Preferred Answer Sets for Extended Logic Programs*

Gerhard Brewka[†] Thomas Eiter[‡]

March 3, 1999

Abstract

In this paper, we address the issue of how Gelfond and Lifschitz's answer set semantics for extended logic programs can be suitably modified to handle prioritized 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, while the second guarantees the existence of a preferred answer set for programs possessing at least one answer set.

Adding priorities to rules is not new, and has been explored in different contexts. However, we show that many approaches to priority handling, most of which are inherited from closely related formalisms like default logic, are not suitable and fail on intuitive examples. Our approach, which obeys abstract, general principles that any approach to prioritized knowledge representation should satisfy, handles them in the expected way. Moreover, we investigate the complexity of our approach. It appears 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 [54, 67, 68, 34]. 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 [57]. 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 [31], hierarchic autoepistemic logic [37], prioritized default logic [48, 8, 3], prioritized theory revision [6, 49], or prioritized abduction [23]. 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, and to a less extent as a tool for

* A short and partial version of this paper appeared in: Proceedings 6th International Conference on Principles of Knowledge Representation and Reasoning (KR '98), Trento, Italy, June 2–4, A.G. Cohn, L. Schubert and S.C. Shapiro (eds), pages 86–97, 1998.

† Universität Leipzig, Institut für Informatik, Abteilung Intelligente Systeme, Augustusplatz 10-11, D-04109 Leipzig, Germany. Email: brewka@informatik.uni-leipzig.de

‡ Institut und Ludwig Wittgenstein Labor für Informationssysteme, Knowledge-Based Systems Group, Technische Universität Wien, Treitlstraße 3, A-1040 Wien, Austria. Email: eiter@kr.tuwien.ac.at

knowledge representation and reasoning. However, in the recent past, it has become evident that logic programming can serve as a powerful framework for knowledge representation, cf. [28, 4]. If logic programming wants to successfully stand this challenge, it must provide the features which have been recognized as indispensable in the context of knowledge representation. One such feature is the possibility to handle specificity and priority of knowledge.

This motivates an investigation of the possibilities to enhance logic programs with priorities. In a previous paper [9], one of the authors has defined a prioritized version of well-founded semantics for extended logic programs. In the present paper, we investigate prioritized programs under answer set semantics [30], which is regarded as the second standard semantics for extended logic programs. Notice, however, that the approach in the present paper is not a simple extension of the previous approach: it is based on different philosophical grounds.

Let us first characterize somewhat more precisely what we want to achieve. We consider programs 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*. The definition of preferred answer sets is thus one of the main contributions of this paper.

To give the flavor of our approach, we consider a simple motivating example. The following logic program represents the classical birds & penguins example:

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

This program has two answers sets: $A_1 = \{peng(tweety),\, bird(tweety),\, \neg flies(tweety)\}$ and $A_2 = \{peng(tweety),\, bird(tweety),\, flies(tweety)\}$ (for precise definitions, see Section 2).

Now let us assume that the rule numbers express priorities, such that rule (1) has the highest priority and rule (4) the lowest. Then, the answer set A_2 is no longer intuitive: indeed, $flies(tweety)$ is concluded by applying rule (4); on the other hand, the rule (3) has higher priority than (4), and thus $\neg flies(tweety)$ should be concluded. Our approach handles this example as desired, as A_1 is the preferred answer set.

As mentioned before, a number of prioritized versions of default logic [60] have been defined. It is well-known that extended logic programs can be translated to default theories in a straightforward manner, cf. [30, 48], and that the answer sets of a program are in one-to-one correspondence with the extensions of the respective default theory. One may thus conclude that the existing proposals for prioritized default logics, e.g. [48, 8, 3, 61, 18], are sufficient to specify preferred answer sets. However, it turns out that all these approaches are unsatisfactory. This will be demonstrated in Section 3 and Section 9. The intuitive reason for the failure of these approaches is that some of them implicitly recast Reiter's default logic to a logic of graded beliefs, while others overly enforce the application of rules with high priority, which leads to counterintuitive behavior.

Our approach takes a different perspective, which is dominated by 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 the other approaches. Our formalization of these ideas involves a dual of the standard Gelfond-Lifschitz reduction and a certain operator used to check satisfaction of priorities.

At this point the reader might ask a methodological question: how can we be sure that the approach presented here, even if it performs better than existing approaches on several examples, will not fail on future examples? Is what we propose an ad hoc fix that will need further fixing after a short period of time in the light of new examples?

This is certainly a concern which needs to be taken very seriously. In order to base our approach on firmer ground, we set forth some abstract principles that, as we believe, any formalization of prioritized logic programs should satisfy, and in a wider context related formalisms for knowledge representation as well. We demonstrate that our approach satisfies these principles, while other approaches fail on them. The investigation of abstract principles for priorities in knowledge representation formalisms is to the best of our knowledge novel and constitutes another contribution of this paper. We certainly do not claim our principles to be complete in any reasonable sense – an exhaustive investigation of this issue is a comprehensive task of its own, comparable to analogous studies in the field of nonmonotonic reasoning [1, 19, 20, 39, 36, 43]. Nevertheless, we believe that considering such abstract principles as a coherence check is an important step into the right direction, leading to a more effective and focused development of prioritized frameworks.

The remainder of this 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. As it appears, preferred answer sets are in some contexts too strict, and a program which has an answer set may lack having a preferred one. In order to handle this problem, 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 the issue of computing preferred and weakly preferred answer sets. We describe algorithms and analyze the complexity of our approach, where we focus on the propositional case. It appears that strong preference does not add to the complexity of answer sets in the most important reasoning tasks, and that weak preference leads only to a mild increase of complexity. In particular, all considered reasoning problems are in the polynomial time closure of NP. Section 8 demonstrates how the preferred answer set approach can be applied to problems in qualitative decision making. Section 9 discusses related work, and Section 10 concludes the paper by considering possible extensions and outlining further work.

In order to increase readability and not to distract from the flow of reading, proofs of technical results have been moved to the appendix, with the exception of short proofs and those which, as we think, should be seen together with the result.

2 Preliminaries and Notation

We assume that the reader is familiar with the basic conceptsto of logic programming; see [42] for background. In the present paper, we focus on extended logic programs as in [30], which have two kinds of negation.

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

$$c \leftarrow a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m \tag{1}$$

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.[1] 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 [30]. Answer sets are defined in analogy to stable models [29], 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 [29].

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 [30].

We illustrate the definitions using the birds & penguins example from above.

Example 2.1 Consider the program P containing the following rules:

$$\begin{array}{rr} peng(tweety) \leftarrow & \neg flies(x) \leftarrow \mathrm{not}\ flies(x),\ peng(x) \\ bird(tweety) \leftarrow & flies(x) \leftarrow \mathrm{not}\ \neg flies(x),\ bird(x) \end{array}$$

This program has two answer sets, namely $A_1 = \{peng(tweety),\ bird(tweety),\ \neg flies(tweety)\}$ and $A_2 = \{peng(tweety),\ bird(tweety),\ flies(tweety)\}$. Indeed, the A_1-reduct of P^* is the program:

$$peng(tweety) \leftarrow$$

[1] Like other authors, we reserve the term logic program for a finite collection of rules. This conforms with the view that a program should be a finite object. Other authors consider logic programs that are (restricted) recursive sets of rules, cf. [45]. The term "program" seems inappropriate for any non-recursive collection of clauses.

$$
\begin{aligned}
bird(tweety) &\leftarrow \\
\neg flies(tweety) &\leftarrow peng(tweety)
\end{aligned}
$$

Applying $Cn(\cdot)$ to this program yields $Cn(P^{*A_1}) = \{peng(tweety), bird(tweety), \neg flies(tweety)\} = A_1$. Thus, $\gamma_{P^*}(A_1) = A_1$, which means that A_1 is an answer set of P. Similarly, the A_2-reduct of P^* is the program:

$$
\begin{aligned}
peng(tweety) &\leftarrow \\
bird(tweety) &\leftarrow \\
flies(tweety) &\leftarrow bird(tweety)
\end{aligned}
$$

Applying $Cn(\cdot)$ to this program yields $\{peng(tweety), bird(tweety), flies(tweety)\}$. Thus, $\gamma_{P^*}(A_2) = A_2$, and hence A_2 is an answer set of P. Clearly no further answer set exists.

Therefore, neither $P \models flies(tweety)$ nor $P \models \neg flies(tweety)$ holds; both literals $flies(tweety)$ and $\neg flies(tweety)$ are unknown. ∎

3 Problems with Existing DL-Approaches

Different prioritized versions of Reiter's default logic [60] have been proposed in the literature, e.g. [48, 8, 3, 61, 18]. 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 be *generating* with respect to 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 $(T, <)$ and 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 most of the existing treatments of preferences for default logic, described in [48, 8, 3, 61], violate one of these principles. As mentioned in the introduction, we think that any formalization of prioritized logic programs should satisfy the above principles. Even if the reader does not subscribe to this view, the above and similar principles are still of interest as they may be used for classifying different patterns of reasoning.

3.2 Control of Reiter's quasi-inductive definition

The first group of proposals [48, 8, 3] uses preferences to control the quasi-inductive definition of extensions [60]: in each step of the generation of extensions the defaults with highest priority whose prerequisites have already been derived are applied. Now what is wrong with this idea? The answer is: 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 can lead to situations where Principle I is violated.

The mentioned approaches differ in technical detail. We do not 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:[2]

(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. ∎

This unpleasant behavior of selecting E_2 in the previous example was already observed in [8]. In that paper, the author tried to defend his approach arguing that there is only weak evidence for the literal a in our example. We revise our view, however, and do not support this argument any longer. After all, default logic is not a logic of graded belief where degrees of evidence should play a role. Default logic models acceptance of belief based on defeasible arguments. Since a is an accepted belief, rule (1) should be applied and E_1 should be the preferred extension in the example.

3.3 Rintanen's approach

An entirely different approach was proposed by Rintanen in [61] for normal default logic, which has been extended to full default logic in [62]. Rintanen uses a total order on defaults to induce a lexicographic order on extensions.

Call a default rule $r = \alpha{:}\beta_1, \ldots, \beta_n/\gamma$ *applied* in a set of formulas E (denoted $appl(r, E)$), if $E \models \alpha$ and $\{\neg\beta_1, \ldots, \neg\beta_n\} \cap E = \emptyset$ holds. Denote by $appl(r, E, E')$ that $appl(r, E)$ is true and $appl(r, E')$ is false.

Then, an extension E of the default theory $\Delta = (W, D)$ is a preferred extension with respect to a partial ordering $<$ on D, if there exists a strict total ordering $<'$ compatible with $<$, such that for all extensions E' of Δ and default rules $r \in \Delta$, it holds that $appl(r, E', E)$ is true only if there is some default rule $r' \in D$ such that $r' <' r$ and $appl(r', E, E')$ is true. In other words, E is preferred if we can arrange the rules in a total

[2] We assume that the reader knows the basic to of standard default logic [60]; informally, a default rule $a{:}b/c$ corresponds to the clause $c \leftarrow \mathrm{not}\neg b, a$.

ordering such that any rule r applied in an arbitary extension E' is either applicable in E, or it is preceded by a rule r' that is applied in E but not in 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.

According to Rintanen's approach, however, E_2 is the (unique) preferred extension. 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.

∎

In [62], Rintanen presents also a variant of the approach described above, which is weaker, i.e. admits more preferred extensions. An extension is a weakly preferred extension of $\Delta = (D, W)$ with respect to $<$, if for every extension E' of Δ there exists a strict total ordering $<'$ compatible with $<$ such that for every default rule $r \in D$, $appl(r, E', E)$ is true only if some $r' \in D$ exists such that $r' <' r$ and $appl(r', E, E')$ holds. As easily seen, every preferred extension is weakly preferred; the converse is false, however. On the previous example, this variant behaves as the first approach, and is thus also not satisfactory. Further approaches to prioritized default logic are defined in [62, p. 227], where defeated default rules are used instead of applied rules in the definition of preferred extension. However, these approaches violate the principles described above.

Since all these approaches suffer from drawbacks, they cannot serve as a basis for a satisfactory definition of preferred answer sets. Our new proposal will be developed in the following section.

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.

Based on these considerations, we develop our approach.

4.1 Prioritized extended logic programs

We start with the syntactical part of our framework for specifying priorities on extended logic programs, and define the concepts of prioritized rule bases and extended logic programs as follows.

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 which is well-behaved with respect to R. In particular, $\mathcal{R}$ is a prioritized (extended) logic program, if R is an (extended) logic program.* ∎

We call a partial order $<$ well-behaved with respect to R iff the ground instantiation of $(R, <)$ is defined (see Definition 4.2 below). Recall that a strict partial order is an irreflexive ($a \not< a$, for all elements a) 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". Our goal is to use this information to define the notion of a preferred answer set.

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. E.g., if we have the facts $bird(tweety)$ and $peng(tweety)$, then the ordering of these facts is intuitively irrelevant, and we do not want to care about it.

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

$$\begin{array}{ll} r_1: & p(x) \leftarrow q(x) \\ r_2: & r(x) \leftarrow \neg s(x, y) \end{array}$$

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, i.e., to satisfaction of the second-order axiom

$$(\forall X \subseteq R).(X \neq \emptyset \rightarrow (\exists x \in X)(\forall y \in X).(x = y \vee x < y)).$$

Any such ordering is called a *well-ordering*; notice that well-orderings are customary with infinite rule systems, cf. [44, 48].

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. For example, $ord(\emptyset) = 0$, $ord(\{a\}) = 1$, $ord(\{p(a), p(f(a)), p(f(f(a))), \ldots\} = \omega$ and so on. 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. [66].

Definition 4.2 *Let P be a logic program, $<$ a strict partial order on P. Then, the ground instantiation of $(P, <)$ is the pair $(P^*, <^*)$ 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 $r_1 < r_2$, provided that $<^*$ is a strict partial ordering; otherwise, the ground instantiation is undefined.* ∎

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 $(P,<)$. Moreover, it is easy to avoid such conflicts, by tagging rules with dummy literals such that common rule instances are not possible. An alternative would be to incorporate implicit contradiction removal into the definition, by adding the constraint $r_1^* \neq r_2^*$. However, it seems more appropriate that syntactically inconsistent priorities should raise an exception which is handled in an explicit way, rather than by some implicit procedure. This is why we require the partial order to be well-behaved in Definition 4.1.

Definition 4.3 *A full prioritization of any prioritized 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.* ∎

A prioritized program $(P,<)$ may give rise to more than one well-ordered ground rule base $(P^*,<')$ (possibly to infinitely many), where $<'$ refines the ordering $<^*$, even if the order $<$ on P is total. The question rises how to deal with the ambiguity of full prioritizations $(R,<')$ of a ground rule base $\mathcal{R}$ in a definition of preferred answer set for $\mathcal{R}$. A possible solution would be that some canonical $(R,<')$ is chosen from $\mathcal{FP}(\mathcal{R})$, based on some principle (e.g., lexicographic ordering), and only this particular $(R,<')$ is considered in the definition of preferred answer set. On the other hand, every full priorization $(R,<')$ is compatible with the priority specification in $\mathcal{R}$ (in particular, with the priority information in a prioritized logic program $\mathcal{P}$ whose grounding is $\mathcal{R}$), 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. The investigation of the properties of preferred answer sets in Section 5, in particular the observation in the paragraph after Proposition 5.4, provides support for this decision.

4.2 Preferred answer sets

For a ground rule base $\mathcal{R} = (R,<)$ (resp., a prioritized program $\mathcal{P} = (P,<)$), we define its answer sets to be the answer sets of R (resp., P), and denote the collection of all answer sets by $\mathcal{AS}(\mathcal{R})$ (resp., $\mathcal{AS}(\mathcal{P})$). Thus, $\mathcal{AS}(\mathcal{R}) = \mathcal{AS}(R)$ (resp., $\mathcal{AS}(\mathcal{P}) = \mathcal{AS}(P)$), which means that the ordering $<$ is simply disregarded.

In the rest of this section, we introduce our concept of preferred answer sets, which takes the ordering $<$ into account. We will first formulate the criterion which an answer set must satisfy in order to be preferred, given a fully prioritized ground rule base $\mathcal{R} = (R, <)$. We then extend it to arbitrary ground rule bases and logic programs.

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. To check whether A is an answer set of such a rule base R of ground rules, we must reduce it with A, and then check whether the resulting rule set has consequences A.

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 $C_{\mathcal{R}} : 2^{Lits} \to 2^{Lits}$. 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 the following two conditions are satisfied: (1) r is not defeated by literals collected so far, and (2) if $head(r) \in A$, then r must be applied in A. Note that condition (2) is necessary to avoid situations in which a literal in A is derived by two rules r_1 and r_2 during the construction of the fixpoint $C_{\mathcal{R}}(A) = A$ such that $r_1 < r_2$, where r_2 is applicable in A but r_1 is not. This would attribute the conclusion $head(r_1)$ a priority which is higher than effectively sanctioned by the rules. We discuss this aspect later in Example 5.3 and the preceding paragraph, which also shows that in general Principle I would be violated if condition (2) is omitted. The formal definition of $C_{\mathcal{R}}$ is as follows.

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

$$S_\alpha = \begin{cases} \bigcup_{\beta<\alpha} S_\beta, & \text{if } r_\alpha \text{ is defeated by } \bigcup_{\beta<\alpha} S_\beta \text{ or} \\ & head(r_\alpha) \in S \text{ and } r_\alpha \text{ is defeated by } S, \\ \bigcup_{\beta<\alpha} S_\beta \cup \{head(r_\alpha)\}, & \text{otherwise.} \end{cases}$$

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.

To see this it is helpful to note that an answer set A divides the rules R into three groups, namely *generating rules*, which are applied and contribute to the construction of A, *dead rules*, which are not applicable in A but whose consequences would not add anything new if they were applied since they appear in A anyway, and *zombie rules*, which are those not applicable in A and whose consequences do not belong to A. Only zombie rules have the potential to render answer sets non-preferred. This is the case if at least one zombie is not "killed" by a generating rule of higher priority. By examining rules with decreasing priority, collecting heads of generating rules and neglecting dead rules during this process the construction of $C_{\mathcal{R}}$ guarantees that indeed all zombies are defeated by rules with higher preference whenever A is a fixpoint of $C_{\mathcal{R}}$.

