0pt

Preferred Answer Sets for Extended Logic Programs

Gerhard Brewka
Universität Leipzig
Institut für Informatik
Augustusplatz 9/10, D-04109 Leipzig
Germany
brewka@informatik.uni-leipzig.de

Thomas Eiter
Justus-Liebig-Universität Gießen
Institut für Informatik
Arndtstraße 2, D-35392 Gießen
Germany
eiter@informatik.uni-giessen.de

Abstract

In this paper, we extend Gelfond and Lifschitz's answer set semantics to prioritized extended logic programs. In such programs, an ordering on the program rules is used to express preferences. We show how this ordering can be used to define preferred answer sets and thus to increase the set of consequences of a program. We define a strong and a weak notion of preferred answer sets. The first takes preferences more seriously and satisfies reasonable principles for priority handling in rule based systems, while the second guarantees the existence of a preferred answer set for consistent programs. We show that strong preference on answer sets does not add on the complexity of the principal reasoning tasks, and weak preference leads only to a mild increase in complexity.

1 Introduction

Preferences among default rules play an important role in applications of nonmonotonic reasoning. One source of preferences that has been studied intensively is specificity [27, 35, 36]. In case of a conflict between defaults we tend to prefer the more specific one since this default provides more reliable information.

Specificity is an important source of preferences, but not the only one. In the legal domain it may, for instance, be the case that a more general rule is preferred since it represents federal law as opposed to state law [29]. In these cases preferences may be based on some basic principles regulating how conflicts among legal rules are to be resolved. Also in other application domains, like model based diagnosis, configuration or decision making, preferences play a fundamental role and their relevance is well-recognized.

Prioritized versions for most of the existing nonmonotonic formalisms have been proposed, e.g., prioritized circumscription [16], hierarchic autoepistemic logic [19], prioritized default logic [24, 5, 1], prioritized theory revision [3, 25], or prioritized abduction [11]. Somewhat surprisingly, preferences have received less attention in logic programming. This may be explained by the fact that for a long period, logic programming was mainly conceived as a logical paradigm for declarative programming. However, in the recent past, it has become evident that logic programming can also serve as a powerful framework for knowledge representation, cf. [13, 2]. If logic programming wants to successfully stand this challenge, it must provide features like preference handling which have been recognized as indispensable in the context of knowledge representation.

In this paper we consider extended logic programs under answer set semantics supplied with priority information, which is given by a supplementary strict partial ordering of the rules. This additional information is used to solve potential conflicts; that is, we want to conclude more than in standard answer set semantics. On the other hand, unless the program with the given preference information is unsatisfiable (in a sense to be made precise) we want to conclude only literals that are contained in at least one answer set. The best way to achieve these goals is to use the preferences on rules for selecting a subset of the answer sets, which we call the *preferred answer sets*.

Our approach is based on the following two main ideas. The first is that the application of a rule with nonmonotonic assumptions means to jump to a conclusion, and this conclusion is yet another assumption which has to be used globally in the program for the issue of deciding whether a rule is applicable or not. The second is that the rules must be applied in an order compatible with the priority information. We take this to mean that a rule is applied *unless it is defeated via its assumptions by rules of higher priorities.* This view is new and avoids the unpleasant behavior which is present with other approaches.

The paper is organized as follows. The next section recalls the definitions of extended logic programs and introduces basic notations. Section 3 introduces two basic principles for preference handling in rule based nonmonotonic systems, reviews some approaches to prioritized default logic and demonstrates that they fail to satisfy the principles. In Section 4, we then present our approach, by introducing the concept of preferred answer sets. We demonstrate the approach on a number of examples, and investigate in Section 5 its properties. The existence of preferred answer sets for consistent programs is not guaranteed. We define in Section 6 also a weak notion of preferred answer sets. Intuitively, the strong notion of answer sets takes preferences more seriously, which may lead to a situation in which the priorities are incompatible with the answer set condition; this is avoided in the weaker notion. In Section 7, we address complexity issues. Section 8 discusses further related work and concludes.

Due to space limitations all proofs are left out. They are contained in the full version of the paper which is available from the authors.

2 Preliminaries and Notation

We assume that the reader is familiar with the basic concepts of logic programming; see [21] for background. In the present paper, we focus on extended logic programs, which have two kinds of negation, as described in [15].

As usual, let $\mathcal{L}$ be an underlying countable first-order language. Unless stated otherwise, $\mathcal{L}$ is the language generated by the program or rule base under consideration.

A *rule* r is a formula

$$c \leftarrow a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m \qquad (1)$$

where the a_i, b_j and c are classical literals, i.e., either positive atoms or atoms preceded by the classical negation sign $\neg$. We denote by $head(r)$ the head of rule r. The symbol not denotes negation by failure (*weak negation*), while $\neg$ denotes *strong negation* (sometimes called explicit negation). We will call $a_1, \ldots, a_n$ the *prerequisites* of the rule and use $pre(r)$ to denote the set of prerequisites of r. A rule is called *prerequisite-free*, if $n = 0$.

A *rule base* R is a (possibly infinite) collection of rules; an *extended logic program* (*logic program* or *program*, for short) P is a finite rule base.

As usual, a rule (resp. rule base, program) is *ground*, if no variable occurs in it; a rule base (resp., program) is prerequisite-free, if all rules in it are prerequisite-free.

For a rule base R, we denote by R^* the ground instantiation of R over the Herbrand universe of the language $\mathcal{L}$. Moreover, we denote by $Lits$ the set of all classical ground literals of $\mathcal{L}$.

We say a rule r of the form (1) is *defeated by a literal* ℓ, if $\ell = b_i$ for some $i \in \{1, \ldots, m\}$, and we say it is *defeated by a set of literals* X, if X contains a literal that defeats r.

Let us recall the definition of the answer set semantics for extended logic programs [15]. Answer sets are defined in analogy to stable models [14], but taking into account that atoms may be preceded by classical negation.

Definition 2.1 *Let R be a collection of ground rules, and let $X \subseteq Lits$ be a set of ground literals. The reduct of R with respect to X (for short, X-reduct of R), denoted R^X, is the collection of rules resulting from R by*

- *deleting each rule which is defeated by X, and*
- *deleting all weakly negated literals from the remaining rules.*

This reduction of R is often called *Gelfond-Lifschitz reduction*, after its inventors [14].

Definition 2.2 *Let R be a collection of ground rules without weak negation. Then, $Cn(R)$ denotes the smallest set $S \subseteq Lits$ of ground literals such that*

1. *S is closed under R, i.e., for any rule $a \leftarrow a_1, \ldots, a_n$ in R, if $a_1, \ldots, a_n \in S$, then $a \in S$; and*
2. *S is logically closed, i.e., either S is consistent or $S = Lits$.*

Definition 2.3 *Let R be a collection of ground rules. Define an operator $\gamma_R(X)$ on the sets X of ground literals as follows:*

$$\gamma_R(X) = Cn(R^X)$$

Then, a set X of ground literals is an answer set of R iff $X = \gamma_R(X)$.

