set.

¿From Theorem 4.10 we obtain that exactly the same pairs of programs are strongly $<^D$- and $<^W$- equivalent, while not all of them are strongly $<^B$- equivalent. That is, the semantical differences between $D$- and $W$- preference have no longer any effect when considering strong order equivalence. On the contrary, for $B$- semantics the differences to $D$- and $W$- preferences are strengthened under strong order equivalence, since it decouples preference handling from rule application, which can not be compensated by suitable extensions.

We have furthermore studied possibilities to simplify ordered programs. In Corollaries 5.1 and 5.2 we have given abstract conditions for simplifications, and have assessed under which circumstances simplification rules from standard ASP can be applied on ASP modules with preferences.

The reader may have noted that we did not define a notion which would correspond to SE-models in standard ASP strong equivalence. Indeed there seem to be considerable difficulties: SE-models can only express statements on atoms, and not directly on rules. However, as demonstrated e.g. in Theorem 4.3, strong order equivalence depends heavily on the rules involved in the program. We therefore conjecture that "SE-models" for $D$-, $W$-, and $B$- preferred answer sets must include a means for stating relationships among rules, and leave this topic for future work.

Another way of analyzing strong order equivalence could be offered by a compilation of the programs, as done e.g. in the `plp`-System [Delgrande *et al.*, 2003; plp, 2002]. There, ordered programs are compiled into logic programs such that their answer sets directly correspond to the preferred answer sets of the original ordered program. Since the compilation is a logic program, one could suppose that two ordered programs are strongly order equivalent if the compilation of them possess the same SE-models. However, the compilation introduces additional symbols. Hence, the answer sets of two compiled programs, which represent the same preferred answer set, are in general incomparable because of these additional symbols. One would therefore have to consider strong equivalence under a projection of the answer sets. Furthermore, for an exact notion one would also have to take into account that compiled programs have a particular, rather than arbitrary, structure. Such a notion of strong equivalence has (to the best of our knowledge) not been studied so far. Therefore, we leave this kind of analysis for future work.

Corollary 4.5 shows that no transformation from an ordered program $(\Pi, <)$ into a logic program $\Pi'$ exists such that $(\Pi, <)$ is strongly order equivalent to $\Pi'$. Also, Corollaries 4.8 and 4.9 describe rigorous conditions on ordered programs being strongly order equivalent under the considered preference semantics. Hence, further work could be to define and study weaker notions of order equivalence.

## References


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

[Delgrande *et al.*, 2003] J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *TPLP*, 3(2):129–187, 2003.

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

[Eiter *et al.*, 2004] T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying logic programs under uniform and strong equivalence. In *LPNMR'04*, pages 87–99. Springer, 2004.

[Eiter *et al.*, 2002] T. Eiter, M. Fink, G.Sabbatini, and H. Tompits. A Generic Approach for Knowledge-Based Information Site Selection. In *KR-02*, pages 459–469. Morgan Kaufmann, 2002.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP*, pages 1070–1080. The MIT Press, 1988.

[Lifschitz *et al.*, 2001] V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM TOCL*, 2(4):526–541, 2001.

[Osorio *et al.*, 2001] M. Osorio, J.A. Navarro, and J. Arrazola. Equivalence in answer set programming. In *LOPSTR*, pages 57–75. Springer, 2001.

[plp, 2002] http://www.cs.uni-potsdam.de/∼torsten/plp, 2002.

[Schaub and Wang, 2003] T. Schaub and K. Wang. A semantic framework for preference handling in answer set programming. *TPLP*, 3(4-5):569–607, 2003.

[Turner, 2003] H. Turner. Strong equivalence made easy: nested expressions and weight constraints. *TPLP*, 3(4-5):609–622, 2003.

[Wang *et al.*, 2000] K. Wang, L. Zhou, and F. Lin. Alternating Fixpoint Theory for Logic Programs with Priority. In *CL 2000*, pages 164–178. Springer.