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 $C_{\mathcal{R}}(A) = A$.* ∎

To illustrate this definition, let us consider a simple example.

Example 4.1 Let $\mathcal{R} = (R, <)$ where R consists of the following two rules:

$$\begin{array}{ll} (1) & a \leftarrow \text{not } b \\ (2) & b \leftarrow \text{not } a \end{array}$$

and the ordering $<$ says $(1) < (2)$. The above program has the two answer sets $A_1 = \{a\}$ and $A_2 = \{b\}$. The set A_1 is a preferred answer set. Indeed, the computation of $C_{\mathcal{R}}(A_1)$ yields $S_0 = \{a\}$ (rule (1) is applied and a is included) and $S_1 = S_0 = \{a\}$ (as rule (2) is defeated by $\{a\}$); note that $ord(<) = 2$. Hence, $C_{\mathcal{R}}(A_1) = S_1 = \{a\} = A_1$.

On the other hand, A_2 is not a preferred answer set, since $C_{\mathcal{R}}(A_2) = \{a\}$. Thus, $\mathcal{R}$ has the unique preferred answer set A_1. ∎

Let us check that the existence of a single preferred answer set as in the previous example is not incidental, and 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. Then $\mathcal{R}$ has at most one preferred answer set.*

Proof. Let A_1 and A_2 be two answer sets of R, $A_1 \neq A_2$, and assume both are fixpoints of $C_{\mathcal{R}}$. Let r be the $<$-least rule such that r is applied in one of the answer sets but $head(r) = l$ is not contained in the other. Without loss of generality we assume $l \in A_1$. Since $l \notin A_2$, r must be defeated by the head of a rule r' applied in A such that $r' < r$. But since r is the $<$-least rule whose head is in exactly one of A_1 and A_2, the head of r' must be in A_1, so r cannot be applied in A_1, contrary to our assumption. ∎

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. We need some formal definitions.

Definition 4.6 *Let $(R, <) = \{r_\alpha\}_<$ be a fully prioritized ground rule base, and let X be a set of ground literals. Let ${}^X\mathcal{R} = ({}^XR, {}^X<)$ be the fully prioritized ground rule base 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 first rule in R with respect to $<$ such that r' results from r by step 2. ∎

It is easily seen that ${}^X<$ is indeed a well-ordering on ${}^XR$, and that ${}^X\mathcal{R}$ is thus indeed a rule base. 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. Observe that we could alternatively, similar as in [41], consider programs as multisets of rules, such that common rule instances are possible. However, we prefer to keep the familiar framework of programs as sets of rules; the elimination of duplicate rules is simple and intuitive, and does not lead to technical problems.

Example 4.2 Consider the prioritized rule base $\mathcal{R} = (R, <)$, where R is the ground instantiation of the rules

$$
\begin{array}{ll}
(1) & p(a, f(a)) \leftarrow \\
(2) & q(f(a)) \leftarrow \\
(3) & r(x) \leftarrow q(x), \mathrm{not}\, p(x, f(a)) \\
(4) & r(y) \leftarrow p(a, y), \mathrm{not}\, p(y, f(y))
\end{array}
$$

and $<$ orders the ground rules such that instances of a rule with lower number are before instances of rules with a higher number, and instances from the same rule are ordered by increasing depth of the ground term replacing the variable, i.e.,

$$
\begin{array}{l}
p(a, f(a)) \leftarrow, \\
q(f(a)) \leftarrow,
\end{array}
$$

$$
\begin{aligned}
r(a) &\leftarrow q(a), \mathrm{not}\, p(a, f(a)), \\
r(f(a)) &\leftarrow q(f(a)), \mathrm{not}\, p(f(a), f(a)), \\
&\dots \\
r(a) &\leftarrow p(a, a), \mathrm{not}\, p(a, f(a)), \\
r(f(a)) &\leftarrow p(a, f(a)), \mathrm{not}\, p(f(a), f(f(a))), \\
&\dots
\end{aligned}
$$

The order type of $<$ is $\omega + \omega = 2\omega$, i.e., $ord(<) = 2\omega$. Let $X = \{q(f^i(a)),\, p(a, f^i(a)) \mid i \geq 0\}$. Then, the dual reduct ${}^X\mathcal{R} = ({}^X R, {}^X<)$ is the following totally ordered rule base:

$$
\begin{aligned}
p(a, f(a)) &\leftarrow , \\
q(f(a)) &\leftarrow , \\
r(a) &\leftarrow \mathrm{not}\, p(a, f(a)), \\
r(f(a)) &\leftarrow \mathrm{not}\, p(f(a), f(a)), \\
&\dots \\
r(a) &\leftarrow \mathrm{not}\, p(a, f(a)), \\
r(f(a)) &\leftarrow \mathrm{not}\, p(f(a), f(f(a))), \\
&\dots
\end{aligned}
$$

Note that the rule $p(a) \leftarrow \mathrm{not}\, p(a, f(a))$ results from the dual reduction of different ground instances of (3) and (4). The map f selects for the ordering ${}^X<$ the reduct of the instance of (3). ■

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 ground rule base $\mathcal{R} = (R, <)$, 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 just if it is a fixpoint of Gelfond and Lifschitz's operator γ_R (see Definition 2.3) and a fixpoint of $\lambda_{\mathcal{R}}$.

To see how our approach works, let us discuss 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 4.3 Let us consider the classical Tweety example, but impose some preferences on the rules as in Section 1:

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

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 following rules:

$$
\begin{array}{lr}
(1) & peng(tweety) \leftarrow \\
(2) & bird(tweety) \leftarrow \\
(3) & \neg flies(tweety) \leftarrow \text{not}\, flies(tweety) \\
(4) & flies(tweety) \leftarrow \text{not}\, \neg flies(tweety)
\end{array}
$$

Now, let us determine $\lambda_{\mathcal{P}^*}(A_1) = \bigcup_\alpha A_{1,\alpha}$, by constructing the sequence $A_{1,0}, A_{1,1}, \ldots$ We have $ord(<^*)$ = 4 and obtain

$$
\begin{aligned}
A_{1,0} &= \{peng(tweety)\}, \\
A_{1,1} &= \{peng(tweety), bird(tweety)\}, \\
A_{1,2} &= \{peng(tweety), bird(tweety), \neg flies(tweety)\}, \text{ and} \\
A_{1,3} &= A_{1,2} = \bigcup_{\alpha < ord(<^*)} A_{1,\alpha}.
\end{aligned}
$$

Thus, $\lambda_{\mathcal{P}^*}(A_1) = \{peng(tweety), bird(tweety), \neg flies(tweety)\} = A_1$; 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.

Thus, A_1 is the unique preferred answer set of P, which is intuitive.

Notice that we have a similar behavior if we introduce other individuals in this scenario, e.g. by adding a fact $ostrich(sam)$ or a rule $bird(father(x)) \leftarrow bird(x)$. In the latter case, the ground version of the program is not fully prioritized, and different fully prioritized versions have to be explored. However, for each of them, we obtain the same unique preferred answer set, which contains A_1. Thus, the way of resolving unspecified priorities does not matter in this case. ∎

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:

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

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:

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

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 generating 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 ground rule base, and let $A \in \mathcal{AS}(R)$. Then, A is a preferred answer set of $\mathcal{R}$, if and only if for each rule $r \in R$ which is a zombie, i.e., $pre(r) \subseteq A$ and $head(r) \notin A$, there is a generating rule $r' \in R$ such that $r' < r$ and $head(r')$ defeats r.* ∎

The proposition formalizes our intuitive explanation of the definition of preferred answer sets for prerequisite-free ground programs (cf. the discussion preceding Definition 4.5), and extends it to the case with prerequisites.

For the proof of this proposition, the following lemma is useful. Slightly abusing notation we use ${}^{A}r$ to denote the prerequisite-free part of rule r whenever $pre(r) \in A$. If $pre(r) \notin A$ then ${}^{A}r$ is undefined.

Lemma 5.2 *Let $\mathcal{R} = (R, <)$ be a fully prioritized ground rule base, and let $A \in \mathcal{AS}(\mathcal{R})$. Then, the following are equivalent:*

(i) *A is a preferred answer set of $\mathcal{R}$.*

(ii) *For every $r \in R$, the reduct ${}^{A}r$ is defined and fires in the construction of $\lambda_{\mathcal{R}}(A)$, if and only if r is a generating rule of A.*

Proof. (of Proposition 5.1).

($\Rightarrow$). Assume the condition of the right hand side is violated. Let r be the $<$-least rule violating this condition. Clearly, ${}^{A}r$ is contained in ${}^{A}\mathcal{R}$. Since A is a preferred answer set, it follows from Lemma 5.2 A that ${}^{A}r$ does not fire in the construction of $\lambda_{\mathcal{R}}(A)$. Hence, ${}^{A}r$ must be defeated by some rule in ${}^{A}\mathcal{R}$ with higher priority. From Lemma 5.2 again, it follows that ${}^{A}r$ is defeated by the reduct ${}^{A}r'$ of some rule $r' \in R$ such that $r' < r$ and r' is a generating rule of A. It follows that r is defeated by r', which raises a contradiction.

($\Leftarrow$). Suppose that the right hand side holds, and assume further that A is an answer set which is not preferred. We derive a contradiction. Since A is not an answer set, by Lemma 5.2 there must exist a $<$-least rule $r \in R$ such that ${}^{A}r$ is applied in the construction of $\lambda_{\mathcal{R}}(A)$ iff r is not applied in A. There are two cases:

Case 1: r *is not applied in* A. Then, ${}^{A}r$ fires in the construction of $\lambda_{\mathcal{R}}(A)$. By choice of r, for every rule $r' \in R$ with $r' < r$ it holds that r' is a generating rule of A iff ${}^{A}r'$ fires in the construction of $\lambda_{\mathcal{R}}(A)$. It follows that r is not defeated by any $r' < r$ which is applied in A. However, this r violates the condition of the right hand side, which is a contradiction.

Case 2: r *is applied in* A. Then, ${}^{A}r$ is not applied in the construction of $\lambda_{\mathcal{R}}(A)$. A similar argument shows that r must be defeated by some rule $r' < r$ which is applied in A. This means that r is not applied in A, which is a contradiction. ∎

With this result we can 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 5.3 *The preferred answer set approach satisfies Principles I and II as described in Section 3.1.*

Proof. *Principle I*: Let $\mathcal{R} = (R, <_{\mathcal{R}})$ be a prioritized logic program and let A_1 and A_2 be two answer sets of R generated by the (ground) rules $R' \cup \{d_1\}$ and $R' \cup \{d_2\}$, respectively. Moreover, assume that $d_1 <_{\mathcal{R}} d_2$. We show that A_2 is not a preferred answer set of $\mathcal{R}$ using the characterization of Proposition 5.1.

Since d_1 and d_2 are applied in A_1 and A_2, respectively, their prerequisites must be provable using the rules in R'. Now assume A_2 is a preferred answer set of $\mathcal{R}$. Then A_2 is a preferred answer set of some $\mathcal{P} = (P, <) \in FP(\mathcal{R}^*)$. Since the prerequisites of d_1 are in A_2 and $head(d_1)$ cannot be in A_2 due to the well-known maximality of answer sets, Proposition 5.1 implies that d_1 is defeated by the head of a rule d in

$R' \cup \{d_2\}$ with higher priority than d_1. Since we have $d_1 <_{\mathcal{R}} d_2$ and thus $d_1 < d_2$, d can only be in R'. But then A_1 cannot be an answer set of R, contrary to our assumption.

Principle II: Let A be a preferred answer set of a prioritized logic program $\mathcal{R} = (P, <_{\mathcal{R}})$. Then A is a preferred answer set of some $\mathcal{P} = (P, <) \in FP(\mathcal{R}^*)$. Let r be a rule such that at least one prerequisite of r is not contained in A. Furthermore, let $\mathcal{R}' = (P \cup \{r\}, <_{\mathcal{R}'})$ where $<_{\mathcal{R}'}$ agrees with $<_{\mathcal{R}}$ on rules in P. Then there exists some $\mathcal{P}' = (P \cup \{r\}, <') \in FP(\mathcal{R}'^*)$ such that $<'$ agrees with $<$ on rules in P. We show that A is a preferred answer set of $\mathcal{P}' = (P \cup \{r\}, <')$ and thus a preferred answer set of $\mathcal{R}'$.

Clearly, A is an answer set of $P \cup \{r\}$ since $\gamma_P(A) = \gamma_{P \cup \{r\}}(A)$ whenever some prerequisite of r is not in $\gamma_P(A)$. Moreover, since A does not contain all prerequisites of r this rule is not contained in the dual A-reduct of $\mathcal{P}'$. We thus have $C_{{}^A\mathcal{R}'}(A) = C_{{}^A\mathcal{R}}(A) = A$, that is A is a preferred answer set of $\mathcal{P}'$. ∎

Tracing back the proof of this proposition reveals that the clause "$head(r_\alpha) \in S$ *and* r_α *defeated by* S" is needed in the definition of the sequence S_α in Definition 4.4. It would have been tempting to omit this clause, and in many cases the resulting modified definition amounts to the same as the original one. However, there are cases in which the modified, simpler definition is not satisfactory.

Example 5.3 Consider the following program:

$$
\begin{array}{ll}
(1) & p \leftarrow \mathrm{not}\, q \\
(2) & q \leftarrow \mathrm{not}\, \neg q \\
(3) & \neg p \leftarrow \mathrm{not}\, p \\
(4) & p \leftarrow \mathrm{not}\, \neg p
\end{array}
$$

Again, we assume that (i) is preferred over (j) if $i < j$. The proposed variants of the definition would select from the answer sets $A_1 = \{p, q\}$ and $A_2 = \{\neg p, q\}$ the set A_1 as preferred answer set; however, this violates Principle I. Indeed, A_1 has the generating rules $R_1 = \{(2), (4)\}$ and A_2 has the generating rules $R_2 = \{(2), (3)\}$; thus, by Principle I, A_1 must not be preferred over A_2. The reason A_1 is a preferred answer set according to the modified definition is that rule (1) fires in the construction of the sequence S_α for $S = A_1$; however, this firing is unwanted, since rule (1) is defeated by the answer set which is finally constructed. This shows that the proposed "simplification" of our definition does not work. ∎

Observe that in the previous example, all rules except (1) correspond to normal default rules of the form $\alpha{:}\beta/\beta$ in Reiter's default logic. We remark that the presence of a non-normal rule is crucial: if all rules of a program express normal defaults, then the definition of S_α can be simplified by removing the clause "$head(r_\alpha) \in S$ *and* r_α *defeated by* S", without changing the semantics.

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

Proposition 5.4 *Let* $<_1$ *and* $<_2$ *be strict 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 5.5 *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* $\mathcal{R} = (R, <)$.

Proof. A suitable well-ordering $<$ can be constructed as follows. Let *App* be the collection of all rules r in R which are applied in A, i.e., generating in A. Now let $<$ be an arbitrary well-ordering of R such that $r_1 < r_2$ holds for all $r_1 \in App$, $r_2 \in R \setminus App$, i.e., the rules in *App* are ordered before all other rules. Then, clearly for every zombie rule $r \in R$ with respect to A, i.e., $pre(r) \subseteq A$, while r is not applied in A and $head(r) \notin A$, there is some rule $r' < r$ which is applied in A and $head(r')$ defeats r. Hence, by Proposition 5.1, A is a preferred answer set of $\mathcal{R}$. ∎

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

Corollary 5.6 *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 5.7 *Let $\mathcal{P} = (P, <)$ be a prioritized program. If $A = Lits$ is an answer set of $\mathcal{P}$, then $Lits$ is the unique preferred answer set of $\mathcal{P}$.*

Proof. The set $Lits$ is an answer set of P, if and only if $Cn(P^+) = Lits$ is inconsistent, where P^+ are the rules in P^* without weak negation. Every such rule $c \leftarrow a_1, \ldots, a_n$ survives the dual reduction, and the literal c is in ${}^AP$. Hence, for any full prioritization $\mathcal{P}' \in \mathcal{FP}(\mathcal{P}^*)$, clearly $\lambda_{\mathcal{P}'}(Lits) = Lits$, which implies that $Lits$ is a preferred answer set. ∎

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

Example 5.4 Consider the following program:

(1) $b \leftarrow \text{not}\neg b, a$
(2) $c \leftarrow \text{not } b$
(3) $a \leftarrow \text{not } c$

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 simple programs that have answer sets but for which no preferred answer set exists.

Example 5.5 Consider the following program:

(1) $c \leftarrow \text{not } b$
(2) $b \leftarrow \text{not } a$

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. For example, the above program is stratified and thus has a clear generally accepted semantics in which the evaluation order of the rules is intuitively determined. The user-defined priority of the second rule over the first one is not compatible with this natural order, and one might simply ignore it. In the next section, we show how this view of relaxing priority information can be implemented in our approach.

6 Weakly Preferred Answer Sets

As we have seen in the previous section, there are prioritized programs which have answer sets, but no preferred answer set, a situation which may be counterproductive in certain scenarios. To address this problem, we propose a relaxation that gives us exactly the preferred answer sets whenever they exist and some approximation, called *weakly preferred answer sets*, in the other cases.

Before we present the relaxation we want to point out that there is a price to pay: consistency preservation turns out to be incompatible with our principles, at least under rather natural conditions for approximating answer sets. We have the following proposition.

Proposition 6.1 *Let Rel be a relaxation of the notion of preferred answer sets, that is a function assigning to each prioritized ground rule base $\mathcal{R}$ a subset of $\mathcal{AS}(\mathcal{R})$ such that*

1. *$Rel(\mathcal{R}) = \mathcal{PAS}(\mathcal{R})$ whenever $\mathcal{PAS}(\mathcal{R}) \neq \emptyset$, and*
2. *$Rel(\mathcal{R}) \neq \emptyset$ whenever $\mathcal{AS}(\mathcal{R}) \neq \emptyset$.*

Defining the preferred belief sets of $\mathcal{R}$ as $Rel(\mathcal{R})$ violates Principle II from Section 3.1.