The collection of answer sets of R is denoted by $\mathcal{AS}(R)$. For an arbitrary logic program P, the collection of answers sets $\mathcal{AS}(P)$ is given by $\mathcal{AS}(P) = \mathcal{AS}(P^*)$.

A ground literal L is a consequence of a program P under answer set semantics, denoted $P \models L$, iff L is contained in all answer sets of P. For more on answer sets, consult [15].

3 Why Existing DL-Approaches Do Not Work

Different prioritized versions of Reiter's default logic [31] have been proposed in the literature, e.g. [24, 5, 1, 32, 9]. We will show that all of them suffer from weaknesses and thus cannot serve –via the standard translation from extended logic programs to default theories– as a satisfactory specification of preferred answer sets.

3.1 Principles for priorities

Before discussing these approaches, we want to formulate two principles which as we believe should be satisfied by any system which is based on prioritized defeasible rules. Since we want our principles to cover different approaches like default logic, prioritized logic programs, etc. we use in this section the generic terms *belief set* (for extension, answer set, ...) and *prioritized theory* (for prioritized default theory, prioritized logic program,...) in our formulations.

The first principle can be viewed as a meaning postulate for the term "preference" and states a minimal requirement for preference handling in rule based systems:

Principle I.
Let B_1 and B_2 be two belief sets of a prioritized theory $(T, <)$ generated by the (ground) rules $R \cup \{d_1\}$ and $R \cup \{d_2\}$, where $d_1, d_2 \notin R$, respectively. If d_1 is preferred over d_2, then B_2 is not a (maximally) preferred belief set of T.

In this context, a rule r is said to *generate* a belief set B, if the prerequisites of r are in B and B does not defeat r. It is hard to see how the use of the term "preference among rules" could be justified in cases where Principle I is violated.

The second principle is related to relevance. It tries to capture the idea that the decision whether to believe a formula p or not should depend on the priorities of rules contributing to the derivation of p only, not on the priorities of rules which become applicable when p is believed:

Principle II.
Let B be a preferred belief set of a prioritized theory $(T, <)$, r a (ground) rule such that at least one prerequisite of r is not in B. Then B is a preferred belief set of $(T \cup \{r\}, <')$ whenever $<'$ agrees with $<$ on priorities among rules in T.

Thus, adding a rule which is not applicable in a preferred belief set can never render this belief set non-preferred unless new preference information changes preferences among some of the old rules (e.g. via transitivity). In other words, a belief set is not blamed for not applying rules which are not applicable.

We will see that each of the existing treatments of preferences for default logic, described in [24, 5, 1, 32, 9], violates one of these principles.

3.2 Control of Reiter's quasi-inductive definition

The first group of proposals [24, 5, 1] uses preferences to control the quasi-inductive definition of extensions [31]: in each step of the generation of extensions the defaults with highest priority whose prerequisites have already been derived are applied. This leads to situations where the preferred extensions do not take seriously what they believe. It may be the case that a less preferred default is applied although the prerequisite of a conflicting, more preferred default is believed in a preferred extension. As we will see, this leads to the violation of Principle I.

The mentioned approaches differ in technical detail. We do not want to present the exact definitions here. Instead, we will illustrate the difficulties using an example for which all three approaches obtain the same result.

Example 3.1 Assume we are given the following default theory:[1]

(1) $a : b/b$
(2) $true : \neg b/\neg b$
(3) $true : a/a$

Assume further that (1) is preferred over (2) and (2) over (3). This default theory has two Reiter extensions, namely $E_1 = Th(\{a, b\})$, which is generated by rules (1) and (3), and $E_2 = Th(\{a, \neg b\})$, which is generated by rules (2) and (3). The single preferred extension in the approaches mentioned above is E_2. The reason is that the prerequisite of (2) is derived before the prerequisite of (1) in the construction of the extension. The approaches thus violate Principle I. ∎

[1] We assume that the reader knows the basic concepts of standard default logic [31]; informally, a default rule $a{:}b/c$ corresponds to the clause $c \leftarrow \mathrm{not}\neg b, a$.

3.3 Rintanen's approach

Rintanen [32] uses a total order on (normal) defaults to induce a lexicographic order on extensions.

Call a default rule $r = a{:}b/b$ *applied* in a set of formulas E (denoted $r \in appl(E)$), if a and b are in E. An extension E is then preferred over extension E', if and only if there is a default $r \in appl(E) \setminus appl(E')$ satisfying the following condition: if r' is preferred over r and $r' \in appl(E')$, then $r' \in appl(E)$.

Unfortunately, also this approach leads to counterintuitive results and to a violation of our principles.

Example 3.2 Consider the following default theory, which is similar to the one in Example 3.1:

(1) $a : b/b$
(2) $true : \neg a/\neg a$
(3) $true : a/a$

Again (1) is preferred over (2), and (2) over (3). The default theory has two Reiter extensions, namely $E_1 = Th(\{\neg a\})$ and $E_2 = Th(\{a, b\})$. Intuitively, since the decision whether to believe a or not depends on (2) and (3) only, and since (2) is preferred over (3), we would expect to conclude $\neg a$, in other words, to prefer E_1.

Rintanen, however, prefers E_2. The reason is that in E_2 default (1) is applied. Belief in a is thus accepted on the grounds that this allows us to apply a default of high priority. This is far from being plausible and amounts to wishful thinking. It is also easy to see that Principle II is violated: E_1 clearly is the single preferred extension of rules (2) and (3) in Rintanen's approach. Adding rule (1) which is not applicable in E_1 makes E_1 a non-preferred extension. ∎

4 Prioritized Programs and Preferred Answer Sets

In this section, we present our approach for incorporating priorities into extended logic programs. In this approach, priorities are specified like in other approaches by an ordering of the rules. This ordering will be used to test if an answer set is constructed by applying the rules in a proper order. However, and this is the salient point of our approach, the proper order of rule application will not be enforced during a quasi-inductive construction of an answer set, but will rather be ensured in a separate additional condition which requires a dual reconstruction of the answer set. Our approach derives from the following two underlying ideas:

(1) Applying rules with default negation means to jump to conclusions, and any such a conclusion has to be used globally throughout the program when the applicability of a rule is tested.

(2) Rules must be applied in an order compatible with the priorities. It appears that this is intuitively the case if each rule whose prerequisites are true and whose assumptions are not defeated by the rules with higher priority, is applied.

4.1 Prioritized extended logic programs

We start with the syntactical part of our framework.

Definition 4.1 *A prioritized rule base is a pair $\mathcal{R} = (R, <)$ where P is a rule base and $<$ a strict partial order on R. In particular, $\mathcal{R}$ is a prioritized (extended) logic program, if R is an (extended) logic program.*

Recall that a strict partial order is an irreflexive and transitive relation. The order $<$ is used to express preference information: $r_1 < r_2$ stands for "r_1 has higher priority than r_2".