Proof. Consider Example 5.5. The program, let's call it $\mathcal{P}$, has a single answer set $A = \{b\}$ which, according to clause 2 in the proposition, is contained in $Rel(\mathcal{P})$. Let $\mathcal{P}'$ be $\mathcal{P}$ extended by the rule $d = a \leftarrow c$ with highest priority. Note that d's prerequisite is not in A. Nevertheless, according to clause 1, $A \notin Rel(\mathcal{P}')$ since $\mathcal{P}'$ has the single preferred answer set $\{a, c\}$. Principle II is thus violated. ∎

This shows that we will have to trade satisfaction of Principle II for consistency of the semantics. We remark that technically, some preference Rel which satisfies Principle I can always be constructed in the case of a finite $\mathcal{R}$, though such a preference need not be appealing in general. In the infinite case, no such preference may exist, if generating rules are understood to use assumptions in the body.

The basic idea of our relaxation 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 $<_1$.

Example 6.1 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 amounts to counting these inversions in a way. 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. Denote by $Inv(<_1, <_2)$ the set of all inversions of $<_2$ in $<_1$, and let $V_\beta(<_1, <_2)$ denote the well-ordering induced by $<_1$ on the set*

$$\{r_\alpha \mid r_\alpha <_1 s_\beta, (r_\alpha, s_\beta) \in Inv(<_1, <_2)\},$$

i.e., the elements of M which are in $<_1$ before the element s_β and in $<_2$ after s_β.

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

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

In the previous example, we have $Inv(<_1, <_2) = \{(c, a), (c, b), (d, b)\}$ and $V_0(<_1, <_2) = c$, $V_1(<_1, <_2) = c < d$, $V_2(<_1, <_2) = \emptyset$, and $V_3(<_1, <_2) = \emptyset$; hence, $d(<_1, <_2) = 1 + 2 + 0 + 0 = 3$.

The definition of $d(<_1, <_2)$ involves ordinal arithmetic, which may be less familiar to the reader; see e.g. [66] 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 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 [66]. 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 $<_2$.

In general, $d(<_1, <_2)$ is a transfinite ordinal which is bounded by $ord(<_1) \cdot ord(<_2)$. Thus, if the underlying set is countable (as in our application), then $d(<_1, <_2)$ is countable.

Example 6.2 Suppose we have ground atoms p and $q(a), q(f(a)), \ldots$ which are ordered as follows:

$$\begin{array}{ll} <_1: & p,\, q(a),\, q(f(a)),\, q(f(f(a))),\, \ldots \\ <_2: & q(a),\, q(f(a)),\, q(f(f(a))),\, \ldots\, p \end{array}$$

Then, $d(<_1, <_2) = 1 + 1 + \cdots + 0 = \omega$; for every q-atom, there is one violation of $<_2$ in $<_1$, and no violation for atom p. On the other hand, $d(<_2, <_1) = \omega + 0 + 0 + \cdots = \omega$; the ordering $<_2$ has ω many violations of the ordering of p in $<_1$, and no violations for the q-atoms. ∎

Clearly, $d(<, <) = 0$ holds for any well-ordering $<$, while symmetry does not hold in general. E.g., consider the following orderings of ground atoms $p(x)$, $q(x)$ over terms $a, f(a), \ldots$:

$$\begin{array}{ll} <_1: & p(a),\, p(f(a)),\, p(f(f(a))),\, \ldots\, q(a),\, q(f(a)),\, q(f(f(a))),\, \ldots \\ <_2: & p(a),\, q(a),\, p(f(a)),\, q(f(a)),\, p(f(f(a))),\, q(f(f(a))),\, \ldots \end{array}$$

i.e., $<_1$ orders first, increasing by depth, all atoms $p(f^i(a))$ and then all atoms $q(f^i(a))$, while $<_2$ orders all atoms by depth and breaks ties by preferring p-atoms over q-atoms; thus, $ord(<_1) = \omega + \omega = 2\omega$ and $ord(<_2) = \omega$. We have

$$d(<_1, <_2) = 0 + \omega + 0 + \omega + 0 + \omega + \cdots = \omega^2,$$

while

$$d(<_2, <_1) = 0 + 1 + 2 + \cdots + 0 + 0 + 0 + \cdots = \omega.$$

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 6.2 *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., $d(<_2, <_1) = d(<_1, <_2)$.*

Proof. Clearly, $Inv(<_2, <_1) = \{(m, m') \mid (m', m) \in Inv(<_1, <_2)\}$. Observe that if $d(<_1, <_2)$ is finite, then only finitely many $V_\beta(<_1, <_2)$ are nonempty, and each of them is finite. It follows that also only finitely many nonempty $V_\alpha(<_2, <_1)$ exist, and each of them is finite; thus, also $d(<_2, <_1)$ is finite. Since $d(<_1, <_2)$ is finite, $d(<_1, <_2)$ amounts to the number of elements in $Inv(<_1, <_2)$; since $|Inv(<_2, <_1)| = |Inv(<_1, <_2)|$ and $d(<_2, <_1)$ is finite, it follows $d(<_1, <_2) = d(<_2, <_1)$. ∎

Corollary 6.3 *On the full prioritizations of a ground rule base, the distance measure d' which is defined by $d'(<_1, <_2) = card(d(<_1, <_2))$, where $card(\alpha)$ is the cardinal associated with the ordinal α, is symmetric.*

Proof. If $d(<_1, <_2)$ is finite, this holds by the previous proposition. If $d(<_1, <_2)$ is infinite, then also $d(<_2, <_1)$ is infinite, and since $ord(<_1)$ and $ord(<_2)$ are countable, both $d(<_1, <_2) \leq ord(<_1) \cdot ord(<_2)$ and $d(<_2, <_1) \leq ord(<_2) \cdot ord(<_1)$ are countable; hence, $card(d(<_1, <_2)) = card(d(<_2, <_1)) = \aleph_0$. ∎

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.

The value of $d(<_1, <_2)$ has an alternative, transformation-based interpretation which intuitively alludes to the amount of work that is needed in order to transform the ordering $<_1$ into the ordering $<_2$; $V_\beta(<_1, <_2)$ says how many elements r_α must be moved behind s_β. In particular, in the finite case, $d(<_1, <_2)$ amounts to the smallest number of successive switches of neighbored elements which are needed to transform $<_1$ into $<_2$. In fact, this is precisely the number of switches executed by the well-known bubble-sort algorithm.[3] E.g., for the orderings $c <_1 a <_1 d <_1 b$ and $a <_2 b <_2 c <_2 d$ from Example 6.1, bubble-sort performs three switchings in sorting $<_1$ into $<_2$ (c with a, d with b, and c with b); this yields $d(<_1, <_2) = 3$. In the infinite case, a similar interpretation does not apply in all cases; we come back to a transformational view of distance at the end of this section.

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 $\mathcal{R}'$, i.e.,*

$$pvd_\mathcal{R}(A) = \min\{d(<_1, <_2) \mid (R, <_1) \in \mathcal{FP}(\mathcal{R}), A \in \mathcal{PAS}(R, <_2)\}.$$

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 5.5, 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 $\mathcal{P}$.* ∎

Example 6.3 Consider the following program $\mathcal{P}$:

$$\begin{array}{ll} (1) & a \leftarrow \text{not}\, c \\ (2) & c \leftarrow \text{not}\, b \\ (3) & \neg d \leftarrow \text{not}\, b \\ (4) & b \leftarrow \text{not} \neg b, a \end{array}$$

This program has two answer sets, namely $A_1 = \{a, b\}$ and $A_2 = \{c, \neg d\}$; however, none of them is preferred. We have $pvd_\mathcal{P}(A_1) = 2$, because (2) and (3) are zombie rules which can be defeated only by (4), which must be moved in front of these rules; this takes two switches. On the other hand, we have $pvd_\mathcal{P}(A_2) = 1$, since the single zombie rule (1) is defeated if (2) is moved in front of it (note that (4) is a dead rule for A_2). Hence, $pvd(\mathcal{P}) = 1$, and A_2 is the weakly preferred answer set of $\mathcal{P}$. ∎

The following results are easily verified.

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

[3] Recall that bubble-sort runs through the list of elements and switches neighbored elements which are in the wrong order; this process is repeated, until the list is sorted.

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

Proposition 6.5 *Let $\mathcal{P} = (P, <)$ be a prioritized logic program. If $\mathcal{AS}(\mathcal{P}) \neq \emptyset$, then $\mathcal{WAS}(\mathcal{P}) \neq \emptyset$.*

Proof. Since every set of ordinals has a minimum, $\min\{pvd_{\mathcal{P}}(A) \mid A \in \mathcal{AS}(P)\} = pvd_{\mathcal{P}}(A)$ holds for some $A \in \mathcal{AS}(P)$. Hence, from the definition, A is a weakly preferred answer set of $\mathcal{P}$. ∎

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

Let us now turn to the question whether Principles I and II are preserved when we switch from strongly preferred answer sets to weakly preferred ones. Unfortunately, this is not the case.

Proposition 6.6 *The weakly preferred answer set approach does not satisfy Principle I from Section 3.1.*

Proof. Consider the following program $\mathcal{P}$:

(1)	$a \leftarrow \mathrm{not}\, \neg a$	(4)	$c \leftarrow \mathrm{not}\, \neg c$
(2)	$\neg a \leftarrow \mathrm{not}\, a$	(5)	$\neg a \leftarrow \mathrm{not}\, b, a$
(3)	$\neg a \leftarrow \mathrm{not}\, c, a$	(6)	$b \leftarrow \mathrm{not}\, b$

This program has two answer sets, namely $A_1 = \{a, b, c\}$, which is generated by $R_1 = \{(1), (4), (6)\}$, and $A_2 = \{\neg a, b, c\}$, which is generated by $R_2 = \{(2), (4), (6)\}$. Assuming that $(i) < (j)$ if $i < j$, we have that none of these answer sets is strongly preferred. The set A_2 is a weakly preferred answer set, since after switching rules (1) and (2), A_2 is a preferred answer set (thus $pvd_{\mathcal{P}}(A_2) = 1$), while $pvd_{\mathcal{P}}(A_1) = 2$ (switch rules (3),(4) and (5),(6)). Since rule (1) is preferred over (2), by Principle I, A_2 must not be selected as a weakly preferred answer set. ∎

The failure of Principle II is immediate from Prop. 6.1. We believe that the use of weakly preferred answer sets is justifiable even in the light of these properties. In fact, we have defined the concept as an *approximation* of preferred answer sets; if no preferred answer set exists, then we must relax the requirements and this may include Principles I and II; we trade this for consistency of the semantics.

In the above example programs, we have argued that an answer set A of a program $\mathcal{P}$ is weakly preferred by moving generating rules forward such that all zombies are defeated. We now establish that this intuitive way of making A weakly preferred is in fact legal. We start with some formal notions on well-orderings.

A *switch* on a well-ordering $<$ of a finite set M is a pair $s = (i, i+1)$ of positions in $<$ such that $0 \leq i$ and $i + 1 \leq |M|$. The result of applying s on $<$, denoted $s(<)$, is the well-ordering resulting from $<= \{r_\alpha\}_<$ by interchanging r_i and r_{i+1}. A *switching sequence* is finite sequence $\sigma = s_1, s_2, \ldots, s_n$ of switches, whose application on $<$, denoted by $\sigma(<)$, is defined as $\sigma(<) = s_n(s_{n-1}(\cdots(s_1(<))\cdots))$; the length of σ is denoted by $|\sigma|$.

Suppose $<'$ is a well-ordering of M. Then, $<'$ can be obtained from $<$ as follows. Let $<_{-1}$ be $<$ and denote by $<_i$ the ordering resulting from $<_{i-1}$ by applying the switching sequence τ_i, which switches the i-th element of $<'$ from its position j_i in $<_{i-1}$ to position i, for $i = 0, \ldots, |M| - 1$. Then, $<'$ is identical to $<_k$, where $k = |M| - 1$. Observe that $j_i \geq i$ holds for all i, and that τ_k is void. The sequence $\sigma = \tau_1, \tau_2, \ldots, \tau_k$ is called the *canonical switching sequence of* $<'$. Intuitively, in σ the ordering $<'$ is built from $<$ by a kind of reverse bubble-sort, where the next iteration starts at some point in the sequence $<_{i-1}$ rather than at its end.

Example 6.4 For the orderings $c < a < d < b$ and $a <' b <' c <' d$ as in Example 6.1, we have:

i	τ_i	$<_i$
-1	—	c, a, d, b
0	$(0, 1)$	a, c, d, b
1	$(2, 3), (1, 2)$	a, b, c, d
2	void	a, b, c, d
3	void	a, b, c, d

Thus, the canonical switching sequence is $\sigma = (0,1), (2,3), (1,2)$. ∎

The following property is easily seen.

Proposition 6.7 *Let σ be the canonical sequence transforming $<$ into $<'$. Then, $|\sigma| = d(<, <')$.*

Proof. Observe that starting from $<$, each switch in σ reduces the number of inversions for $<'$ by one. Hence, the length of σ is the number of inversions of $<'$ in $<$, which coincides with $d(<, <')$. ∎

Let $\mathcal{P} = (P, <)$ be a fully prioritized finite ground program, and let A be an answer set of $\mathcal{P}$. We call a well-ordering $<'$ such that A is a preferred answer set of $(P, <')$ and $d(<, <') = pvd_{\mathcal{P}}(A)$ an *optimal ordering* for A.

Lemma 6.8 *Let $A \in \mathcal{AS}(\mathcal{P})$, where $\mathcal{P} = (P, <)$ is a fully prioritized finite ground program, and let $<'$ be an optimal ordering for A. If τ_i from the canonical switching sequence σ of $<'$ is nonvoid, then $<_{i-1}$ has at position i a non-defeated zombie.*

The lemma implies that in the canonical transformation of $<$ into an optimal ordering $<'$ of A, we never move a rule to the position i unless there is a non-defeated zombie. The next result tells us that without loss of generality, we may assume that only a rule defeating this zombie is moved to this position.

Theorem 6.9 *Let $A \in \mathcal{AS}(\mathcal{P})$, where $\mathcal{P} = (P, <)$ is a fully prioritized finite ground program. Then, there exists an optimal ordering $<'$ for A such that τ_i from the canonical switching sequence σ of $<'$ is nonvoid if and only if $<_{i-1}$ has at position i a non-defeated zombie z, and every nonvoid τ_i moves a rule defeating z to position i.*

The first part of this theorem follows from the previous lemma, while the second is established by modifying an optimal ordering such that only rules defeating zombies are moved. As a consequence, a weakly preferred answer set of $(P, <)$ can be obtained by eliminating all non-defeated zombies through moving generating rules defeating them ahead as economically as possible. This justifies the informal argumentation in Example 6.3 and the proof of Proposition 6.1.

Notice, however, that it is not true that a rule defeating a zombie is moved to position i in *every* optimal ordering for A.

Example 6.5 Consider the following program $\mathcal{P}$:

(1)	$a \leftarrow \mathrm{not}\, b$
(2)	$c \leftarrow$
(3)	$b \leftarrow$

where $(1) < (2) < (3)$. The set $A = \{b, c\}$ is the unique answer set of $\mathcal{P}$, and both $(3) <' (1) <' (2)$ and $(2) <'' (3) <'' (1)$ are optimal orderings for A; however, $\tau''_0 = (0,1)$ in the canonical switching sequence $\sigma'' = (0,1), (1,2)$ of $<''$ moves (2) to position 0, which does not defeat the zombie (1). ∎

We conclude this section with some comments on possible alternative ways to define weakly preferred answer sets. First, let us remark that a distance measure between well-orderings may be defined which is strictly guided by a transformation-oriented view, based on a proper formalization of the notion of switch. The distance from $<_1$ to $<_2$ can then be chosen as the shortest sequence which transforms $<_1$ into $<_2$. However, the formal definition is, due to the need for executing transfinite switches in general, more involved than the one from above. In several cases, the measures do coincide, but there are cases in which the transformation-based distance value is not much intuitive. E.g., in Example 6.2, the transformation-based distance from $<_1$ to $<_2$ would be ω, since we must move atom p with a transfinite switch in ω many steps behind all other atoms; the distance from $<_2$ to $<_1$ would be ω^2, however, since we must first move $q(a)$ behind p, then $q(f(a))$, and so on; this needs ω many transfinite switches, i.e., ω^2 switches in total. However, this value is not much appealing. We therefore prefer the simpler function for measuring the distance from above.

$$
\begin{tikzcd}
 & \mathrm{NP} \arrow[dr, "\subsetneq"] & & \\
\mathrm{P} \arrow[ur, "\subsetneq"] \arrow[dr, "\subsetneq"'] & & \mathrm{P}^{\mathrm{NP}}[O(\log n)] \arrow[r, "\subseteq"] & \mathrm{P}^{\mathrm{NP}} \\
 & \text{co-NP} \arrow[ur, "\subsetneq"'] & &
\end{tikzcd}
$$

Figure 1: Relationship of complexity classes

Our approach takes the view that the quality of the approximation of a preferred answer set is given by the number of preference violations, and we do not distinguish between violations of high priority and low priority. In particular, it may be the case that a single violation is present, and regardless of whether this violation involves the two top-ranked rules or the two bottom-ranked ones, this violation counts the same.

An alternative, strictly hierarchic view of violations would be considering the first violation of the priority ordering as decisive for how far two orderings are from each other. Then, an answer set is selected, if the highest priority violation compared to a preferred answer set occurs as late as possible, i.e., it is possible to keep the ordering of priorities as long as possible. However, this approach has some drawbacks. In the infinite case, the existence of a weakly preferred answer set is a priori not guaranteed, since no longest sequence of obeyed priorities might exist. Another aspect is that a single violation may weigh more than a large – even infinite – number of other violations; it is questionable whether this is satisfactory in certain situations. These considerations led us to not consider such an approach to weakly preferred answer sets here.

An intermediate approach, which actually generalizes both approaches discussed above would be assigning penalties to the violation of priorities, and selecting the answer sets which have the smallest penalty. However, this approach requires knowledge about how priority violations should be mapped into penalty values, and it is often not clear how such values should be reasonably assigned. For this reason, we believe that currently the development of such a framework is not a major issue.

7 Algorithms and 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; [35, 53] are good sources. For a background on complexity results in logic programming, refer to [64, 22, 17]. In our analysis, we focus on the case of finite propositional (ground) prioritized programs, but we will also address nonground programs.

We briefly recall the definitions of the complexity classes that we need for assessing the complexity of preferred and weakly preferred answer sets. P (resp., NP) is the class of decision problems, i.e., Yes/No problems, which are solvable in polynomial time on a deterministic (resp., nondeterministic) Turing machine; the class P^{NP} (also denoted Δ_2^p) contains the decision problems solvable on a deterministic Turing machine in polynomial time with access to an oracle for problems in NP; $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ (alias $\Delta_2^p[O(\log n)]$, Θ_2^p[70]) denotes the subclass of P^{NP} in which the oracle access is limited to $O(\log n)$ many queries, where n is the size of the problem input. As usual, for a complexity class C, the class of the complementary problems is denoted by co-C; note that co-P $=$ P, co-$\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ = $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$, and co-$\mathrm{P}^{\mathrm{NP}}$ = P^{NP}. The inclusion relationship of these classes is shown in Figure 1; each inclusion is widely believed to be proper.

For measuring the complexity of computational problems with output, we use the concept of a search problem, in which an arbitrary solution out of a set of possible solutions must be computed; here, a Turing machine has an output tape whose contents is the result of the computation if the machine halts in an accepting state. In case of a nondeterministic machine, output of different solutions in different runs is admissible. The search analogues of the decisional classes P, NP, P^{NP} etc are denoted by prefixed "F".

7.1 Preferred answer sets

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 [46, 47, 7] and the obvious correspondence between the preferred answer sets of the prioritized program $\mathcal{P} = (P, <)$ where $<$ is empty (cf. Corollary 5.6) 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. Intuitively, the many possible total refinements of a partial ordering $<$ on the rules are a source of complexity; since a full prioritization unambiguously fixes the order in which the rules have to be considered, we might expect that for fully prioritized programs the complexity is lower than for general programs. This intuition may be nurtured by the fact that in some nonmonotonic formalisms, full prioritization decreases complexity, cf. [49].

The first result is that priorities do fortunately not increase the complexity of extension checking for a program, i.e., given a set A of literals and a prioritized program $\mathcal{P}$, deciding whether this set constitutes a preferred answer set of the program. We first note that this holds in the case where the program is fully prioritized. The result can be derived from known results and an inspection of the definitions.

Proposition 7.1 *Given a fully prioritized (finite) propositional program $\mathcal{P} = (P, <)$ and a set $A \subseteq Lits$, both deciding whether $A \in \mathcal{AS}(P)$ and $A \in \mathcal{PAS}(\mathcal{P})$ is possible in polynomial time. Moreover, these problems are complete for* P *(under logspace-reductions).*

We have added in this proposition the matching lower bound of the complexity, to sharply characterize the computational properties of preferred extension checking; P-completeness of this problem tells us, adopting a general belief in the complexity theory community, that this problem is not amenable to parallelization.

For deriving an analogous result in the case of arbitrary rather than full prioritization, we note the following key lemma. Recall that we have divided in Section 4 the rules of a prerequisite-free prioritized program $\mathcal{P} = (P, <)$ with respect to an answer set A into generating rules (those which are applied in A), dead rules (those which are not applicable in A, but the head is from A), and zombie rules (those which are not applicable in A, and whose head is not in A), and that only zombie rules can prevent preference of A.

For a propositional prioritized program $\mathcal{P} = (P, <)$ and $A \in \mathcal{AS}(\mathcal{P})$, we construct a labeled directed graph $G(\mathcal{P}, A)$ as follows. The vertices are the rules P, and an edge is directed from r to r' if $r < r'$. Moreover, label each vertex r as follows: if the dual reduct ${}^{A}r$ is a generating rule of ${}^{A}P$, then label r with "g"; if ${}^{A}r$ is a zombie rule, then label r with "z"; otherwise, i.e., ${}^{A}r$ is a dead rule or r vanishes in the dual reduction, then label r with "i" (for irrelevant). Figure 2 shows some examples.

From G, we construct a full prioritization $\mathcal{P}' = (P, <')$ of $\mathcal{P}$ which proves that A is a preferred answer set, by employing a modified topological sorting procedure as follows.

Algorithm FULL-ORDER

Input: Propositional prioritized program $\mathcal{P} = (P, <)$, answer set $A \in \mathcal{AS}(\mathcal{P})$.

Output: Full prioritization $\mathcal{P}' \in \mathcal{FP}(\mathcal{P})$ such that $A \in \mathcal{PAS}(\mathcal{P}')$, if $A \in \mathcal{PAS}(\mathcal{P})$; otherwise, "no".

Method:

Step 1. Construct graph $G = G(\mathcal{P}, A)$, and initialize $T := \emptyset$, $<' := \emptyset$.

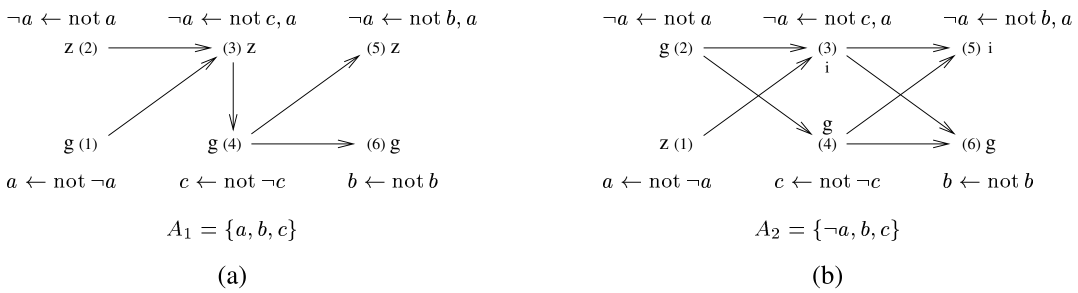

Figure 2: Examples of graph $G(\mathcal{P}, A)$ (transitive edges are omitted)

Step 2. If G is empty, then output $\mathcal{P}' = (P, <')$ and halt;

Step 3. Pick any source of G, i.e., a vertex r which has no incoming edges, such that either r is not labeled "z", or r is defeated by T; if no such r exists, then output "no" and halt.

Step 4. if r is labeled "g", then set $T := T \cup \{h(r)\}$ where $h(r)$ is the head of r;

Step 5. remove r from G, and continue at step 2. ∎

Notice that this algorithm is nondeterministic, since in Step 1, in general different vertices r can be selected, which leads to different outputs.

Example 7.1 Consider the prioritized program $\mathcal{P}$ which is represented by the graph $G(\mathcal{P}, A_1)$ in Figure 2.a, and the answer set A_1. In step 3 of FULL-ORDER, (1) and (2) are sources; since (1) is labeled "g" and (2) is labeled "z" but not defeated by $T = \emptyset$, the algorithm selects (1) and sets in Step 4 $T := \{a\}$. In the next iteration, rule (2) is the only source and selected, as it is now defeated by T; T is not updated. In the next iteration, rule (3) is the only source; however, (3) is labeled "z" and not defeated by $T = \{a\}$. Thus, the algorithm outputs "no" and halts. Observe that A_1 is not a preferred answer set of $\mathcal{P}$.

Now consider the reordered program $\mathcal{P}_1$ represented by the graph in Figure 2.b, and the answer set A_2. In step 3, rules (1) and (2) are the sources; (2) is selected and $T := \{\neg a\}$ in step 4. Thereafter, rule (1) is selected, as it is now defeated by T. In the next round, both (3) and (4) are selectable; suppose (4) is selected first, which results in $T := \{\neg a, c\}$, and thereafter (3) is selected (which leaves T unchanged). Now, both (5) and (6) are selectable; suppose (5) is chosen and finally (6), which effects the update $T := \{\neg a, b, c\}$. Since G is empty now, the algorithm outputs $\mathcal{P}'_1$, where $(2) <' (1) <' (4) <' (3) <' (5) <' (6)$ and halts. As easily checked, A_2 is a preferred answer set of $\mathcal{P}_1$. ∎

The next lemma states that the algorithm works in fact properly.

Lemma 7.2 *Let $\mathcal{P}$ be a propositional prioritized program, and let $A \in \mathcal{AS}(\mathcal{P})$. Then, $A \in \mathcal{PAS}(\mathcal{P})$, if and only if some execution of* FULL-ORDER *outputs a fully prioritized program $\mathcal{P}'$. Moreover, $\mathcal{P}' \in \mathcal{FP}(\mathcal{P})$ and $A \in \mathcal{PAS}(\mathcal{P}')$ hold for every such $\mathcal{P}'$, and either all runs output some $\mathcal{P}'$, or all output "no".*

Now the previously announced result on recognition of preferred answer sets is not hard to prove.

Theorem 7.3 *Let $\mathcal{P} = (P, <)$ be a prioritized finite propositional logic program, and let $A \subseteq Lits$. Then, deciding whether $A \in \mathcal{PAS}(\mathcal{P})$ is polynomial (more precisely, complete for* P*).*

Proof. By Lemma 7.2, $A \in \mathcal{PAS}(\mathcal{P})$ can be decided by checking whether $A \in \mathcal{AS}(\mathcal{P})$ and running a deterministic s of FULL-ORDER on $\mathcal{P}$; the former check is polynomial (Proposition 7.1), and it is not hard to see that any run of FULL-ORDER is polynomial-time bounded. Hence, the problem is in P. Hardness for P (under log-space reductions) is inherited from the subcase of fully prioritized programs (Lemma 7.2). ∎

Thus, model checking is polynomial, and priorities do not affect the complexity. We remark that using proper data structures, algorithm FULL-ORDER can be implemented to run in linear time. Since also $A \in \mathcal{AS}(\mathcal{P})$ is decidable within the same time, recognition of a preferred answer set can be done efficiently.

Let us see whether we have a similar behavior for deciding the consistency of a prioritized program, i.e., the existence of a preferred answer set. As discussed above, this problem is intractable in the general case, which is inherited from the complexity of stable models. However, this does not rule out the possibility that the consistency check and finding some preferred answer set is polynomial, if we have a fully prioritized program; notice that in case of prerequisite-free programs, there is at most one preferred answer set (Lemma 4.1). Unfortunately, also in this case, a polynomial time algorithm is unlikely to be found.

Theorem 7.4 *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.*

The proof of this result is by a reduction from the satisfiability problem. We remark that our reduction is *parsimonious* [35], i.e., the number of solutions is preserved in the reduction. This implies that counting the number of preferred answer sets is a difficult problem. We obtain from Theorems 7.3, 7.4 and well-known results about the satisfiability problem the following side result. The class #P (see [35]) contains the search problems for which the number of solutions of each instance can be expressed by the number of accepting computations of a nondeterministic polynomial time Turing machine.

Corollary 7.5 (proof of Theorem 7.4) *Given a finite propositional fully prioritized program $\mathcal{P} = (P, <)$, counting the number of preferred answer sets of $\mathcal{P}$ is* #P*-complete.*

It is easy to see that in the proof of Theorem 7.4, a satisfying assignment for the clause set $\mathcal{C}$ in the reduction can be easily constructed from any arbitrary preferred answer set of the program $\mathcal{P}$. Hence, computation of an arbitrary satisfying truth assignment is polynomial-time reducible to the computation of a preferred answer set. Therefore, we obtain the following result.

Corollary 7.6 (proof of Theorem 7.4) *Given a finite propositional prioritized program $\mathcal{P} = (P, <)$, computing an arbitrary preferred answer sets of $\mathcal{P}$ is* FNP*-complete.*

Finally, let us turn to the problem of inference. We consider here cautious inference; for brave inference, we obtain dual complexity results.

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 $Cn(\mathcal{P})$.

Like in many other nonmonotonic systems, also in case of prioritized logic programs the complexity of inferencing is related to the complexity of consistency, and can be reduced to the complementary problem. In fact, from Theorem 7.4 it is not difficult to derive the following result.

Theorem 7.7 *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.*

7.2 Weakly preferred answer sets

As we have seen in the previous subsection, adding priorities to programs is computationally speaking well-behaved, in the sense that selecting preferred answer sets among all answer sets under consideration does not add to the complexity of the main computational problems. Let us now see whether this pertains if we consider weakly preferred answer sets instead of strongly preferred ones.

As we will see, weak preference increases the complexity of the main computational tasks on prioritized programs. The reason is that weak preference bears an implicit NP optimization problem, whose computation is slightly harder than NP. However, the increase in complexity is rather small and does not lift the problems to higher regions of the polynomial hierarchy; they stay within the polynomial time closure of NP, i.e, P^{NP} (resp., $\mathrm{FP}^{\mathrm{NP}}$). A precise account of the decisional problems shows that they do in fact belong to $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$; we discuss implications of these results below.

Recall that an answer set A of a prioritized program $\mathcal{P}$ is weakly preferred, if its preference violation degree $pvd_{\mathcal{P}}(A)$ is smallest over the preference violation degrees of all answer sets of $\mathcal{P}$, i.e, coincides with $pvd(\mathcal{P})$. Thus, for the recognition of a weakly preferred answer set, the relationship of $pvd_{\mathcal{P}}(A)$ to $pvd(\mathcal{P})$ is important.

A natural algorithm for deciding whether A is weakly preferred is the following.

Algorithm REC-WAS

Input: Propositional prioritized program $\mathcal{P} = (P, <)$, answer set $A \in \mathcal{AS}(\mathcal{P})$.

Output: “yes”, if $A \in \mathcal{WAS}(\mathcal{P})$; otherwise, “no”.

Method:

Step 1. compute $pvd_{\mathcal{P}}(A)$;

Step 2. compute $pvd(\mathcal{P})$;

Step 3. if $pvd_{\mathcal{P}}(A) = pvd(\mathcal{P})$, then output “yes”, otherwise output “no”. ∎

This algorithm calls subroutines for computing $pvd_{\mathcal{P}}(A)$ and $pvd(\mathcal{P})$, which must be refined. For this aim, let us first see over which values the preference violation degree may range.

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 6.5 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 $<'$. Thus, $pvd_{\mathcal{P}}(A)$, and hence also $pvd(\mathcal{P})$ ranges over the segment $[0, s]$ of the integers.

In terms of a decision problem, 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 7.8 *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 $\mathrm{F}\Delta_2^p[O(\log n)]$.*

A more precise account of computing $pvd_{\mathcal{P}}(A)$ and $pvd(\mathcal{P})$ shows that the problems are complete for subclasses of $\mathrm{F}\Delta_2^p[O(\log n)]$ under appropriate reductions, but we are less interested in such results here.

Thus, in the light of Proposition 7.8, we obtain that algorithm REC-WAS can be implemented such that it runs in polynomial time while making $O(\log n)$ accesses to an NP oracle. As a consequence, recognition of a weakly preferred answer set is in $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$. The question is whether this rather straightforward upper bound can be improved by a more sophisticated algorithm, e.g., by one which uses only a constant number of calls to an NP-oracle, or, even an NP style guess-and-check algorithm. The next theorem tells us that the $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ upper bound is paralleled by a matching lower bound, and hence such an improved algorithm is unlikely to exist.

Theorem 7.9 *Given a prioritized propositional program $\mathcal{P} = (P, <)$ and a set $A \subseteq Lits$ of ground literals, deciding whether $A \in \mathcal{WAS}(\mathcal{P})$ is $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-complete. Hardness for $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ holds even if $<$ is a total ordering and P does not contain strong negation.*

The proof of the hardness part of this theorem, in particular under the asserted restriction, is rather technical and exploits particular properties of the distance measure $d(<, <')$ from the definition of weakly preferred answer sets. Let us try to give an intuitive account of why the problem is that complex, even if the rules are fully prioritized.

For determining $pvd_{\mathcal{P}}(A)$ of any answer set A of $\mathcal{P}$, we must switch as few rules in $<$ as possible such that A becomes a preferred answer set; this means that some appropriate generating rule r' must be moved in front of every zombie rule r which is not defeated along the ordering $<$. However, in general several generating rules r' for defeating r exist; moving the "right" generating rules such that the overall switching cost is minimal amounts to an optimization problem, which does not have a unique solution in general. For example, consider the following simple program:

$$(1)\ b \leftarrow \mathrm{not}\, a, \mathrm{not}\, c \quad (2)\ c \leftarrow \quad (3)\ e \leftarrow \mathrm{not}\, a, \mathrm{not}\, d \quad (4)\ a \leftarrow \quad (5)\ d \leftarrow$$

Assuming as usual that $(i) < (j)$ for $i < j$, after moving $c \leftarrow$ in front of (1) and $d \leftarrow$ in front of (3), the unique answer set $A = \{a, c, d\}$ is preferred. However, moving $a \leftarrow$ in front of (1) is another possibility for making A preferred; both possibilities require three switches. In a more complicated setting, overlapping possibilities must be combined, which turns out to be difficult. Indeed, deciding whether a certain number of rule switches is sufficient for turning an answer set A into a preferred answer set is an intractable problem. (Observe that if computing the value of $pvd_{\mathcal{P}}(A)$ for any A were polynomial, then deciding $A \in \mathcal{WAS}(\mathcal{P})$ would be in co-NP.)

Before we turn to the inference problem, let us consider the problem of actually computing a weakly preferred answer set, rather than recognizing it. It is not hard to see that given a prioritized logic program $\mathcal{P}$, we can compute a weakly preferred answer set of $\mathcal{P}$ in polynomial time with the help of an NP oracle, i.e., the problem is in $\mathrm{FP}^{\mathrm{NP}}$. Indeed, the problem can be solved by computing first $pvd(\mathcal{P})$, and then constructing some $A \in \mathcal{WAS}(\mathcal{P})$ by stepwise extending a partial weakly preferred answer set A' using the oracle. More formally, this algorithm is as follows.

Algorithm COMPUTE-WAS

Input: Propositional prioritized program $\mathcal{P} = (P, <)$.

Output: Some $A \in \mathcal{WAS}(\mathcal{P})$, if $\mathcal{WAS}(\mathcal{P}) \neq \emptyset$; otherwise, "no".