We use here a partial order rather than a total order on the rules, which is appropriate for different reasons. The first reason is that in some scenarios it may be unwanted or even unnatural to specify an order between rules. A second reason is that, as rules represent their ground instances, we can pass easily from a program P with variables to its ground instantiation P^* while the intuitive meaning of priorities on rules is preserved. E.g., if we have rules such that $r_1 < r_2$, then by passing to the ground instances, the naturally induced partial order says that every instance of r_1 has higher priority than any instance of r_2, while no priorities are given between the instances of r_1 and of r_2, respectively.

If we adopt the view that rules should be applied one at a time, then a partial ordering $<$ on the rules is a representative of all possible refinements of $<$ to total orderings. However, not all total orderings are intuitively acceptable. It is natural to assume that in any totally ordered rule base, some rule has highest priority. This reflects the view of proceeding from most important to less important matters, and means that infinitely decreasing chains $r_1 > r_2 > r_3 > \ldots$ of rules are excluded. Formally, this amounts to the condition that the ordering $<$ on R is well-founded. Any such ordering is called a *well-ordering*.

Each total ordering of a finite set is trivially a well-ordering, while this is not true for infinite sets. It is well-known that each well-ordering $<$ of a set M corresponds by its order type to a unique ordinal number $ord(<)$, and thus to a (possibly transfinite) enumeration $m_0, m_1, \ldots, m_\alpha, \ldots$ of the elements in M, where α is an ordinal number such that $0 \leq \alpha < ord(<)$; we use the notation $(M, <) = \{m_\alpha\}_<$

for this. Moreover, every well-ordering $<$ induces on every nonempty subset $M' \subseteq M$ a unique well-ordering $<'$. For a background on well-orderings, see e.g. [34].

Definition 4.2 *Let $\mathcal{P} = (P, <)$ be a prioritized logic program. Then, the ground instantiation of $\mathcal{P}$ is the prioritized rule base $\mathcal{P}^* = (P^*, <^*)$ where $<^*$ is the relation on P^* satisfying $r_1^* <^* r_2^*$ iff r_1^* and r_2^* are instances of rules r_1 and r_2 in P, respectively, such that $r_1 < r_2$.*

Note that in general, the relation $<^*$ in $(P^*, <^*)$ is not a partial ordering; this happens if the priorities $<$ are not consistent, in the sense that ground instances of rules may lead to a priority conflict. E.g., if we have rules $r_1 : p(x) \leftarrow q(x, a)$ and $r_2 : p(b) \leftarrow q(y, x)$ where $r_1 < r_2$, then the rule $r_3 : p(b) \leftarrow q(b, a)$ is a common instance of r_1 and r_2, which raises the contradicting priority $r_3 <^* r_3$. Such contradictions can be effectively recognized from $(P, <)$, however, and we thus may assume that $<^*$ is in fact a partial order. Moreover, it is easy to avoid such conflicts, by tagging rules with dummy literals such that common rule instances are not possible.

Definition 4.3 *A full prioritization of any ordered ground rule base $\mathcal{R} = (R, <)$ is any pair $\mathcal{R}' = (R, <')$ where $<'$ is a well-ordering on R compatible with $<$, i.e., $r_1 < r_2$ implies $r_1 <' r_2$, for all $r_1, r_2 \in R$. By $\mathcal{FP}(\mathcal{R})$ we denote the collection of all full prioritizations of $\mathcal{R}$. We say that $\mathcal{R}$ is fully prioritized, if $\mathcal{FP}(\mathcal{R}) = \{\mathcal{R}\}$, i.e., $\mathcal{R}$ coincides with its unique full prioritization.*

Every well-ordering $<'$ which refines $<$ is compatible with the priority specification, and if some answer set is acceptable under $<'$, it should be acceptable under $<$ as well; we thus will define the concept of preferred answer set under this credulous view of priorities.

4.2 Preferred answer sets

We will first formulate the criterion which an answer set must satisfy in order to be preferred, given a fully prioritized rule base $\mathcal{R} = (R, <)$.

Recall that in the definition of answer sets, the case of rules with weak negation is reduced to a particularly easy special case, namely to rules without weak negation. Similarly, there is a special case where it is particularly easy to check for an answer set A whether a full prioritization $(R, <)$ was taken into account adequately: ground rule bases R without prerequisites, i.e. collections where the bodies of all rules contain only weakly negated literals. In this case, we simply have to check whether each rule in R whose head is *not* in A is *defeated* by the consequences of applied rules of higher priority.

To model this, we associate with each fully prioritized rule base $\mathcal{R}$ of prerequisite-free ground rules an operator $C_{\mathcal{R}} : 2^{Lits} \rightarrow 2^{Lits}$. An answer set A satisfies the priorities, in case it is a fixpoint of $C_{\mathcal{R}}$. In the construction of $C_{\mathcal{R}}(A)$, rules are applied in the order of their priorities. In each step, the head of the current rule r is added to the collected literals, if (1) r is not defeated by literals collected so far, and (2) it is not the case that $head(r) \in A$ yet r is defeated in A. Note that condition (2) is necessary to avoid situations where a rule defeated in A nevertheless increases the priority of a different derivation of its head through another rule. The formal definition is as follows.

Definition 4.4 *Let $\mathcal{R} = (R, <)$ be a fully prioritized rule base of prerequisite-free ground rules, S a set of literals, and let $\mathcal{R} = \{r_\alpha\}$. We define the sequence S_α, $0 \le \alpha < ord(<)$, of sets $S_\alpha \subseteq Lits$ as follows:*

$S_\alpha =$

$\bigcup_{\beta<\alpha} S_\beta$ *if r_α is defeated by $\bigcup_{\beta<\alpha} S_\beta$ or $c \in S$ and r_α is defeated by S,*

$\bigcup_{\beta<\alpha} S_\beta \cup \{c\}$ *otherwise,*

where $c = head(r_\alpha)$. The set $C_{\mathcal{R}}(S)$ is the smallest set of ground literals such that

(i) $\bigcup_{\alpha<ord(<)} S_\alpha \subseteq C_{\mathcal{R}}(S)$, and

(ii) $C_{\mathcal{R}}(S)$ is logically closed.

Note that for $\alpha = 0$, $\bigcup_{\beta<\alpha} S_\beta = \emptyset$ holds. Moreover, for each successor ordinal $\alpha + 1$, the definition of $S_{\alpha+1}$ can be simplified by replacing $\bigcup_{\beta<\alpha+1} S_\beta$ with S_α; the above definition is uniform and more succinct, however. The sequence S_α monotonically increases and converges to $\bigcup_{\alpha<ord(<)} S_\alpha$.

$C_{\mathcal{R}}$ is not meant to return consequences of R. It may well be the case that a rule r_α is applied in the production of $C_{\mathcal{R}}(S)$ although the rule is later defeated by some less preferred rule r_β, i.e., $\alpha < \beta$. However, if some answer set A of R is a fixpoint of $C_{\mathcal{R}}$, then we can be sure that all preferences in $<$ were taken into account adequately, that is, a rule whose head is not in A is defeated by a more preferred rule applied in A. We therefore define preferred answer sets as follows:

Definition 4.5 *Let $\mathcal{R} = (R, <)$ be a fully prioritized rule base of prerequisite-free ground rules, and let A be an answer set of R. Then A is the preferred answer set of $\mathcal{R}$, if and only if $C_{\mathcal{R}}(A) = A$.*

Let us check that the use of the term "*the*" preferred answer set is justified in Definition 4.5.

Lemma 4.1 *Let $\mathcal{R} = (R, <)$ be a fully prioritized rule base of prerequisite-free ground rules, $\mathcal{R}$ possesses at most one preferred answer set.*

In the case of a rule base R of arbitrary ground clauses, we perform a reduction which can be viewed as dual to the Gelfond-Lifschitz reduction: given a set of ground literals A, we eliminate rules whose *prerequisites* are not in A and eliminate all prerequisites from the remaining rules. This yields a rule base of prerequisite-free ground rules $\mathcal{R}' = (R', <')$, together with an ordering $<'$ that is inherited from $\mathcal{R}$. We then can check whether a given answer set is a fixpoint of $C'_{\mathcal{R}}$.

Intuitively, our construction amounts to guessing provable prerequisites and checking whether the assumption that exactly these prerequisites hold is possible under the prioritized interpretation of rules.

Definition 4.6 *Let $(R, <) = \{r_\alpha\}_<$ be a fully prioritized set of ground rules, and let X be a set of ground literals. Let ${}^X\mathcal{R} = ({}^XR, {}^X<)$ be the fully prioritized set of ground rules such that ${}^XR$ is the set of rules obtained from R by*

1. *deleting every rule having a prerequisite ℓ such that $\ell \notin X$, and*
2. *removing from each remaining rule all prerequisites,*

and ${}^X<$ is inherited from $<$ by the map $f : {}^XR \longrightarrow R$, i.e., $r'_1 \; {}^X< r'_2$ iff $f(r'_1) < f(r'_2)$, where $f(r')$ is the $<$-least rule r_α in R such that r' results from r by step 2.

It is easily seen that ${}^X<$ is indeed a well-ordering on ${}^XR$. The definition of ${}^X<$ may look somewhat involved, but we have to respect a possible clash of rule priorities due to Step 2 of the reduction.

With these definitions, we are prepared for generalizing the notion of preferred answer sets to arbitrary programs:

Definition 4.7 *A set of ground literals A is a preferred answer set of a fully prioritized set of ground rules $\mathcal{R} = (R, <)$, if A is a preferred answer set of ${}^A\mathcal{R}$.*

A set of ground literals A is a preferred answer set of a prioritized logic program $\mathcal{P}$, if A is a preferred answer set for some $\mathcal{R} \in \mathcal{FP}(\mathcal{P}^*)$.

The collection of all preferred answer sets of $\mathcal{R}$ (resp. $\mathcal{P}$) is denoted by $\mathcal{PAS}(\mathcal{R})$ (resp. $\mathcal{PAS}(\mathcal{P})$).

Notation. For convenience, we introduce an operator $\lambda_{\mathcal{R}}$ as follows: $\lambda_{\mathcal{R}}(X) = C_{{}^X\mathcal{R}}(X)$. An answer set is then obviously preferred if and only if it is a fixpoint of $\lambda_{\mathcal{R}}$. In other words, A is a preferred answer set iff it is a fixpoint of Gelfond and Lifschitz's operator γ_R (see Definition 2.3) and a fixpoint of $\lambda_{\mathcal{R}}$.

Here is a first simple example. More examples will be discussed in the following sections. For simplicity, unless specified otherwise we will assume that the rules in each example are ordered by the numbering of the rules, i.e., rules with lower numbers are preferred over those with higher numbers.

Example 4.1 Let us consider the classical Tweety example, but impose some preferences on the rules as in Section 1:

$$\begin{array}{ll}
(1) & peng(tweety) \leftarrow \\
(2) & bird(tweety) \leftarrow \\
(3) & \neg flies(x) \leftarrow \mathrm{not}\; flies(x),\; peng(x) \\
(4) & flies(x) \leftarrow \mathrm{not}\; \neg flies(x),\; bird(x)
\end{array}$$

As mentioned above, P has two answer sets: $A_1 = \{peng(tweety),\; bird(tweety),\; \neg flies(tweety)\}$ and $A_2 = \{peng(tweety), bird(tweety), flies(tweety)\}$.

Notice that P^* contains besides (1) and (2) the rules (3) and (4) instantiated with $x = tweety$, and the inherited ordering $<^*$ on P^* is already a well-ordering. Thus, $\mathcal{P}^* = (P^*, <^*)$ is fully prioritized.

Let us first check whether A_1 is a preferred answer set. First, we have to determine the dual reduct ${}^{A_1}\mathcal{P}^*$. It contains the reduct of all four rules. Now, let us determine $\lambda_{\mathcal{P}^*}(A_1)$. It is not difficult to see that A_1 is indeed reconstructed in this process; consequently, the answer set A_1 is preferred.

On the other hand, let us compute $\lambda_{\mathcal{P}^*}(A_2)$. The dual reducts ${}^{A_2}\mathcal{P}^*$ and ${}^{A_1}\mathcal{P}^*$ coincide, and $\lambda_{\mathcal{P}^*}(A_2) = A_1$. Hence, A_2 is not preferred. ∎

5 Properties of Preferred Answer Sets

In this section, we show that the preferred answer set approach has several appealing properties, which make it particularly attractive.

First of all, we show that the problems with existing DL-approaches discussed in Section 3 are resolved in our approach.

Example 5.1 Consider the logic programming version of the example we used in Section 3.2 to demonstrate the violation of Principle I in the first group of DL-proposals:

$$\begin{array}{ll}
(1) & b \leftarrow \mathrm{not} \neg b, a \\
(2) & \neg b \leftarrow \mathrm{not}\; b \\
(3) & a \leftarrow \mathrm{not} \neg a
\end{array}$$

The program has two answer sets: $A_1 = \{a, b\}$ and $A_2 = \{a, \neg b\}$. As intended, A_2 is not preferred since $\lambda_{\mathcal{P}}(A_2) = A_1$. The single preferred answer set is A_1. ∎

Example 5.2 Let us turn to the problematic example for Rintanen's approach. Also this case is handled adequately:

$$\begin{array}{lr}(1) & b \leftarrow \mathrm{not}\neg b, a \\ (2) & \neg a \leftarrow \mathrm{not}\ a \\ (3) & a \leftarrow \mathrm{not}\neg a\end{array}$$

We obtain the two answer sets $A_1 = \{\neg a\}$ and $A_2 = \{a, b\}$. The answer set A_1 is preferred: ${}^{A_1}\mathcal{P}$ contains merely rules (2) and (3), and A_1 is fixpoint of $\lambda_{\mathcal{P}}$. On the other hand, A_2 is not preferred since $\lambda_{\mathcal{P}}(A_2) = \{b, \neg a\}$. This is exactly what we expect. ∎

The desired behavior of our approach on the problematic examples for the other approaches is not incidental. In fact, it satisfies the Principles I and II which we have introduced above, as we demonstrate next.