Method:

- **Step 1.** check if $\mathcal{AS}(\mathcal{P}) = \emptyset$; if true, then output "no" and halt, else compute $pvd(\mathcal{P})$, and set $A := \emptyset$, $S := Lits$;
- **Step 2.** If $S = \emptyset$, then output A and halt, else choose some $L \in S$;
- **Step 3.** Check if some $A' \in \mathcal{WAS}(\mathcal{P})$ exists such that $A \cup \{L\} \subseteq A'$; if true, then set $A := A \cup \{L\}$;
- **Step 4.** Set $S := S \setminus \{L\}$, and continue at step 2. ∎

It is easily seen that this algorithm is correct. Moreover, the checks in steps 1 and 3 are in NP; in step 3, this is true because if $pvd(\mathcal{P})$ is known, together with A' a polynomial-size proof of the fact that $A' \in \mathcal{WAS}(\mathcal{P})$ can be guessed and verified in polynomial time.

The reader might guess that computing a weakly preferred answer set is complete for $\mathrm{FP}^{\mathrm{NP}}$; however, this intuition is elusive, and there is no evidence of this fact. The reason is that intuitively, in the second phase of algorithm COMPUTE-WAS (steps 2–4) the access to the NP oracle has much of a deterministic reconstruction of a nondeterministic computation, in which, given $\mathcal{P}$ and $pvd(\mathcal{P})$, a weakly preferred answer set $A \in \mathcal{WAS}(\mathcal{P})$ may be guessed and verified in polynomial time. The above algorithm deterministically reconstructs a successful guess for A step by step; intuitively, the oracle accesses in this phase are "weak" in a sense. On the other hand, the ones in the first phase (step 1) are "strong", since computing $pvd(\mathcal{P})$ requires the full power of $O(\log n)$ many oracle calls. Overall, in total only $O(\log n)$ many strong oracle accesses are made. Problems of a similar characteristics are not known to be complete for $\mathrm{FP}^{\mathrm{NP}}$.

From these observations, one may next guess that the problem is probably in $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$, if we do some clever packing of oracle calls in phase two such that $O(\log n)$ overall oracle calls are sufficient. Unfortunately, this intuition is elusive; in fact, even if the optimal preference violation degree $pvd(\mathcal{P})$ is known, it is not clear how to construct a weakly preferred answer set in polynomial time using only $O(\log n)$ many NP oracle calls.

The precise complexity of computing a weakly preferred answer set can be assessed in terms of the less familiar complexity class FNP//OptP$[O(\log n)]$ introduced in [15, 16]. This class contains the search problems such that for each instance I, an arbitrary solution for I can be computed by a nondeterministic Turing machine in polynomial time, if the machine receives as additional input the value $g(I)$ of a fixed function g from the class OptP$[O(\log n)]$; the latter class contains all functions f, whose value $f(x)$ has $O(\log n)$ many bits, where n is the size of x, and such that $f(x)$ can be obtained as the "optimal" (i.e., minimal or maximal) value computed by some nondeterministic polynomial-time Turig machine in all runs on input x. Well-known problems in OptP$[O(\log n)]$ are e.g. computing the size of a maximum clique in a graph, or computing the size of a smallest vertex cover for a graph (i.e., a subset of the vertices which meets each edge). The precise technical definition of FNP//OptP$[O(\log n)]$ can be found in [16]. Notice that few natural problems complete for this class are known.

It is easy to see that computing an arbitrary weakly preferred answer set of a propositional prioritized program $\mathcal{P} = (P, <)$ belongs to the class FNP//OptP$[O(\log n)]$. Indeed, the function $pvd(\mathcal{P})$ is in OptP$[O(\log n)]$ (with obvious extensions for the case where $\mathcal{P}$ has no weakly preferred answer sets), and if $pvd(\mathcal{P})$ is known, a full prioritization $\mathcal{P}' = (P, <')$ of $\mathcal{P}$ and an $A \in \mathcal{PAS}(\mathcal{P}')$ such that $d(<, <') \leq pvd(\mathcal{P})$ can be nondeterministically generated and verified in polynomial time. On the other hand, it can be shown that computing a weakly preferred answer set for $\mathcal{P}$ is among the hardest problems in FNP//OptP$[O(\log n)]$.

Theorem 7.10 *Given a prioritized propositional program $\mathcal{P} = (P, <)$, computing an arbitrary $A \in \mathcal{WAS}(\mathcal{P})$ is complete for* FNP//OptP$[O(\log n)]$. *Hardness holds even if $<$ is a total ordering and P does not contain strong negation.*

Let us finally consider the inference problem. Analogous to inference from preferred answer sets, 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 $\mathcal{P}$.

For the problem of inference, we obtain a complexity result analogous to the one for model checking.

Theorem 7.11 *Deciding, given a finite prioritized propositional program $\mathcal{P} = (P, <)$ and a literal L, whether $\mathcal{P} \models_w L$ is $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-complete. Hardness for $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ holds even if $<$ is a total order and P does not have strong negation.*

Notice that the proof of the theorem shows that reasoning from weakly preferred answer sets remains a hard problem, even if all answer sets are known and are part of the problem input. Thus, theoretically speaking, by first computing all answer sets and then selecting the weakly preferred ones among them, we might face in the second step still the full intrinsic complexity of the problem.

The results on weakly preferred answer sets can be interpreted as follows. Informally, they show that this semantics is amenable to parallelization, if an NP oracle is available. In fact, it is known that the class $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ coincides with the class $\mathrm{P}^{\mathrm{NP}}_{\parallel}$, in which a polynomial-time bounded Turing machine may access an NP oracle, but all queries must be prepared before the first call; thus, all oracle queries can be solved in parallel. As a consequence, both recognition of a weakly preferred answer set and weak inference from a prioritized logic program can be parallelized to NP. This can be regarded as a positive property, even if we lack efficient parallelization hardware for NP problems to date.

For the computation of a weakly preferred answer set, we obtain a similar parallelization property. However, from the above results it is not known that this problem is in $\mathrm{FP}_{\|}^{\mathrm{NP}}$, the search analogue of $\mathrm{P}_{\|}^{\mathrm{NP}}$. In fact, the class $\mathrm{FP}_{\|}^{\mathrm{NP}}$ is contained in FNP//OptP[$O(\log n)$], but it is believed that this inclusion is strict. In particular, no FNP//OptP[$O(\log n)$]-complete problem is known to be in $\mathrm{FP}_{\|}^{\mathrm{NP}}$. Nonetheless, it is known that the problems in $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ are in $\mathrm{RP} \cdot \mathrm{FP}_{\|}^{\mathrm{NP}}$, which is a randomized s of $\mathrm{FP}_{\|}^{\mathrm{NP}}$; informally, this class contains the problems for which a solution can be randomly generated with very high probability in polynomial time, if a call to a problem in $\mathrm{FP}_{\|}^{\mathrm{NP}}$ is admitted. Thus, a weakly preferred answer set can be randomly computed with high probability using *parallel* NP oracle calls; since the success probability can be pushed higher than the reliability of hardware, this is in practice as good as nonrandomized computation.

7.3 Nonground programs

Before closing this section, we briefly address the complexity of preferred and weakly preferred answer sets for nonground programs.

For function-free programs, the complexities of the decisional problems from above intuitively increase as usual by one exponential, from P to EXPTIME, NP to NEXPTIME, etc. A particular interesting case is inference from the weakly preferred answer sets, however. The complexity increases in this case to the class $\mathrm{PSPACE}^{\mathrm{NP}}$, i.e., polynomial space with an oracle for NP where the oracle space is unbounded. Indeed, the class $\mathrm{PSPACE}^{\mathrm{NP}}$ is the exponential analogue of $\mathrm{LOGSPACE}^{\mathrm{NP}}$, and $\mathrm{LOGSPACE}^{\mathrm{NP}}$ coincides with $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ (see e.g. [70]). Notice that that membership algorithm for $\mathcal{P} \models_w L$ which we have described in the proof of Theorem 7.11 actually can be implemented to run in logarithmic space; properly adapted for function-free programs (writing the ground instance $\mathcal{P}^*$ of the input $\mathcal{P}$ to the query tape instead of nonground $\mathcal{P}$), it works in polynomial space. However, we omit the technical details and do not investigate into upgrading complexity results; this can be done utilizing the methods developed in [24, 33]. For the search problems, we have to respect that the output size is in general exponential in the input size; the study of respective complexity classes is less developed, and we do not head in this direction here. However, we may expect that an analogous exponential complexity increase is present with function-free programs.

For programs with functions, we obtain from the complexity of stable models [65, 64, 45] that preferred answer set semantics can express Π_1^1-complete sets over the Herbrand universe generated by the program, and that existence of a preferred answer set can express Σ_1^1-complete sets on this universe. On the other hand, Σ_2^1 resp. Π_2^1 is an upper bound for deciding existence of a preferred answer set and inference from the preferred answer sets, respectively; an answer set A with a proper well-ordering $<$ such that $\lambda_{\mathcal{P}}(A) = A$ can be expressed in Σ_2^1. It remains to see, however, in which cases the same upper bounds as for stable models, i.e., Π_1^1 and Σ_1^1, can be obtained. For weakly preferred answer sets, the complexity in the case with functions may be higher. These interesting issues are left for further study.

8 An Application: Qualitative Decision Making

In this section we want to discuss how our approach can be used to reduce qualitative decision making to answering queries to a knowledge base. Assume you want to buy a car. You have collected the following information about different types of cars:

$$expensive(Chevrolet), safe(Chevrolet), safe(Volvo), nice(Porsche), fast(Porsche)$$

Your decision which car to buy is based on different criteria. We can use rules corresponding to normal defaults [60] in order to represent the properties which you consider relevant. Let's assume you like fast and nice cars. On the other hand, your budget does not allow you to purchase a very expensive car. Moreover, you have to take your wife's wishes into account, and she insists on a car which is known to be safe.

(1) $\neg buy(x) \leftarrow \text{not}\, buy(x),\ expensive(x)$
(2) $buy(x) \leftarrow \text{not}\, \neg buy(x),\ safe(x)$
(3) $buy(x) \leftarrow \text{not}\, \neg buy(x),\ nice(x)$
(4) $buy(x) \leftarrow \text{not}\, \neg buy(x),\ fast(x)$

Since buying more than one car is out of the question these different decision criteria may obviously lead to conflict, and a preference ordering is necessary. Since there is not much you can do about your restricted budget, rule (1) gets highest preference. Moreover, since your wife is very concerned about safety you better give (2) higher priority than (3) and (4). Since there is tremendous traffic on highways in your area, (3) is more important to you than (4). This means we have

$$(1) < (2) < (3) < (4)$$

We still have to represent that you cannot afford more than one car. A straightforward idea would be to use the rule

(0) $\neg buy(y) \leftarrow buy(x),\ x \neq y$

with highest priority. Unfortunately, this does not work. The reason is that the high priority of the instances of this rule would allow us to defeat instances of (2), (3) and (4) even if this is not intended. In our example we would obtain two preferred answer sets, the intended one containing $buy(Volvo)$, but also an unintended one (at least from your wife's point of view) containing $buy(Porsche)$. In the latter, the instance of (2) with $x = Volvo$ would be defeated by the instance of (0) with $x = Porsche$ and $y = Volvo$.

To represent our problem adequately, we have to make sure that the consequences of a certain decision, namely that certain cars are not purchased, do not have higher priority than the decision itself, that is, the decision to buy a specific car. Instead of adding the single rule (0) we therefore represent our criteria for buying a car as pairs of rules. To each rule of the form

(r) $buy(x) \leftarrow \text{not}\, \neg buy(x),\ c_1(x),\ \ldots,\ c_n(x)$

we add a second one of the form

(r') $\neg buy(y) \leftarrow c_1(x),\ \ldots,\ c_n(x),\ buy(x),\ x \neq y$

with the same priority as (r). In our example we have to add

(2') $\neg buy(y) \leftarrow safe(x),\ buy(x),\ x \neq y$
(3') $\neg buy(y) \leftarrow nice(x),\ buy(x),\ x \neq y$
(4') $\neg buy(y) \leftarrow fast(x),\ buy(x),\ x \neq y$

and use the following preferences

$$(1) < \{(2),(2')\} < \{(3),(3')\} < \{(4),(4')\}$$

Given this information we obtain a single preferred answer set containing $buy(Volvo)$.

This leaves you somewhat dissatisfied since you really like the Porsche. You might try to convince your wife that in case a car is both nice and fast you would have heard about any safety problems. That is, you would like to add the following pair of rules:

(1.5) $buy(x) \leftarrow \text{not}\, \neg buy(x),\ \text{not}\, \neg safe(x),\ nice(x),\ fast(x)$

$$(1.5') \qquad \neg buy(y) \leftarrow \text{not}\, \neg safe(x),\, nice(x),\, fast(x),\, buy(x),\, x \neq y$$

If your wife accepts this rule with preferences $\{(1.5), (1.5')\} < \{(2), (2')\}$ you are happy since now the single preferred answer set contains $buy(Porsche)$.

9 Related Work

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. In this section, we discuss (in alphabetical order) their relationships to our approach. We also compare our work with a proposal by Delgrande and Schaub in which prioritized default theries are compiled to standard default theories [18].

Analyti and Pramanik. Yet another semantics for extended logic programs, called *reliable semantics* (RS), has been proposed by Analyti and Pramanik in [2]. Their approach is based upon and generalizes the well-founded semantics of extended logic programs. The set of consequences is given by its *reliable model*, which is computable as the least fixpoint of a monotonic operator. RS was developed to obtain meaningful consequences from programs that may derive contrary literals L and $\neg L$, assuming that rules have different reliability. A strict partial order $<$ is imposed on the rules where $r < r'$ reads "r is less reliable than r'." Each rule r is assigned a suspect set S_r of literals from its body, and partial models (which may contain weakly negated literals) are subject to constraints (i.e., rules with empty heads); if applying r violates a constraint, then informally r and rules deriving a literal in S_r are viewed "suspect" for this violation, and the reliability ordering is used for resolving the conflict.

Apparently, the framework of Analyti and Pramanik is quite different from our prioritized logic programs. Our approach views preference at an abstract level, rather than as reliability information, and is not tailored as a contradiction removal system. As for the principles, since RS generalizes the well-founded semantics, it literally violates Principle II in general; a variant of RS which assigns every program a set of r-stable models also violates Principle I as follows from a correspondence to partial stable models of normal logic programs. However, this assessment is not fully appropriate, since reliability of rules is only relevant for resolving head-conflicts of rules. A proper reformulation of the principle I and II for this task remains for further study.

Brewka. In the article [9], 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 did not want to complicate matters and 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 [8]: if preferences are represented in the language, then an answer set A is preferred if there is a total order $<$ of the program rules such that $<$ is compatible with the preference information in A and $\lambda_<(A) = A$. Intuitively, $<$ is compatible with A if the addition of a syntactic representation of $<$ to A does not lead to inconsistency.

Buccafurri et al. and Laenens et al. An approach that is closer in spirit to ours is ordered logic programming, which has been proposed by Laenens and Vermeir [40] and further developed e.g. in [27, 12, 14]. An ordered logic program is a set of components forming an inheritance hierarchy, where each component consists of a set of rules. The inheritance hierarchy, which is modeled as a strict partial ordering, is used to settle conflicts among rules: rules from components lower in the hierarchy have preference over those from components higher up in the hierarchy, since the former components are considered more specific. Different semantics of ordered logic programs, akin to the standard semantics of traditional logic programming have been made; in particular, an appealing definition of a stable model for an ordered logic program was given in [12]. 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 provide only one kind of negation, namely strong negation. Negation as failure is not in the language, but can be simulated by using special predicates and additional components on the hierarchy [14].
3. On the other hand, disjunctive ordered logic programs [14] allow disjunction in rule heads, which has not been considered in this paper (our approach can be easily extended to this case, though).

As for the principles I and II, Buccafurri et al. report that ordered logic programming and its extension with disjunction satisfy these principles, recast to the framework of disjunctive ordered logic [14].

Delgrande and Schaub. A further approach for adding priorities to default theories has been presented by Delgrande and Schaub [18]. 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 Reiter [60] default theory $T = (D', W')$, by naming the defaults and introducing special purpose predicates. The latter include a binary predicate $\prec(\delta, \delta')$ for expressing priority information, $ok(\delta)$ for controlling application of rule δ, $bl(\delta)$ for expressing that δ is blocked, and $ap(\delta)$ for expressing that δ is applied. The control of rule application is expressed by a first-order formula in W', which essentially states that consideration (i.e., potential application) of a rule δ is ok (represented by $ok(\delta)$), if every rule δ' which has higher priority is either applied or blocked (expressed by facts $ap(\delta')$ and $bl(\delta')$, respectively, each of which can only be derived if $ok(\delta')$ is true). The extensions of the ordered default theory $(W, D, <)$ are the extensions of T, restricted to the language of (W, D); these are always extensions of the unordered default theory (W, D). An extended approach allows for dynamic preferences similar as in [32, 58, 63, 71], by supporting the explicit representation of preference information between defaults in W and D.

Delgrande and Schaub's proposal is different from ours. Like Gelfond and Son [32], they translate priorities into the object language of a non-prioritized underlying formalism, rather than treating priorities at the meta-level. The naming of defaults and use of first-order domain closure axioms as described in [18] assume a finite set of defaults. Furthermore, we reduce a partial ordering to all well-orderings for defining preferred answer sets, which is not the case in the approach of Delgrande and Schaub, where preferred extensions are constructed using the priority information just as given by $<$.

As for our Principles I and II, it appears (Schaub, personal communication) that they are both satisfied by the approach of Delgrande and Schaub. Observe, however, that they treat priorities as second-class entities, while other approaches, including ours, handle priorities as first-class entities. In the group of the latter considered in this paper, our approach to preferred answer sets is the only one which satisfies both Principle I and II.