We start by showing that our approach captures a natural and intuitive idea about preferences. Recall that a rule r is applied in A exactly if $pre(r) \subseteq A$ and r is not defeated in A.

Proposition 5.1 *Let $\mathcal{R} = (R, <)$ be a fully prioritized rule base of ground rules, and let $A \in \mathcal{AS}(R)$. Then, A is a preferred answer set of $\mathcal{R}$, if and only if for each rule $r \in R$ such that $pre(r) \subseteq A$ and $head(r) \notin A$, there is a rule $r' \in R$ generating A such that $r' < r$ and $head(r')$ defeats r.*

We can also show that the behavior observed in Examples 5.1 and 5.2 is not incidental: our approach actually satisfies the desired principles introduced in Section 3.1.

Proposition 5.2 *The preferred answer set approach satisfies Principles I and II as described in Section 3.1.*

The following property, which is easily verified, confirms that our reduction of partial orders to full prioritizations works as intended.

Proposition 5.3 *Let $<_1$ and $<_2$ be partial orders on a program P such that $<_2$ refines $<_1$, i.e., $<_1 \subseteq <_2$. Then, $\mathcal{PAS}(P, <_2) \subseteq \mathcal{PAS}(P, <_1)$.*

Thus, as desired, removing priorities between rules does not affect a preferred answer set of a program and adding additional preference information can never lead to the loss of a conclusion.

Proposition 5.4 *Let R be a collection of ground rules, and let $A \in \mathcal{AS}(R)$ be an answer set. Then, there exists some well-ordering $<$ of R such that $A \in \mathcal{PAS}(\mathcal{R})$ where $\mathcal{R} = (R, <)$.*

An immediate consequence of this property is that prioritized logic programs are a conservative generalization of extended logic programs.

Corollary 5.5 *Let $\mathcal{P} = (P, <)$ without priorities, i.e., $<$ is empty. Then, $\mathcal{PAS}(\mathcal{P}) = \mathcal{AS}(\mathcal{P})$.*

Another property is that an inconsistent answer set is insensitive to any rule ordering.

Proposition 5.6 *Let $\mathcal{P} = (P, <)$ be a prioritized program such that $\mathcal{FP}(\mathcal{P}) \neq \emptyset$. If $A = Lits$ is an answer set of $\mathcal{P}$, then $Lits$ is the unique preferred answer set of $\mathcal{P}$.*

Preferred answer sets are not always unique, even for totally ordered programs.

Example 5.3 Consider the following program:

$$\begin{array}{lr}(1) & b \leftarrow \mathrm{not}\neg b, a \\ (2) & c \leftarrow \mathrm{not}\ b \\ (3) & a \leftarrow \mathrm{not}\ c\end{array}$$

This program has two answer sets: $A_1 = \{a, b\}$ and $A_2 = \{c\}$. Both are preferred. ∎

On the other hand, it is not difficult to see that there are also programs that have answer sets but for which no preferred answer set exists.

Example 5.4 Consider the following program:

$$\begin{array}{lr}(1) & c \leftarrow \mathrm{not}\ b \\ (2) & b \leftarrow \mathrm{not}\ a\end{array}$$

The single answer set of this program is $A = \{b\}$. However, $C_{^A\mathcal{P}}(A) = \{c, b\}$ and thus A is not preferred. ∎

Two views seem possible here: one might say that this is exactly what is expected in a situation like this; the priorities cannot be satisfied and thus the prioritized program should be considered inconsistent. This corresponds to a rather strict interpretation of the meaning of priorities.

However, one might also argue that the preference information should help us in selecting an answer set from all existing answer sets, but never render an otherwise consistent program inconsistent. In the next section, we show how this view of relaxing priority information can be implemented in our approach.

6 Weakly Preferred Answer Sets

We now propose a relaxation of our approach that gives us exactly the preferred answer sets whenever they exist and some approximation, called *weakly preferred answer sets*, in the other cases. Of course, we want weakly preferred answer sets to be answer sets. Note that these requirements are incompatible with our two principles as simple examples demonstrate (to see that Principle II is incompatible add the rule $a \leftarrow c$ with highest priority to the program of Example 5.4; an example demonstrating violation of Principle I is contained in the full paper). We trade in this section satisfaction of our principles for consistency of the semantics.

The basic idea is to consider in the comparison of answer sets the degree to which preferences are violated. Intuitively, the degree depends on how much of the current priority information has to be changed in order to obtain a preferred answer set. To measure this appropriately, we need a suitable notion of distance between two well-orderings of the same set.

With an eye on our application, we may consider the distance from a well-ordering $<_1$ of a set M to another well-ordering $<_2$ of M as the number of pairs m, m' such that $m <_1 m'$ but $m' <_2 m$, i.e., the number of violations (or *inversions*) of the ordering $<_2$ in $<_1$. For example, consider $M = \{a, b, c, d\}$ and the two orderings $c <_1 a <_1 d <_1 b$ and $a <_2 b <_2 c <_2 d$; then the pairs (c, a), (c, b), and (d, b) are the inversions of $<_2$ in $<_1$. What we want to do is basically count these inversions. This leads us to the following definition.

Definition 6.1 *Let $<_1$ and $<_2$ be well-orderings of the set M with corresponding enumerations $(M, <_1) = \{r_\alpha\}_{<_1}$ and $(M, <_2) = \{s_\beta\}_{<_2}$, respectively. Then, denote by $Inv(<_1, <_2)$ the set of all inversions of $<_2$ in $<_1$, and let for $\beta < ord(<_2)$ denote $V_\beta(<_1, <_2)$ the well-ordering induced by $<_1$ on the set*

$$\{r_\alpha \mid r_\alpha <_1 s_\beta, s_\beta <_2 r_\alpha\} \subseteq M.$$

Then, the distance from $<_1$ to $<_2$, denoted $d(<_1, <_2)$, is defined as the ordinal

$$d(<_1, <_2) = \sum_{\beta < ord(<_2)} ord(V_\beta(<_1, <_2,)).$$

The definition of $d(<_1, <_2)$ involves ordinal arithmetic, which may be less familiar to the reader; see e.g. [34] for a detailed treatment. Informally, addition $\alpha + \beta$ of ordinals amounts to appending an ordering of type β to an ordering of type α. Note that ordinal arithmetic generalizes familiar integer arithmetic to the infinite, and has there different properties. E.g., $1 + \omega = \omega$, which is different from $\omega + 1$; hence, addition of ordinals is not commutative in general.

The value of $d(<_1, <_2)$ is indeed well-defined, as every element of the infinite sum is an ordinal and the sequence is well-founded; for such sums, the result is an ordinal [34]. Intuitively, $ord(V_\beta(<_1, <_2))$ is the number of violations of the priority of the element s_β which is at position β in $<_2$, and $d(<_1, <_2)$ counts in a way all these violations. Counting them is easy if the underlying set M is finite, while it is not in the infinite case. The problem is that in the latter case, the order in which a collection of ordinals is summed matters in general; the result may even be undefined. Our selection in the definition of $d(<_1, <_2)$ is driven by our application, and sums the violations in order of decreasing position with respect to the violated ordering $<_2$.

Clearly, $d(<, <) = 0$ holds for any well-ordering $<$, while symmetry does not hold in general. If we wish, we can make the distance symmetric by choosing the smaller of $d(<_1, <_2)$ and $d(<_2, <_1)$. Or, we cast $d(<, <')$ to its associated cardinal; this is sufficient to make the distance symmetric for our application. In particular, for finite distance values, symmetry is automatically given.

Proposition 6.1 *Let $<_1, <_2$ be well-orderings of the same set M such that $d(<_1, <_2) < \omega$ is finite. Then, d is symmetric on $(<_1, <_2)$, i.e., $d(<_2, <_1) = d(<_1, <_2)$.*

Corollary 6.2 *On the full prioritizations of a ground rule base, the distance measure $d'(<_1, <_2) = card(d(<_1, <_2))$, where $card(\alpha)$ is the cardinal associated with the ordinal α, is symmetric.*

Note that using d' is particularly useful, if we want to avoid a fine-grained distinction of priority violation and we are only interested whether a finite or infinite number of priority violations is present; $d'(<_1, <_2)$ is either finite or $\aleph_0$, the smallest uncountable cardinal.

It can be checked that d verifies all axioms of a standard metric in the finite case, i.e., $d(<, <) = 0$, $d(<_1, <_2) = d(<_2, <_1)$, and $d(<_1, <_3) \leq d(<_1, <_2) + d(<_2, <_3)$.

With the above notion of distance from one well-ordering to another, we can weaken the definition of preferred answer sets as follows.

Definition 6.2 *Let $\mathcal{R} = (R, <)$ be a partially ordered ground rule base, and let $A \in \mathcal{AS}(R)$. The preference violation degree of A in $\mathcal{R}$, denoted $pvd_{\mathcal{R}}(A)$, is the minimum distance possible from any full prioritization of $\mathcal{R}$ to any fully prioritized rule base $\mathcal{R}' = (R, <')$ such that A is a preferred answer set of $\mathcal{R}'$.*

Moreover, let $pvd(\mathcal{R}) = \min_{A \in \mathcal{AS}(R)} pvd_{\mathcal{R}}(A)$ be the preference violation degree of $\mathcal{R}$.

For any answer set A of a prioritized program $\mathcal{P} = (P, <)$, the preference violation degree of A in $\mathcal{P}$, denoted $pvd_{\mathcal{P}}(A)$, is defined as $pvd_{\mathcal{P}}(A) = pvd_{\mathcal{P}^*}(A)$, and the preference violation degree of $\mathcal{P}$, denoted $pvd(\mathcal{P})$, is $pvd(\mathcal{P}) = pvd(\mathcal{P}^*)$.

Note that $pvd_{\mathcal{R}}(A)$ is indeed well-defined. This is a consequence of Proposition 5.4, which says that for every answer set A there exists a total ordering of the rules such that A is preferred.

Based on the definition of preference violation degree, we formalize the concept of weakly preferred answer sets as follows:

Definition 6.3 *Let $\mathcal{P} = (P, <)$ be a prioritized logic program. Then, an answer set A of P is a weakly preferred answer set of $\mathcal{P}$ if and only if $pvd_{\mathcal{P}}(A) = pvd(\mathcal{P})$. By $\mathcal{WAS}(\mathcal{P})$ we denote the collection of all weakly preferred answer sets of $\mathcal{P}$.*

The following results are easily verified.

Proposition 6.3 *Let $\mathcal{P} = (P, <)$ be a prioritized logic program. If $\mathcal{PAS}(\mathcal{P}) \neq \emptyset$, then $\mathcal{PAS}(\mathcal{P}) = \mathcal{WAS}(\mathcal{P})$.*

Preferred answer sets are just those with preference violation degree zero.

Proposition 6.4 *Let $\mathcal{P} = (P, <)$ be a prioritized logic program. If $\mathcal{AS}(\mathcal{P}) \neq \emptyset$, then $\mathcal{WAS}(\mathcal{P}) \neq \emptyset$.*

In particular, whenever P has a unique answer set, this set will be weakly preferred.

7 Computational Complexity

In this section, we address the issue of computing preferred and weakly preferred answer sets, and analyze the complexity of the main computational tasks in this context.

We assume that the reader is familiar with the basic concepts of complexity theory; [18, 26] are good sources. For a background on complexity results in logic programming, refer to [33, 10, 8]. In our analysis, we focus on the case of finite propositional (ground) prioritized programs.

Due to space limitations we can only list the major results without proofs and thorough discussion. These can be found in the full paper.

Of course, since extended logic programs without priorities are a special case of prioritized programs, all lower complexity bounds for answer sets of extended logic programs carry over to preferred answer sets. In particular, by the complexity results on the stable models of a general propositional logic program P [22, 23, 4] and the obvious correspondence between the preferred answer sets of the prioritized program $\mathcal{P} = (P, <)$ where $<$ is empty (cf. Corollary 5.5) and the stable models of P, we obtain that deciding whether a prioritized program has a preferred answer set is NP-hard, and, moreover, that inference of a literal from all preferred answer sets of a prioritized program is a co-NP-hard problem.

Natural questions are now whether preferred answer sets are harder than answer sets in general and, moreover, how the complexity is affected if we have programs with a particular priority ordering. We have proven the following theorem:

Theorem 7.1 *Let $\mathcal{P} = (P, <)$ be a prioritized finite logic program, and let $A \subseteq Lits$. Then, deciding whether $A \in \mathcal{PAS}(\mathcal{P})$ is polynomial (more precisely, complete for* P*).*

Indeed, for a full prioritized $\mathcal{P}$, it is easily seen that the test is polynomial. We have to check whether A is an answer set of P, which is well-known polynomial, and that $\lambda_{\mathcal{P}}(A) = A$, where $\lambda_{\mathcal{P}}(A) = C_{{}^A\mathcal{P}}(A)$. The dual reduct ${}^A\mathcal{P}$ is clearly computable in polynomial time, as is, given ${}^A\mathcal{P}$ and A, the set $C_{{}^A\mathcal{P}}(A)$. For an arbitrary $\mathcal{P}$, it suffices to consider a refinement of $<$ to some full prioritization $<'$ in which informally all rules that are not applied in A and possibly harm the reconstruction of A are ordered to the rear (discussed in detail in the full paper).

However, for the consistency problem, we have the following result.