Moreover, Delgrande and Schaub do not obtain satisfactory results in all cases. This is shown e.g. by the default theory in Example 3.1 from Section 3.2, where Schaub and Delgrande select no extension; this is counterintuitive, since $E_1 = Th(\{a, b\})$ is expected to be preferred. Similarly, no extension is selected for the variant of the birds & penguins example described in the next paragraph, if the priorities $(3) < (4) < (5)$ are adopted. An intuitive explanation for this behavior is that Delgrande and Schaub's approach can be viewed as a "one-pass" test on an extension E in which the rules are considered in the order of decreasing priorities. If a rule is encountered which is neither applicable nor blocked, then the extension E is rejected. On the other hand, we use a kind of look-ahead strategy, in which a rule of lower priority may be applied to establish the precondition of a yet inapplicable rule. This way, intuitively more extensions can be accepted. The previous examples show that in the framework of Delgrande and Schaub dependencies between rules must be carefully respected for assigning priorities.

Gelfond and Son. In a recent paper, Gelfond and Son have also tackled the problem of adding priorities on defaults expressing normality in the language of extended logic programs [32]. Similar to [18, 63, 58, 72], their approach foresees the specification of preference over rules in the object language. An important difference to our approach is that rules are, similar as in [52, 58, 63], 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.

Gelfond and Son's approach has a multi-sorted logical language which has constants for individuals, definite rules, and default rules for expressing statements of the form "If $l_1, \ldots, l_m$ are true, then normally l_0 is true," functions and relations for the domain as well as special predicates for defining rules and expressing preference. For example, the previous default rule is expressed by the formula

$$default(d, l_0, [l_1, \ldots, l_m])$$

where d is the name of the default rule; here, $[l_1, \ldots, l_m]$ is Prolog-like list notation. Informally, this default amounts to the rule $l_0 \leftarrow l_1, \ldots, l_m, \mathrm{not}\, \neg l_0$ in extended logic programming. Moreover, the language allows to express that two default rules conflict (e.g., default rules with opposite literals in the head may be declared as conflicting). Like preferences, conflicts can be declared dynamically by means of rules.

The semantics of the language is defined in terms of a transformation of any program $\mathcal{P}$ into an extended logic program $t(\mathcal{P})$ whose answer sets are, roughly speaking, cast into answer sets of the program $\mathcal{P}$. The transformation is basically a meta-interpreter for programs. Further extensions of the language by allowing defeasible literals in bodies of default rules and exceptions to defaults are discussed.

An important feature of Gelfond and Son's approach is an explicit distinction between provability by default and "strong" provability in the construction of an answer set; a literal is strongly provable, if it is provable from definite rules (whose application requires that all literals in the body are strongly provable), and a literal is provable by default, if it is provable from default rules, where the rule bodies hold by default.

By this feature, Gelfond and Son's approach and ours are fundamentally different in terms of their underlying philosophical principles. The former expresses a form of graded belief: strongly provable literals have a higher degree of belief than those provable by default. Our approach is opposite and rejects such a view, as it takes the point that if a literal is proved, it should be used like any other proved literal, regardless of the way of derivation. Therefore, a comparison of our approach to the one of Gelfond and Son must be taken with care.

Gelfond and Son have compared a suitable instance of their framework (which is more general than ours) to preferred answer sets in [32].[4] They have shown that for the syntacic subclass of static and hierarchical domain descriptions (programs) with basic preference relations, inference of literals under their and perferred answer set semantics coincide. Informally, this class contains a kind of stratified programs, where the head of a default rule d is in a higher stratum than the head of every other default rule d' which occurs in the body of d. Moreover, conflicts and preference are only expressible on defaults d and d' with contrary heads, and such heads must belong to the same stratum. Further conditions on the default-free part of the program, including consistency, are imposed. An example of a program in this class is the following variant of the birds & penguins example:

(1)	$bird \leftarrow$
(2)	$swims \leftarrow$
(3)	$\neg flies \leftarrow \mathrm{not}\, flies,\, peng$
(4)	$flies \leftarrow \mathrm{not}\, \neg flies,\, bird$
(5)	$peng \leftarrow \mathrm{not}\, \neg peng,\, bird, swims$

Here, (3) and (4) are in the same stratum and (5) is in a lower stratum; the default-free part (1)–(2) is obviously consistent. This program has two answer sets, namely $A_1 = \{bird,\, swims,\, flies,\, peng\}$, and

[4] In fact, the comparison relies on a preliminary definition of preferred answer sets which used the simplified definition of the sequence S_α as discussed before Example 5.3. However, for programs with normality defaults of the form considered in [32], the preliminary and present definition of preferred answer sets are equivalent (see also the paragraph after Example 5.3).

$A_2 = \{bird, swims, \neg flies, peng\}$. If we assume the (single) preference $(3) < (4)$, then our approach (as well as Gelfond and Son's) yields A_2 as the unique preferred answer set, which seems intuitive. Note that A_2 remains the unique preferred answer set, regardless of a priority assigned to rule (5); other approaches, including [3, 8, 48], select A_1 under priorities $(3) < (4) < (5)$.

The equivalence result of Gelfond and Son, which establishes a correspondence between preferred answer sets and the answer sets of a normalized version of the domain description, allows to mutually exploit results from [32] and the present paper. Our results entail properties of corresponding domain descriptions in Gelfond and Son's framework, while their transformation $t(\mathcal{P})$ can be utilized for readily reducing the inference problem for a class of prioritized logic programs to the inference problem for extended logic programs. It remains to be seen whether the equivalence result can be extended to further classes of programs.

Inoue and Sakama. Preferred answer sets have previously been defined by Sakama and Inoue [63]. In their approach, a preference relation on the answer sets of a generalized extended disjunctive logic program is defined, which is derived from a reflexive and transitive preference relation on the set of (possibly weakly negated) classical literals $Lits$. Informally, in their approach an answer set A_1 is at least as preferable as an answer set A_2, if there are literals $a \in A_1 \setminus A_2$ and $b \in A_2 \setminus A_1$ such that a has at least the priority of b (expressed by $b \preceq a$), and there is no literal $c \in A_2 \setminus A_1$ which has strictly higher priority than a (i.e., $a \preceq c$ and $c \not\preceq a$); an answer set A_1 is preferred, if there does not exist another answer set A_2 which is strictly preferable over A_1. As demonstrated in [63], this approach is suited for selecting answer sets based on a notion of likelihood for (possibly negated) facts being true, and provides flexibility to dynamically adapt this likelihood depending on the particular context. Moreover, it is shown that several forms of common sense reasoning, including a default reasoning system, can be expressed in this framework.

Clearly, Inoue and Sakama's approach, whose elements remind of the perfect model semantics for disjunctive logic programs, was devised for a goal different from ours, and is applicable to a wide range of problems. There is a suggestive straightforward syntactic translation of our framework to the one in [63] as follows: For each rule r of the form $Head \leftarrow Body$ in an extended logic program P, introduce an atom a_r and add to the program the rule $a_r \leftarrow Body$. The only nontrivial preferences on the literals are those which reflect rule priorities, i.e., if $r < r'$, then $a_{r'} \preceq a_r$ holds. However, Inoue and Sakama's preferred answer sets of the transformed program, cast to the language of P, do not coincide with our preferred answer sets of $(P, <)$ in general. In particular, for the program in Example 3.2, Principle II is violated by this transformation.

Inoue and Sakama formalize in [63, Section 3.2] another variant of prioritized default reasoning in their framework, which is based on Poole's approach to default reasoning [55] rather than on Reiter's default logic. In this approach, like in [52, 32, 58] definite rules and defeasible rules are distinguished, and an extension is built by selecting a maximal set of defeasible rules which is consistent with the definite rules. In the formalization of [63, Section 3.2], for each defeasible rule $r : Head \leftarrow Body$ an atom δ_r is introduced, and r is replaced by the two rules $Head \leftarrow Body, \delta_r$ and $\delta_r \mid not\, \delta_r \leftarrow$. The nontrivial priorities include $not\, \delta_r \preceq \delta_r$ for each rule, i.e., applying the rule is preferred, and the priorities between rules.

In the case of Example 3.2, the approach selects both answer sets as preferred ones. The reason is that all three rules (1)–(3) may be jointly chosen such that some answer set exists, and all these answer sets are equally preferred. In general, if a program has some answer set if all defeasible rules are included as regular ones, then, regardless of the priority information imposed on the defeasible rules, all answer sets are selected as preferred ones. This might not always be intended; it does not happen, however, if the rules – as in Poole's approach to default logic – do not use negation as failure in rule bodies. Finally, we remark that Inoue and Sakama's approach to prioritized logic programs, similar as our concept of weakly preferred answer sets, always yields some preferred answer set provided that any answer set of the program exists. On the other hand, different from ours, their approach is nonmonotonic with respect to priorities: adding priority information may enlarge the collection of the preferred answer sets, and result in the loss of conclusions.

Kowalski and Sadri. In [38], 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. Technically, this is achieved by a redefinition of answer sets. It turns out that the original answer sets remain answer sets according to the new definition whenever they are consistent. 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. It is difficult to see how, for instance, exceptions of exceptions can be represented.

3. 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.

Nute. An early approach to priorities on rules of a logic program is due to Nute in his formalization of defeasible reasoning [50, 51], which has been further developed later on (see [52]). Nute's approach divides the rules into absolute and defeasible ones, the former being rules which apply under any circumstances and the latter being rules which are applied by default; such rules also express subjunctive implications. Central to Nute's approach is an ordering of the rules by specificity and superiority of absolute rules. Rules with contradictory conclusions are considered competitive, and a general superiority ordering between such rules is defined based on their logical properties. A defeasible reasoning process is defined by a modification of SLD resolution, in which superiority of rules is taken into account.

From this outline, it is evident that Nute's approach is quite different from ours and other approaches discussed. Indeed, in his approach, prioritization is implicitly determined by the logical entrenchment of the rules in the theory represented by the program, and user-defined priority specification is not supported; moreover, the evaluation is proof-theoretic (and thus to a great deal operational) rather than declarative. Thus, Nute's approach offers limited capability of expressing priorities, which is not sufficient for the need in practice. Moreover, like with similar approaches, the operational nature of his approach makes it difficult to assess the proper working on a complex program.

Pradhan and Minker. In [56] 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. Three different semantics of priorities are presented, two of them turn out to be equivalent.

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. For instance, it is difficult to see how our qualitative decision making example could be represented based on preferences among atoms rather than rules.

Prakken and Sartor. Another approach to incorporate priorities into extended logic programs has been presented in [58]. The authors develop a semantics which is based on argumentation-theoretic foundations in the spirit of Dung's semantics for extended logic programming [21]. Similar as [52], Prakken and Sartor distinguish defeasible rules (which may contain assumptions) and strict (non-defeasible) rules, which must not contain assumptions; note that assumption-free rules can be either defeasible or strict, which is not possible in our framework. Priorities are expressed in [58] by means of a strict partial order on the set of rules. The basic concept of the approach is an *argument*, which is intuitively a sequence of rules constituting a proof of a literal. The semantics of a program is defined in terms of the set of justifiable arguments, which contains intuitively those arguments (and only those) which must be accepted from the program, and such that there is no circular dependency among the arguments. This set is the least fixpoint of a monotonic operator; thus, the semantics is deterministic in the sense that it assigns a unique belief set to each program and is therefore closer to well-founded than to answer set semantics.

A further difference between the approach in [58] and ours is that the former uses priorities to resolve conflicts that emerge from arguments which arrive at opposite conclusions, rather than for choosing a rule out of a set of (not necessarily conflicting) rules for application. This is guided by the observation in [58] that in the legal domain, priorities are mainly used for resolving conflicts between rules that arrive at opposite conclusions. Furthermore, conflict resolution is defined in [58] in terms of preferability among sets of rules, rather than pairs of rules. Prakken and Sartor generalize their approach by introducing also defeasible priorities. There is no counterpart for this in our framework.

Zhang and Foo. In two subsequent papers [72, 71], 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 answers 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. Roughly speaking, if we have priority $r < r'$ among rules of $\mathcal{P}$, then r' will be ignored only if r' is defeated by $\mathcal{P}' = \mathcal{P} \setminus \{r'\}$, i.e., every answer set of $\mathcal{P}'$ defeats r'; otherwise, r' is kept in $\mathcal{P}$. Rules r' are removed from $\mathcal{P}$ subsequently until no longer possible (see [71] for more details and formal definitions).

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; As we have seen above, this is not the case in our approach. For example, consider the following simple program P:

$$
\begin{array}{ll}
(1) & \neg p \leftarrow \mathrm{not}\, p \\
(2) & p \leftarrow \mathrm{not}\, q
\end{array}
$$

in which (1) is preferred over (2). The program P has the unique answer set $A = \{p\}$, and Zhang and Foo's semantics returns A as the unique preferred answer set. However, if we take the view that priorities are not defeasible, this is not acceptable straight away. The rule (1) has higher priority than the rule (2), and not applying (1) because it is defeated after application of the lower ranked rule (2) seems hardly defensible; therefore, (1) should be applied, and also (2) should be applied because application of (1) does not defeat (2). The joint application of (1) and (2) results in an inconsistency, however, and thus no answer set for the given rule preference exists. Observe that if we change the priority ordering, then $A = \{p\}$ is the unique preferred answer set; this is exactly what is achieved by our preferred and weakly preferred semantics.

A moment of reflection should convince the reader that this is indeed the desired behavior. The first rule is a CWA default which tells that p is explicitly false if it is not provable, which is more important than the rest of the program; in fact, by this high priority, we must conclude that p is false, as there are no even more important rules which allow to prove p. Such a CWA default should have low priority, however, since a proof of $\neg p$

by failure to prove p is weaker than an explicit proof in terms of other rules. Therefore, the priorities should actually be reversed. On the other hand, if preference is viewed as "use (1) over (2) whenever possible", i.e., the priorities are considered defeasible, then the answer of Zhang and Foo's semantics is meaningful.

Another example in which Zhang and Foo's semantics differs from ours is the program discussed in Example 5.3. Recall that this program (refer to it as $\mathcal{P}$) has the two answer sets $A_1 = \{p, q\}$ and $A_2 = \{\neg p, q\}$, none of which is preferred in our definition. However, according to the definition of Zhang and Foo, the answer set A_2 is preferred over A_1. This seems hardly defensible, since the highest priority rule of the program, $p \leftarrow \mathrm{not}\, q$, is executable if no other rules have been considered so far; hence, p should be concluded, which rules out A_2 as a maximally preferred answer set.

Furthermore, Zhang and Foo's concept of preferred answer set does not obey Principle II. This can be shown by the same example which demonstrates the failure of Principle II for our weak preferred answer set semantics in the proof of Proposition 6.1; in fact, our weak preferred answer sets coincide there with the preferred answer sets of Zhang and Foo.

However, the concept is also different from weakly preferred answer sets, as for the previously considered program $\mathcal{P}$, both answer sets A_1 and A_2 are weakly preferred. Notice that if we would use a stronger concept of weak preference, in which switches of rules are weighted by priorities, then A_1 would be ruled out; this would not be much intuitive.

Zhang and Foo further introduce a concept of dynamic preferred answer sets, which allows for expressing dynamic rule preference, i.e., the preference ordering of the rules can be specified by additional meta-rules represented in the object language. The semantics of such dynamic prioritized programs is then based on preferred answer sets for prioritized logic programs. This approach is very general, but we believe that further investigation is needed to clarify relevant aspects. An intuitive understanding of the global semantics of a program, due to its involved operational definition, seems to require quite some acquaintance with the approach. Furthermore, little is known about the properties of this semantics, apart from the obvious requirement that the preferred answer sets should select a subset of the answer sets of the prioritized program.

10 Further Work and Conclusion

In this paper, we have presented an approach for adding priorities to extended logic programs. Our approach is based on the ideas that applying default rules means to jump to conclusions, and that rules must be applied in an order compatible with the priorities. We have formalized this intuition by introducing the dual reduct of an program, on which the obedience to the preference information which is declared on the rules can be verified by means of a simple operator; the preferred answer sets are those answer sets which amount to fixpoints of this operator. To overcome the situation in which, due to inappropriate priority information, no answer set is preferred, we have proposed a concept of weakly preferred answer set, which can be seen as an approximation of a preferred answer set.

In order to test the coherence of the preferred answer set approach, we have proposed two principles which, as we believe, any formalism for prioritized knowledge representation should satisfy. We have shown that preferred answer sets satisfy these principles, while a number of other approaches to priorities on logic programs, many of which are inherited from prioritized versions of default logic [60], do not satisfy these properties. This means that our approach avoids problems which are present in these other approaches.

Furthermore, we have investigated the computational properties of our approach. We have described algorithms for some of the computational tasks, and we accurately determined the computational complexity of the main reasoning problems. It appeared that our approach has rather benign computationally properties, in the sense that the preferred answer sets do not increase the complexity compared to standard answer sets, and weakly preferred answer sets increase the complexity only marginally.

Several issues remain for further work. One such issue is the extension to a syntactically and semantically richer framework, which offers increased expressive capabilities. Such capabilities can be provided by further extensions of logic programming beyond classical and default negation. For example, one such extension is disjunction in the rule heads, which has been considered in the context of stable and answer set semantics quickly after the invention of the stable semantics [59, 30]. From the technical side, a generalization of our approach to disjunctive extended logic programs appears to be rather easy. The definition of dual reduct need not be changed, and the definition of $C_R(S)$ has to take disjunctive rule heads into account; this can be handled similar to the construction of a model of a disjunctive logic program. Concerning the computational impacts, we may expect an increase of the complexity by one level in the polynomial hierarchy in the propositional case, analogous to the increase for answer set semantics [22].

Another extension is from extended logic programs to the full language of Reiter's default logic; the approach described in this paper can be gracefully generalized to this setting [10].

Further possible extensions of the present work are constructs for expressing dynamic preferences in the framework. Several of the related approaches which we have discussed in Sections 3 and 9 foresee a possibility of explicitly representing preference between rules at the level of the object language, such that rule preferences may dynamically depend on a given context. As we have briefly mentioned above, there are no principal obstacles to extending our approach in this direction, such that priority information comes dynamically into play. In the extended approach, the priority information needed for evaluating the operator $\lambda_R(A)$ on the dual program reduct is extracted from the answer set A, rather than taken from the meta-level.