Theorem 7.2 *Given a finite propositional prioritized program $\mathcal{P} = (P, <)$, deciding whether $\mathcal{PAS}(\mathcal{P}) \neq \emptyset$ is* NP-*complete. The problem is* NP-*hard even if $<$ is a total order and P does not involve strong negation.*

A literal L is a *consequence* of a prioritized program $\mathcal{P}$, denoted $\mathcal{P} \models L$, if L belongs to every preferred answer set of $\mathcal{P}$. The set of all consequences of $\mathcal{P}$ is denoted by $Cn(\mathcal{P})$.

Theorem 7.3 *Given a finite propositional prioritized program $\mathcal{P} = (P, <)$ and a literal L, deciding whether $\mathcal{P} \models L$ is* co-NP-*complete. The problem is* co-NP-*hard even if $<$ is a total order and P does not involve strong negation.*

As discussed in Section 6, for a fully prioritized program $\mathcal{P} = (P, <)$ the value $pvd_{\mathcal{P}}(A)$ amounts to the smallest number of switches of neighbored rules in $<$ such that A is a preferred answer set of the resulting program $\mathcal{P}' = (P, <')$; Moreover, we know from Proposition 6.4 that such a switching is always possible.

For the smallest number of switches, $|P|^2$ is an obvious upper bound. This bound can be tightened to

$$s = |P|-1+|P|-2+\ldots+1 = |P|(|P|-1)/2 = O(|P|^2),$$

which is the maximal number of switches performed by bubble sort when it sorts the input ordered by $<$ into $<'$.

The problem of computing $pvd_{\mathcal{P}}(A)$ can be rephrased as deciding whether $pvd_{\mathcal{P}}(A)$ is smaller than a given bound k. If a procedure for solving this problem is available, then we can actually compute $pvd_{\mathcal{P}}(A)$ in a binary search over $[0, s]$. As it turns out, deciding whether $pvd_{\mathcal{P}}(A) \leq k$ is a problem in NP. Thus, we can compute $pvd_{\mathcal{P}}(A)$ by making $O(\log |P|^2)$ many calls to an NP oracle.

The value of $pvd(\mathcal{P})$ can be computed similarly, if a subroutine solves the problem whether $pvd(\mathcal{P}) \leq k$ holds; also the latter problem is in NP. We thus obtain the following result.

Proposition 7.4 *Given a finite propositional prioritized program $\mathcal{P} = (P, <)$ and $A \in \mathcal{AS}(P)$, computing $pvd_{\mathcal{P}}(A)$ is in $\mathrm{F}\Delta_2^p[O(\log n)]$, i.e., possible in polynomial time with $O(\log n)$ many oracle queries, where n is the input size. Moreover, also computing $pvd(\mathcal{P})$ is in $\mathrm{F}\Delta_2^p[O(\log n)]$.*

We say that a literal L is a *weak consequence* of a prioritized program $\mathcal{P}$, denoted $\mathcal{P} \models_w L$, if L belongs to every weakly preferred answer set of $\mathcal{P}$. We denote by $Cn_w(\mathcal{P})$ the set of all weak consequences of $\mathcal{P}$. We have proven the following result.

Theorem 7.5 *Deciding, given a finite prioritized propositional program $\mathcal{P} = (P, <)$ and a literal L, whether $\mathcal{P} \models_w L$ is $\Delta_2^p[O(\log n)]$-complete. Hardness for $\Delta_2^p[O(\log n)]$ holds even if $<$ is a total order and P does not have strong negation.*

The inference problem $\mathcal{P} \models_w L$ can be solved as follows. First, compute $pvd(\mathcal{P})$ and then query an NP oracle whether some answer set A with $pvd_{\mathcal{P}}(A) \leq pvd(\mathcal{P})$ exists such that $L \notin A$. It can be seen that the oracle is in NP. Hence, a logarithmic number of NP oracle calls suffices, and the problem is in $\Delta_2^p[O(\log n)]$.

In the full paper, we also address the complexity of computing an arbitrary preferred (resp. weakly preferred) answer set.

8 Related Work and Conclusion

Apart from the approaches in the context of default logic which we have reviewed in Section 3, several approaches treating preferences in the context of logic programming have been described in the literature.

Kowalski and Sadri. In [20], Kowalski and Sadri have proposed to consider rules with negation in the head as exceptions to more general rules and to give them higher priority. The main achievement is that programs whose single answer set is inconsistent become consistent in the new semantics. The approach can hardly be viewed as a satisfactory treatment of preferences for several reasons: (1) Preferences are implicit and highly restricted; the asymmetric treatment of positive and negative information in this context seems unjustified from a knowledge representation perspective. (2) Fewer conclusions are obtained than in the original answer set semantics, contrary to what one would expect when preferences are taken into account.

It is, therefore, more reasonable to view Kowalski and Sadri's approach as a contribution to inconsistency handling rather than preference handling.

Pradhan and Minker. In [28] Pradhan and Minker show how priorities can be used to combine different potentially conflicting databases. Preference information is used to determine the information which has to be given up in the merging process.

There are two major differences between this and our work: (1) Pradhan and Minker consider Datalog databases only, that is, neither explicit negation nor negation as failure is admitted. Our approach is thus more general. (2) The underlying preference relation is a relation on atoms, not on rules as in our case. While this appears to be adequate for merging Datalog databases we strongly believe that the more fine grained distinctions which are possible in our approach are necessary for many knowledge representation problems.

Buccafurri et al. An approach that is closer in spirit to ours is ordered logic programming [7]. An ordered logic program is a set of components forming an inheritance hierarchy. Each component consists of a set of rules. Rules lower in the hierarchy have preference over those higher up in the hierarchy since the former are considered more specific. There are two main differences between ordered logic programs and our approach: (1) The preferences of ordered logic programs are predefined through the inheritance hierarchy. (2) Ordered logic programs use only one kind of negation, the distinction between negation as failure and classical negation is not expressible in the language.

Brewka. In the article [6], one of the authors of the present paper has described a prioritized version of the well-founded semantics for extended logic programs. In that paper, the preference information was expressed in the logical language and could thus be derived dynamically. Here we used an additional ordering relation outside the language. However, the representation in the language is not much more difficult and can be done along the lines of [5].

Zhang and Foo. In two subsequent papers [38, 37], Zhang and Foo have presented a framework for prioritized logic programs which at the syntactical level is very close to ours. Also in that framework, priorities are expressed by a strict partial ordering on the rules. However, the similarity is at the surface, since the approach to the semantics followed by Zhang and Foo is fundamentally different. Their semantics is operationally defined, in terms of a (nondeterministic) reduction of the initial prioritized program $\mathcal{P}$, which is reduced to an extended logic program $\mathcal{P}(P)$, whose answer sets are preferred answer sets of $\mathcal{P}$; the preferred answer sets of $\mathcal{P}$ are the answer sets of all programs P to which $\mathcal{P}$ may be (nondeterministically) reduced.