For the implementation of our approach, we can take benefit from the complexity results which we have established. They unveil a close computational relationship of prioritized logic programs to recent extensions of the stable model semantics by constraints [13]. In the quoted paper, the authors define a concept of weak constraint, which as opposed to traditional a constraint may be violated in a model; however, globally as many constraints as possible should be satisfied. As shown in [13], stable models with weak constraints allow for expressing problems which are $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-complete; hence, by our complexity results, a polynomial-time translation of weakly preferred answer set semantics into stable models with weak constraints is feasible. An experimental implementation of preferred and weakly preferred answer sets on top of the deductive reasoning system `dlv` [25, 26] is planned for the future.

Acknowledgments. The authors thank M. Gelfond, H. Herre, K. Inoue, N. Leone, Ch. Sakama, T. Schaub, and N. Zhang for helpful and clarifying discussions on the topic of the paper. In particular, we appreciate the many comments of N. Leone and K. Inoue. Furthermore, we appreciate the comments of the anonymous referees pointing out corrections and improvements. Finally, we are grateful to V. Marek for bibliographic hints on well-orderings, and to H. Blair and A. Nerode for discussions on this subject.

This work was partially supported by the Austrian Science Fund Project N Z29-INF.

A Appendix: Proofs

For convenience, we use in the following the notation $R < S$, where R and S are sets of rules and $<$ is an ordering, as a shorthand for the property that $r < r'$ holds for all rules r in R and r' in S.

Lemma 5.2 *Let $\mathcal{R} = (R, <)$ be a fully prioritized ground rule base, and let $A \in \mathcal{AS}(\mathcal{R})$. Then, the following are equivalent:*

(i) *A is a preferred answer set of $\mathcal{R}$.*

(ii) *For every $r \in R$, the reduct ${}^{A}r$ is defined and fires in the construction of $\lambda_{\mathcal{R}}(A)$, if and only if r is a generating rule of A.*

Proof. ($\Leftarrow$). If (ii) is true, then clearly $A = \lambda_{\mathcal{R}}(A)$, which means A is a preferred answer set of $\mathcal{R}$.
($\Rightarrow$). Assume that (ii) is false; hence, there are two cases.

(1) r is a rule which is not applied in A, but ${}^{A}r$ fires in the construction of $\lambda_{\mathcal{R}}(A)$. Since ${}^{A}r \in {}^{A}\mathcal{R}$, we have $pre(r) \subseteq A$. Now if $head(r) \in A$ then, according to the construction of $C_{\mathcal{P}}$, r cannot be defeated in A. Hence r must be applied in A, which is contradiction to our assumption. Otherwise, if $head(r) \notin A$, then A is not a fixpoint of $\lambda_{\mathcal{R}}$, and thus not a preferred answer set, which is a contradiction.

(2) r is applied in A, but ${}^{A}r$ does not fire in the construction of $\lambda_{\mathcal{R}}(A)$. Then, $\lambda_{\mathcal{R}}(A)$ must contain a literal defeating r. This literal is not contained in A, which implies that A is not a fixpoint of $\lambda_{\mathcal{R}}$ and thus not a preferred answer set, a contradiction. ∎

Lemma 6.8 *Let $A \in \mathcal{AS}(\mathcal{P})$, where $\mathcal{P} = (P, <)$ is a fully prioritized finite ground program, and let $<'$ be an optimal ordering for A. If τ_i from the canonical switching sequence σ of $<'$ is nonvoid, then $<_{i-1}$ has at position i a non-defeated zombie.*

Proof. Suppose that τ_i moves a rule r' to position i and $<_{i-1}$ has at this position a rule r which is either not a zombie or already defeated. We derive a contradiction. The last step of τ_i, $(i, i+1)$, switches r and r'; in $<_i$, rules r' and r are at positions i and $i+1$, respectively. Observe that since r' is in its final position for $<'$, either r' is not a zombie, or it is defeated in $<_{i-1}$ by some rule at position $< i$.

Consider the ordering $<''$ which coincides with $<_{i-1}$ up to position i, has r' at position $i+1$, and such that its rest at positions $i+2, i+3, \ldots$ coincides with the tail of $<'$ at positions $i+1, i+2, \ldots$ after removal of r. Then, the canonical switching sequences transforming $<$ into $<'$, respectively $\sigma' = \tau'_0, \ldots, \tau'_k$ and $\sigma'' = \tau''_0, \ldots, \tau''_k$, are related as follows. Let j be the position of r in $<'$. Then, $\tau''_h = \tau'_h$, for all $h = 0, \ldots, i-1$; τ''_i is void; $\tau''_h = \tau'_{h-1} \setminus (h-1, h)$, for all $h = i+1, \ldots, j$; and $\tau''_h = \tau'_h$, for all $h = j+1, \ldots, k$. Consequently, the length of σ'' satisfies

$$|\sigma''| \;=\; \sum_{h=0}^{k} |\tau''_h| \;=\; \sum_{h=0}^{i-1} |\tau'_h| + 0 + \sum_{h=i+1}^{j} (|\tau'_h| - 1) + \sum_{h=j+1}^{k} |\tau'_h| \;\leq\; \sum_{h=0}^{k} |\tau'_h| - 1 \;=\; |\sigma'| - 1$$

Hence, by Proposition 6.7, $d(<, <'') < d(<, <')$. Since clearly $A \in \mathcal{PAS}(P, <'')$, it follows that $<'$ is not an optimal ordering for A, which is a contradiction. ∎

Theorem 6.9 *Let $A \in \mathcal{AS}(\mathcal{P})$, where $\mathcal{P} = (P, <)$ is a fully prioritized finite ground program. Then, there exists an optimal ordering $<'$ for A such that τ_i from the canonical switching sequence σ of $<'$ is nonvoid if and only if $<_{i-1}$ has at position i a non-defeated zombie z, and every nonvoid τ_i moves a rule defeating z to position i.*

Proof. (Sketch) We show that given an optimal ordering $<'$ which satisfies the property for all $j = 0, \ldots, i-1$, we can construct another optimal ordering $<''$ which satisfies the property for all $j = 0, \ldots, i$. By repeated application for $i = 0, 1, 2, \ldots$ we obtain the result.

From Lemma 6.8, it follows that τ_i is nonvoid if and only if $<_{i-1}$ has a nondefeated zombie z at position i. Let r be the rule moved by τ_i to position i. Suppose r does not defeat z. Then, j' and i' exist, $i < j' < i'$, such that z is at position i' in $<'$ and the leftmost killer of z, denoted g, is at position j' in $<'$.

Let $<''$ be the ordering which coincides with $<'$ on positions $0, 1, 2, \ldots, i-1$, has at positions $i, i+1$, and $i+2$ the rules g, z, and r, respectively, and whose rest $i+3, i+4, \ldots$ coincides with the rest $i+1, i+2, \ldots$ of $<'$ after removal of g and r. It can be shown that $<''$ is an optimal ordering for A [11]. Moreover, $<''$ fulfills the property for all $j \leq i$. This proves the result. ∎

Proposition 7.1 *Given a fully prioritized (finite) propositional program $\mathcal{P} = (P, <)$ and a set $A \subseteq Lits$, both deciding whether $A \in \mathcal{AS}(P)$ and $A \in \mathcal{PAS}(\mathcal{P})$ is possible in polynomial time. Moreover, these problems are complete for* P *(under logspace-reductions).*

Proof. It is well known that testing whether a set of literals A is an answer set of a program P, i.e., condition $\gamma_P(A) = A$, can be done in polynomial time (cf. [5, 22]); indeed, the Gelfond-Lifschitz reduct P^A is computable in polynomial time, and $Cn(P^A)$ is computable in polynomial time. Thus, deciding whether $A \in \mathcal{AS}(\mathcal{P})$ is polynomial.

For deciding $A \in \mathcal{PAS}(\mathcal{P})$, we note that the additional condition $\lambda_{\mathcal{P}}(A) = A$ is also polynomial-time checkable. Indeed, the dual reduct $\mathcal{P}' = {}^A\mathcal{P}$ can be clearly computed in polynomial time, and each S_α, $0 \leq \alpha < ord(<)$, can be computed in polynomial time. Since $ord(<)$ is the number of rules in $\mathcal{P}'$, it is thus clear that $C_{\mathcal{P}'}(A) = \lambda_{\mathcal{P}}(A)$ is polynomial-time computable, which proves membership in P.

For establishing completeness, we observe that recognizing the (unique) preferred answer set of a prioritized program $\mathcal{P} = (P, <)$ in which no negation (neither classical nor default one) occurs is hard for P; in fact, this preferred answer set trivially corresponds to the least model of P, and recognition of the least model of P is easily proved hard for P from the well-known result that inference of a positive atom p from a propositional logic program P is P-complete (see [17]). Indeed, construct a program P' from P by adding all rules $q \leftarrow p$ where q is a propositional atom from the underlying language. Then, the set PL of all propositional atoms is the least model of P', if and only if p is a consequence of P. Since P' is easily constructed in logarithmic workspace, recognizing the least model of P, and as a consequence, recognition of a preferred answer set of $\mathcal{P}$ is hard for P. This proves the result. ∎

Lemma 7.2 *Let $\mathcal{P}$ be a propositional prioritized program, and let $A \in \mathcal{AS}(\mathcal{P})$. Then, $A \in \mathcal{PAS}(\mathcal{P})$, if and only if some execution of* FULL-ORDER *outputs a fully prioritized program $\mathcal{P}'$. Moreover, $\mathcal{P}' \in \mathcal{FP}(\mathcal{P})$ and $A \in \mathcal{PAS}(\mathcal{P}')$ hold for every such $\mathcal{P}'$, and either all runs output some $\mathcal{P}'$, or all output "no".*

Proof. ($\Leftarrow$) Suppose that some execution of FULL-ORDER produces a prioritized program $\mathcal{P}' = (P, <')$. It is not hard to see that $\mathcal{P}' \in \mathcal{FP}(\mathcal{P})$. Moreover, from Proposition 5.1, we derive that $A \in \mathcal{PAS}(\mathcal{P}')$; indeed, by the construction of $<'$, every zombie rule $r \in P$ for A is defeated by the set T, which contains only heads of rules $r' \in P$ which are generating for A and such that $r' <' r$. Thus, by Proposition 5.1, $A \in \mathcal{PAS}(\mathcal{P}')$, which means $A \in \mathcal{PAS}(\mathcal{P})$.

($\Rightarrow$) Suppose $A \in \mathcal{PAS}(\mathcal{P})$. Then, by definition, there is a full prioritization $\mathcal{P}_1 = (P, <_1) \in \mathcal{FP}(\mathcal{P})$ such that $A \in \mathcal{PAS}(\mathcal{P}_1)$. We claim that there is a run of FULL-ORDER which produces $\mathcal{P}_1$. Indeed, select the rules r in Step 3 along the ordering $<_1$. An easy induction, using Proposition 5.1, shows that every rule r selected in Step 3 which is labeled "z" is defeated by the current set T. Therefore, this run of FULL-ORDER outputs $\mathcal{P}_1$. This proves the first part of the lemma.

Furthermore, it can be shown that there are never runs of FULL-ORDER which output some prioritized program $\mathcal{P}' = (P, <')$ and "no" on the same input, respectively [11]. ∎

Theorem 7.4 *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.*

Proof. For deciding whether $\mathcal{AS}(P) \neq \emptyset$, a proper fully prioritized version $\mathcal{P}' = (P, <') \in \mathcal{FP}(\mathcal{P})$ and an answer set $A \subseteq Lits$ for $\mathcal{P}'$ may be guessed; the size of the guess is polynomial, and checking whether the guess is proper can be done from Proposition 7.3 in polynomial time. Hence, the problem is in NP.

To show the hardness part, we construct for a collection of nonempty propositional clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ on atoms $p_1, \ldots, p_n$ a program P and a linear order $<$ such that the preferred answer sets of $(P, <)$ correspond to the satisfying truth assignments of $\mathcal{C}$. Since the construction is polynomial and propositional satisfiability is a well-known NP-complete problem, this proves NP-hardness.

The program P involves the propositional atoms $p_i, \hat{p}_i$, $i = 1, \ldots, n$, as well as a, b, and $unsat$. It has four groups of rules $R_1, \ldots, R_4$ as follows. The first group R_1 contains the rules

$$p_i \leftarrow p_i, \qquad \hat{p}_i \leftarrow \hat{p}_i, \qquad \text{for all } i = 1, \ldots, n;$$

(The use of these seemingly redundant clauses will become clear later.) The next group R_2 contains the rules

$$p_i \leftarrow \text{not}\, \hat{p}_i, \qquad \hat{p}_i \leftarrow \text{not}\, p_i \qquad \text{for all } i;$$

The group R_3 contains for each clause $C_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$ from $\mathcal{C}$ a rule

$$unsat \leftarrow \ell^+_{i,1}, \ldots, \ell^+_{i,m},$$

where

$$\ell^+_{i,j} = \begin{cases} p_k, & \text{if } L_{i,j} = \neg p_k, \text{ for some } k; \\ \hat{p}_k, & \text{if } L_{i,j} = p_k, \text{ for some } k. \end{cases}$$

The last group R_4 contains the rules

$$\begin{array}{ll} (1) & a \leftarrow \text{not}\, b \\ (2) & b \leftarrow unsat \end{array}$$

Notice that the program $P = \bigcup_{i=1}^{4} R_i$ does not involve strong negation.

It is easy to see that the answer sets of P correspond 1-1 to the truth assignments ϕ to the atoms $p_1, \ldots, p_n$. For each such ϕ, the corresponding answer set A_ϕ contains those atoms p_i such that $\phi(p_i) = \mathit{true}$ and all atoms $\hat{p}_i$ where $\phi(p_i) = \mathit{false}$; if ϕ satisfies $\mathcal{C}$, then A_ϕ contains a, and if ϕ does not satisfy $\mathcal{C}$, then A_ϕ contains $unsat$ and b.

Now let us define priorities. Choose any total order $<$ on P such that $R_i < R_j$ holds, for all $i < j$, and such that the rule (1) is ordered before rule (2).

For any answer set A_ϕ of P, let us trace the construction of $\lambda_{\mathcal{P}}(A_\phi) = \bigcup_i S_i$. The dual reducts ${}^{A_\phi}r$ of the (surviving) rules $r \in R_1$ are considered first; they fire and yield atom p_i (resp. $\hat{p}_i$), precisely if it occurs in A_ϕ, for each $i = 1, \ldots, n$. Next, the rules of R_2 are examined, which all survive the dual reduction; however, each of them is either a dead rule or a zombie, which is defeated by the union $\bigcup_j S_j$ of the preceding sets S_j. Therefore, none of them fires. In the next step, the reducts ${}^{A_\phi}r$ of the rules $r \in R_3$ are considered. Some of them fires and adds $unsat$, precisely if the truth assignment ϕ does not satisfy $\mathcal{C}$ (i.e., iff $unsat \in A_\phi$). Consequently, on the atoms $p_i, \hat{p}_i$, and $unsat$, $C_{\mathcal{P}}(A_\phi)$ will coincide with A_ϕ; it thus depends on the rules of R_4 whether all priorities are properly respected or not. Rule (1) fires and adds a to $C({}^{A_\phi}\mathcal{P})$, since it is not defeated by the preceding sets S_j. If $unsat \in A_\phi$, then the dual reduct of (2) fires and adds b to $C({}^{A_\phi}\mathcal{P})$; this means $\lambda_{\mathcal{P}}(A_\phi) \neq A_\phi$, however, as in this case $a \notin A_\phi$. On the other hand, if $unsat \notin A_\phi$, then (2) does not survive the dual reduction and $\lambda_{\mathcal{P}}(A_\phi) = A_\phi$.

Hence, $\mathcal{PAS}(\mathcal{P}) = \{A \in \mathcal{AS}(P) \mid a \in A, unsat \notin A\}$, and there is a 1-1 correspondence between the preferred answer sets of $\mathcal{P}$ and the satisfying truth assignments of $\mathcal{C}$. Clearly, the program $\mathcal{P}$ is constructible from $\mathcal{C}$ in logarithmic space (and thus in polynomial time); this proves the result. ∎

Theorem 7.7 *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.*

Proof. The membership part is similar as in Theorem 7.4, with the difference that for refuting $\mathcal{P} \models L$, we have to find a preferred answer set A which does not contain L. Hardness follows from a simple reduction of inference to the complement of the consistency problem; given a prioritized program P, let p be a fresh propositional atom. Then, $\mathcal{P} \models p$ iff $\mathcal{P}$ has no preferred answer set. ∎

Proposition 7.8 *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* $\mathrm{F}\Delta_2^p[O(\log n)]$.

Proof. Given a prioritized program $\mathcal{P} = (P, <)$, an answer set $A \in \mathcal{AS}(A)$ and an integer $k \geq 0$, deciding whether $pvd_{\mathcal{P}}(A) \leq k$ is in NP: we can guess a proper full prioritization $\mathcal{P}' = (P, <') \in \mathcal{FP}(\mathcal{P})$ and a total ordering $<''$ such that $A \in \mathcal{PAS}(P, <'')$ and $d(<', <'') \leq k$, and check this guess in polynomial time. Indeed, the guess has clearly polynomial size; computing $d(<, <')$ is obviously polynomial, and checking whether A is a preferred answer set of $\mathcal{P}'$ is polynomial by Proposition 7.1.

As a consequence, the value of $pvd_{\mathcal{P}}(A)$ is computable in a binary search on the range $[0, s]$, where $s = O(|P|^2)$, by making $O(\log |P|) = O(\log |I|)$ many calls to an NP oracle, where $|I|$ is the size of the input $\langle \mathcal{P}, A, k \rangle$. Hence, it follows that computing $pvd_{\mathcal{P}}(A)$ is in $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$.

The value of $pvd(\mathcal{P})$ can be computed similarly in a binary search. Indeed, given $\mathcal{P}$ and k, deciding whether $pvd(\mathcal{P}) \leq k$ is also in NP: Since $pvd(\mathcal{P}) \leq k$ holds iff some $A \in \mathcal{AS}(\mathcal{P})$ exists such that $pvd_{\mathcal{P}}(A) \leq k$, simply extend the previous guess-and-check algorithm for deciding whether $pvd_{\mathcal{P}}(A) \leq k$ by a guess for A; observe that testing $A \in \mathcal{AS}(\mathcal{P})$ is polynomial. Hence, it follows that computing $pvd(\mathcal{P})$ is in $\mathrm{F}\Delta_2^p[O(\log n)]$. ∎

Theorem 7.9 *Given a prioritized propositional program $\mathcal{P} = (P, <)$ and a set $A \subseteq Lits$ of ground literals, deciding whether $A \in \mathcal{WAS}(\mathcal{P})$ is* $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$*-complete. Hardness for* $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ *holds even if $<$ is a total ordering and P does not contain strong negation.*

Proof. (Sketch) From Proposition 7.8, it follows that algorithm REC-WAS can be implemented such that it runs in polynomial time and makes $O(\log n)$ many queries to an NP oracle. Thus, by Proposition 7.1, membership in $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ follows.

To show the hardness part, we reduce the problem PARITY(SAT) to our problem, which is as follows [69]. Given instances $I = \mathcal{C}_1, \ldots, \mathcal{C}_\ell$ of SAT (i.e., propositional clause sets $\mathcal{C}_i$), decide whether the number of Yes-Instances among them is odd.

We construct in [11] from I a prioritized program $\mathcal{P} = (P, <)$ and a set of literals A in polynomial time such that $A \in \mathcal{WAS}(\mathcal{P})$ if and only if I is a Yes-Instance, provided that I satisfies the following property: If $\mathcal{C}_i$ is a No-Instance of SAT, then also $\mathcal{C}_{i+1}$ is No-Instance, for every $i = 1, \ldots, \ell - 1$, where $\ell \geq 3$. Somewhat surprisingly, this strong assertion suffices to prove $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-hardness [69].

The proof is rather technical, and we confine here to give the basic idea of the construction; the details can be found in [11]. The program $\mathcal{P}$ has several components, ordered as follows: $\mathcal{P}_{\mathcal{C}} < \mathcal{P}_{par} < \mathcal{P}_{ch} < \mathcal{P}_{var} < \mathcal{P}_{o/e}$. They serve for the following purposes.

$\mathcal{P}_{\mathcal{C}}$ encodes the clauses of the SAT instances $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$. For each $\mathcal{C}_i$, there is a collection of rules in $\mathcal{P}_{\mathcal{C}}$. Some of them will be zombie rules, and for defeating them some generating rules from the lower priority part $\mathcal{P}_{var}$ have to be moved in front of them. In particular, some distinguished rule $f_i \leftarrow$ must be moved in front of them in any optimal solution, if $\mathcal{C}_i$ is unsatisfiable.

$\mathcal{P}_{par}$ encodes the parity check on the number of satisfiable $\mathcal{C}_i$. In particular, it effects that any weakly preferred answer set contains a designated atom *odd*, if this number is odd, and an atom $even$, if it is even. Zombie rules in $\mathcal{P}_{par}$ may be defeated by rules from $\mathcal{P}_{o/e}$.

$\mathcal{P}_{ch}$ is a "channel" which separates the components $\mathcal{P}_{\mathcal{C}}$ and $\mathcal{P}_{par}$ from the remaining components $\mathcal{P}_{var}$ and $\mathcal{P}_{o/e}$, such that moving rules across this channel is very costly, and as few rules as necessary are moved across it in any weakly preferred answer set. This channel is unidirectional, i.e., in any weakly preferred answer set rules are moved from $\mathcal{P}_{var}$ and $P_{o/e}$ into $\mathcal{P}_{\mathcal{C}}$ and $\mathcal{P}_{par}$, but not vice versa.

$\mathcal{P}_{var}$ and $\mathcal{P}_{o/e}$ contain rules which are potentially moved across the channel $\mathcal{P}_{ch}$, in order to defeat zombie rules in $\mathcal{P}_{\mathcal{C}}$ and $\mathcal{P}_{par}$. In particular, $\mathcal{P}_{o/e}$ contains rules defining atoms *odd* and $even$, respectively.

The program $\mathcal{P}$ is designed to have two answer sets: A_{odd}, which contains *odd*, and A_{even}, which contains $even$, such that $A_{odd} \in \mathcal{WAS}(\mathcal{P})$ iff $\mathcal{C}$ is a Yes-instance, and $A_{even} \in \mathcal{WAS}(\mathcal{P})$ iff $\mathcal{C}$ is a No-instance. ∎

Theorem 7.10 *Given a prioritized propositional program $\mathcal{P} = (P, <)$, computing an arbitrary $A \in \mathcal{WAS}(\mathcal{P})$ is complete for* FNP//OptP$[O(\log n)]$. *Hardness holds even if $<$ is a total ordering and P does not contain strong negation.*

Proof. The proof of membership in FNP//OptP$[O(\log n)]$ is in the discussion preceding the theorem.

Hardness for FNP//OptP$[O(\log n)]$ can be shown by a reduction from the problem X-MAXIMAL MODEL [15]: Given a set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of propositional clauses on atoms $Y = \{y_1, \ldots, y_n\}$ and a set $X \subseteq Y$, compute a model M (satisfying assignment) of $\mathcal{C}$ whose X-part is maximal, i.e., for every other model M' of $\mathcal{C}$ such that $M \neq M'$, there exists some atom $y_i \in X$ such that $M \models y_i$ and $M' \models \neg y_i$.

This problem is proved complete for FNP//OptP$[O(\log n)]$ in [15, 16], under the following concept of "metric reduction" [15]:[5] Problem Π reduces to problem Π', if there are polynomial-time computable functions $f(x)$ and $g(x, y)$, such that: (i) for any instance I of Π, $f(I)$ is an instance of Π', and $f(I)$ has a solution iff I has a solution; and (ii) for any arbitrary solution S of $f(I)$, $g(I, S)$ is a solution of I.

We describe a reduction of X-MAXIMAL MODEL to computing an arbitrary weakly preferred answer set. The reduction must comprise two polynomial-time computable functions $f(\mathcal{C})$ and $g(\mathcal{C}, A)$, such that: (i) $f(\mathcal{C})$ is a prioritized program $\mathcal{P}$, such that $\mathcal{P}$ has some weakly preferred answer set iff $\mathcal{C}$ is satisfiable, and (ii) for every weakly preferred answer set A of $\mathcal{P}$, $g(\mathcal{C}, A)$ is a X-maximal model of $\mathcal{C}$.

The reduction is similar to the one in the proof of Theorem 7.4. We use fresh atoms $unsat$ and $\hat{y}_i$, for every $y_i \in Y$, and define three sets of rules S_1, S_2, and S_3 as follows. The set S_1 contains all rules

$$y_i \leftarrow y_i, \qquad \hat{y}_i \leftarrow \hat{y}_i, \quad \text{for all } y_i \in Y \setminus X;$$

S_2 contains all rules

$$y_i \leftarrow \text{not}\, \hat{y}_i, \qquad \hat{y}_i \leftarrow \text{not}\, y_i \quad \text{for all } y_i \in Y;$$

and S_3 contains for each clause $C_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$ from $\mathcal{C}$ a rule

$$unsat \leftarrow \ell^+_{i,1}, \ldots, \ell^+_{i,k}, \text{not}\, unsat$$

where similar as above $\ell^+_{i,j} = y_h$ if $\ell_{i,j} = \neg y_h$, and $\ell^+_{i,j} = \hat{y_h}$ if $\ell_{i,j} = y_h$, for any atom $y_h \in Y$. Let $\mathcal{P} = (P, <)$, where $P = S_1 \cup S_2 \cup S_3$ and $<$ is any total order on P such that $S_1 < S_2 < S_3$ and the rules in S_2 are ordered as follows:

$$y_1 \leftarrow \text{not}\, \hat{y}_1 \;<\; \hat{y}_1 \leftarrow \text{not}\, y_1 \;<\; y_2 \leftarrow \text{not}\, \hat{y}_2 \;<\; \hat{y}_2 \leftarrow \text{not}\, y_2 \;<\ldots<\; y_n \leftarrow \text{not}\, \hat{y}_n \;<\; \hat{y}_n \leftarrow \text{not}\, y_n.$$

It is easily seen that the answer sets A of $\mathcal{P}$ correspond 1-1 to the models M of $\mathcal{C}$. Due to the ordering of the rules in S_2, by switching some rules $y_i \leftarrow \text{not}\, \hat{y}_i$ and $\hat{y}_i \leftarrow \text{not}\, y_i$ where $y_i \in X$, every answer set of $\mathcal{P}$ can be made preferred; by the rules in S_1, no switch of the respective rules for $y_i \in Y \setminus X$ is necessary. The number $pvd_{\mathcal{P}}(A)$ amounts to the number of atoms $y_i \in X$ which are false in the corresponding model M of $\mathcal{C}$. Thus, the weakly preferred answer sets of $\mathcal{P}$ correspond 1-1 to the models M of $\mathcal{C}$ in which a maximum number of atoms from X are true. Clearly, each of these models is a X-maximal model of $\mathcal{C}$.

Clearly, $\mathcal{P}$ is constructible from $\mathcal{C}$ in polynomial time, and from any $A \in \mathcal{WAS}(\mathcal{P})$, the corresponding X-maximal model M of $\mathcal{C}$ is easily constructed in time polynomial in the size of A. Therefore, polynomial-time functions $f(\mathcal{C})$ and $g(\mathcal{C}, A)$ as required exist, and problem X-MAXIMAL MODEL reduces to computing an arbitrary weakly preferred answer set. This proves hardness for FNP//OptP$[O(\log n)]$. ∎

[5] in [16] a a slightly different form of reduction is used as in [15]. It requires that $f(I)$ must always have solutions, but for any maximal solution S of $f(I)$, the function $g(I, S)$ is only defined if I has solutions; our proof can be easily adapted to this setting.

Theorem 7.11 *Deciding, given a finite prioritized propositional program $\mathcal{P} = (P, <)$ and a literal L, whether $\mathcal{P} \models_w L$ is $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-complete. Hardness for $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$ holds even if $<$ is a total order and P does not have strong negation.*

Proof. The problem $\mathcal{P} \models_w L$ can be solved in polynomial time with $O(\log |P|)$ many NP oracle calls: first compute $pvd(\mathcal{P})$ and then query the NP oracle whether some answer set A with $pvd_{\mathcal{P}}(A) \leq pvd(\mathcal{P})$ exists such that $L \notin A$. The oracle is in NP, since a proper $\mathcal{P}' = (P, <') \in \mathcal{FP}(\mathcal{P})$, a total ordering $<''$, and set of literals A such that $A \in \mathcal{PAS}(P, <'')$, $L \notin A$, and $d(<', <'') \leq pvd(\mathcal{P})$ can be guessed and checked in polynomial time (cf. proof of Proposition 7.8). Hence, the problem is in $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$.

The hardness part follows from the reduction in the proof of Theorem 7.10 and the following lemma: Given a set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of propositional clauses on atoms Y, a set $X \subseteq Y$, and an atom A, deciding whether A is true in all X-maximum models of $\mathcal{C}$ is $\Delta_2^p[O(\log n)]$-complete, where a model M is X-maximum, if it has largest possible X-part, i.e., $|M \cap X| \geq |M' \cap X|$ for all other models M' of $\mathcal{C}$. The weakly preferred answer sets of the prioritized program $\mathcal{P}$ constructed in the proof of Theorem 7.10 correspond 1-1 to the X-maximum models of $\mathcal{C}$ in the obvious way; this implies $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$-hardness of weak inference under the asserted restriction. ∎

References

[1] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] A. Analyti and S. Pramanik. Reliable semantics for extended logic programs with rule prioritization. *Journal of Logic and Computation*, pp. 303–324, 1995.

[3] 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.

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

[5] R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[6] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency Management and Prioritized Syntax-Based Entailment. *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 640–645. Morgan Kaufman, 1993.

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

[8] G. Brewka. Adding Priorities and Specificity to Default Logic. In *Proc. JELIA '94*, LNAI 838, pp. 247–260. Springer, 1994.

[9] G. Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

[10] G. Brewka and T. Eiter. Prioritizing Default Logic: Abridged Report. In *Festschrift on the occasion of Prof.Dr. W. Bibel's 60th birthday*. Kluwer, 1999. To appear.

[11] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. Technical Report INFSYS RR-1843-99-06, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, February 1999.

[12] 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.

[13] F. Buccafurri, N. Leone, and P. Rullo. Strong and Weak Constraints in Disjunctive Datalog. In *Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97)*, LNCS 1265, pp. 2–17. Springer, 1997.

[14] F. Buccafurri, N. Leone, and P. Rullo. Semantics and Expressiveness of Disjunctive Ordered Logic. *Annals of Mathematics and Artificial Intelligence*, 1999. To appear. Abstract in Proc. KR-98.

[15] Z.-Z. Chen and S. Toda. The Complexity of Selecting Maximal Solutions. In *Proc. 8th IEEE Structure in Complexity Theory Conference*, pp. 313–325, 1993.

[16] Z.-Z. Chen and S. Toda. The Complexity of Selecting Maximal Solutions. *Information and Computation*, 119:231–239, 1995.

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

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

[19] J. Dix. A Classification Theory of Semantics of Normal Logic Programs: Strong Properties. *Fundamenta Informaticae*, 22:227–255, 1995.

[20] J. Dix. A Classification Theory of Semantics of Normal Logic Programs: Weak Properties. *Fundamenta Informaticae*, 22:257–288, 1995.

[21] P. Dung. An Argumentation Semantics for Logic Programming with Explicit Negation. In *Proc. ICLP-93*, pp. 616–630. MIT Press, 1993.

[22] 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.

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

[24] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Trans. on Database Systems*, 22(3):364–417, 1997.

[25] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-Monotonic Reasoning. In *Proc. LPNMR-97*, LNCS 1265, pp. 364–375. Springer, 1997.

[26] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons, and Benchmarks. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proc. Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pp. 406–417, June 2–4 1998.

[27] D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for OrderedLogic Programs. In *Proc. KR-92*, pp. 208–217, 1992.

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

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

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

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

[32] M. Gelfond and T. Son. Reasoning with Prioritized Defaults. In *Selected Papers presented at the Workshop on Logic Programming and Knowledge Representation (LPKR '97), Port Jefferson*, LNAI 1471, pp. 164–223. Springer, 1998.

[33] G. Gottlob, N. Leone, and H. Veith. Succinctness as a Source of Complexity in Logical Formalisms. *Annals of Pure and Applied Logic*, 1999. To appear. Preliminary abstract 'Second-Order Logic and the Weak Exponential Hierarchies' in Proc. MFCS '95, LNCS 969, pp. 66–81, Springer 1995.

[34] J. Horty. Some Direct Theories of Nonmonotonic Inheritance. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. III, pp. 111–187. Clarendon Press, Oxford, 1994.

[35] D. S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. A, chapter 2. Elsevier, 1990.

[36] H. Katsuno and A. O. Mendelzon. Propositional Knowledge Base Revision and Minimal Change. *Artificial Intelligence*, 52:253–294, 1991.

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

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

[39] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic Reasoning, Preferential Models, and Cumulative Logics. *Artificial Intelligence*, 44:167–207, 1990.

[40] E. Laenens and D. Vermeir. A Fixpoint Semantics for Ordered Logic. *Journal of Logic and Computation*, 1:159–185, 1990.

[41] N. Leone, P. Rullo, A. Mecchia, and G. Rossi. A Deductive Environment for Dealing with Objects and Non-Monotonic Reasoning. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):539–558, 1997.

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

[43] J. Lobo and C. Uzcátegui. Abductive Consequence Relations. *Artificial Intelligence*, 89(1/2):149–171, 1997.

[44] W. Marek, A. Nerode, and J. Remmel. A Theory of Nonmonotonic Rule Systems II. *Annals of Mathematics and Artificial Intelligence*, 5:229–264, 1992.

[45] W. Marek, A. Nerode, and J. Remmel. The Stable Models of a Predicate Logic Program. *Journal of Logic Programming*, 21(3):129–153, 1994.

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

[47] W. Marek and M. Truszczyński. Computing Intersection of Autoepistemic Expansions. In A. Nerode, W. Marek, and V. Subrahmanian, editors, *Proc. LPNMR-91*, pp. 37–50. MIT Press, 1991.

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

[49] B. Nebel. How Hard is it to Revise a Belief Base ? In D. Dubois and H. Prade, editors, *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, vol. III, pp. 77–145. Kluwer Academic, 1998.

[50] D. Nute. A Non-Monotonic Logic Based on Conditional Logic. Technical Report TR 01–0007, Artificial Intelligence Programs, The University of Georgia, 1985.

[51] D. Nute. General and Special Defeasible Logic. In *Proc. Tübingen Workshop on Semantic Nets and Nonmonotonic Reasoning*, 1989.

[52] D. Nute. Defeasible Logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. III, pp. 353–395. Clarendon Press, Oxford, 1994.

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

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

[55] D. Poole. A Logical Framework for Default Reasoning. *Artificial Intelligence*, 36:27–47, 1988.

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

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

[58] H. Prakken and G. Sartor. Argument-based Logic Programming with Defeasible Priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.

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

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

[61] J. Rintanen. On Specificity in Default Logic. In *Proc. IJCAI '95*, pp. 1474–1479. AAAI Press, 1995.

[62] J. Rintanen. Lexicographic Priorities in Default Logic. *Artificial Intelligence*, 106:221–265, 1998.

[63] C. Sakama and K. Inoue. Representing Priorities in Logic Programs. In *Proc. IJCSLP-96*, pp. 82–96, Bonn, Germany, 1996. MIT-Press.

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

[65] J. Schlipf. The Expressive Powers of Logic Programming Semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995. Abstract in Proc. PODS 90, pp. 196–204.

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

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

[68] D. Touretzky, R. Thomason, and J. Horty. A Skeptic's Menagerie: Conflictors, Preemptors, Reinstaters, and Zombies in Nonmonotonic Inheritance. In *Proc. IJCAI '91*, pp. 478–485, 1991.

[69] K. Wagner. More Complicated Questions about Maxima and Minima, and Some Closures of NP. In *Proc. ICALP-86*, pp. 434–443, 1986. also *Theoretical Computer Science*, 51:53–80, 1987.

[70] K. Wagner. Bounded Query Classes. *SIAM Journal of Computing*, 19(5):833–846, 1990.

[71] Y. Zhang and N. Foo. Answer Sets for Prioritized Logic Programs. In *Proc. ILPS 97*, pp. 69–83, 1997.

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