In general, Zhang and Foo's concept of preferred answer set is different from ours. Indeed, it is easy to see from the definitions that if the program without priorities has an answer set, then also the prioritized program has a preferred answer set. In this respect Zhang and Foo's notion is closer to our weakly preferred answer sets than to preferred answer sets. However, their concept does not coincide with weakly preferred answer sets either. Several examples illustrating the different behavior are contained in the full paper.

Gelfond and Son. In a recent paper, Gelfond and Son have also tackled the problem of adding priorities on defaults in the language of extended logic programs [17]. Their approach admits the specification of preference over rules in the object language. Rules are divided into definite rules and default (defeasible) rules. While definite rules must be strictly obeyed, default rules may be ignored if reasonable in a given context. The semantics of the language is defined in terms of a transformation of any program $\mathcal{P}$ into an extended logic program $t(\mathcal{P})$. The transformation is basically a meta-interpreter for programs.

Gelfond and Son point out that their approach is different from ours, and that it leads in general to different results. In particular, an answer set of a prioritized program may be generated which does not correspond to an answer set of the original, non-prioritized program. This cannot happen in our approach.

Delgrande and Schaub. In [9] Delgrande and Schaub have presented another approach for adding priorities to default theories. A set of default rules D, a background classical theory W, and a preference ordering $<$ (which is a strict partial ordering) on the default rules are transformed into a standard default theory $T = (D', W')$, by naming the defaults and introducing special purpose predicates.

Schaub and Delgrande's approach is different from ours. This can be illustrated by the default theory in Example 3.1 from Section 3.2 where Schaub and Delgrande select the extension $E_2 = Th(\{a, \neg b\})$, which shows that Principle I is violated.

Several issues remain for further work. One such issue is the extension to a syntactically and semantically richer framework,e.g., rules with disjunction in the heads [30, 15]. Another possible extension are constructs for expressing dynamic preferences in the object language. As we have briefly mentioned above, there are no principal obstacles to extending our approach in this direction.

Finally, we plan an experimental implementation of preferred and weakly preferred answer sets on top of the deductive reasoning system `dlv` [12].

Acknowledgements. The authors thank M. Gelfond, H. Herre and N. Leone for helpful and clarifying discussions on the topic of the paper. Furthermore, we are grateful to V. Marek for bibliographic hints on well-orderings, to H. Blair and A. Nerode for discussions on this subject.

References

[1] F. Baader and B. Hollunder. Priorities on Defaults with Prerequisite and their Application in Treating Specificity in Terminological Default Logic. *Journal of Automated Reasoning*, 15:41–68, 1995.

[2] C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[3] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency Management and Prioritized Syntax-Based Entailment. In R. Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 640–645. Morgan Kaufman, 1993.

[4] N. Bidoit and C. Froidevaux. General Logic Databases and Programs: Default Semantics and Stratification. *Information and Computation*, 19:15–54, 1991.

[5] G. Brewka. Adding Priorities and Specificity to Default Logic. In *Proceedings JELIA '94*, LNAI 838, pages 247–260, 1994.

[6] G. Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, pages 19–36, 1996.

[7] F. Buccafurri, N. Leone, and P. Rullo. Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *Journal of Logic Programming*, 27(1):5–43, 1996.

[8] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. In *Proceedings of the Twelth IEEE International Conference on Computational Complexity (CCC '97)*, pages 82–101, 1997.

[9] J. Delgrande and T. Schaub. Compiling Reasoning With and About Preferences into Default Logic. In *Proceedings IJCAI '97*, pages 168–174, 1997.

[10] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[11] T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *Journal of the ACM*, 42(1):3–42, January 1995.

[12] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-Monotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97)*, number 1265 in LNCS, pages 364–375. Springer, 1997.

[13] M. Gelfond. Logic Programming and Reasoning with Incomplete Information. *Annals of Mathematics and Artificial Intelligence*, 12:89–116, 1994.

[14] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[15] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[16] M. Gelfond, H. Przymusinska, and T. Przymusinski. On the Relationship Between Circumscription and Negation as Failure. *Artificial Intelligence*, 38:75–94, 1989.

[17] M. Gelfond and R. Son. Reasoning with Prioritized Defaults. Manuscript, 1997.

[18] D. S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Publishers B.V. (North-Holland), 1990.

[19] K. Konolige. Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning. In *Proceedings AAAI '88*, 1988.

[20] R. Kowalski and F.Sadri. Logic Programs with Exceptions. *New Generation Computing*, 9:387–400, 1991.

[21] J. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1984, 1987.

[22] W. Marek and M. Truszczyński. Autoepistemic Logic. *Journal of the ACM*, 38(3):588–619, 1991.

[23] W. Marek and M. Truszczyński. Computing Intersection of Autoepistemic Expansions. In A. Nerode, W. Marek, and V. Subrahmanian, editors, *Proceedings of the 1st Intl. Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-91)*, pages 37–50, Washington DC, July 1991. MIT Press.

[24] W. Marek and M. Truszczyński. *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer, 1993.

[25] B. Nebel. How Hard is it to Revise a Belief Base ? Technical Report TR No. 83, Inst. für Informatik, Albert-Ludwigs-Universität Freiburg, August 1996. To appear in: *Handbook on Defeasible Reasoning and Uncertainty Management Systems. Volume 2: Belief Change*, D. Dubois and H. Prade (eds).

[26] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[27] D. Poole. On the Comparison of Theories: Preferring the Most Specific Explanation. In *Proceedings IJCAI '85*, 1985.

[28] S. Pradhan and J. Minker. Combining Datalog Databases Using Priorities. *International Journal of Cooperative Intelligent Systems*, 1996.

[29] H. Prakken. *Logical Tools for Modelling Legal Argument*. Dissertation, Vrije Universiteit Amsterdam, 1993.

[30] T. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[31] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[32] J. Rintanen. On Specificity in Default Logic. In C. Mellish, editor, *Proceedings IJCAI '95*, pages 1474–1479. AAAI Press, 1995.

[33] J. Schlipf. Complexity and Undecidability Results in Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):257–288, 1995.

[34] W. Sierpiński. *Cardinal and Ordinal Numbers*. Monografie Matematyczne tom. 34. Państwowe Wydawnictwo Naukowe, Warsaw, 1958.

[35] D. Touretzky. *The Mathematics of Inheritance*. Pitman Research Notes in Artificial Intelligence, London, 1986.

[36] D. Touretzky, R. Thomason, and J. Horty. A Skeptic's Menagerie: Conflictors, Preemptors, Reinstaters, and Zombies in Nonmonotonic Inheritance. In *Proceedings IJCAI '91*, 1991.

[37] Y. Zhang and N. Foo. Answer Sets for Prioritized Logic Programs. In *Proceedings ILPS 97*, pages 69–83, 1997.

[38] Y. Zhang and N. Foo. Towards Generalized Rule-Based Updates. In *Proceedings IJCAI '97*, pages 82–87, 1997.