Computing Preferred and Weakly Preferred Answer Sets by Meta-Interpretation in Answer Set Programming*

Thomas Eiter and **Wolfgang Faber**
Institut für Informationssysteme 184/3
TU Wien
Favoritenstr. 9-11, 1040 Wien, Austria
{eiter,faber}@kr.tuwien.ac.at

Nicola Leone
Department of Mathematics
University of Calabria
87030 Rende (CS), Italy
leone@unical.it

Gerald Pfeifer
Institut für Informationssysteme 184/2
TU Wien
Favoritenstr. 9-11, 1040 Wien, Austria
pfeifer@dbai.tuwien.ac.at

Abstract

Preferred and Weakly Preferred Answer Sets are extensions to Answer Set Programming (ASP) which allow the user to specify priorities for rules. In this paper we present a first implementation of these formalisms by means of "meta-interpreters" on top of `DLV`, an efficient engine for Disjunctive ASP. This approach shows the suitability of ASP in general and of `DLV` in particular for fast prototyping and experimenting with new languages and knowledge-representation formalisms. In addition to two "straightforward" meta-interpreters, we also present a graph-based meta-interpreter that often allows for more efficient computations.

Introduction

Handling preference information plays an important role in applications of nonmonotonic reasoning. A number of different approaches for adding preferences to logic programs and related formalisms have been proposed, including (Baader & Hollunder 1995; Brewka 1994; 1996; Buccafurri, Leone, & Rullo 2000; Delgrande & Schaub 1997; Gelfond & Son 1997; Marek & Truszczyński 1993; Rintanen 1998; Sakama & Inoue 1996; Zhang & Foo 1997). They have been designed for purposes such as capturing specificity or normative preference; see (Brewka & Eiter 1999) for a review and discussion.

The following example shows a classical situation where preference information should be used.

Example 1 (bird & penguin) *Consider the following logic program:*

(1) $peng.$
(2) $bird.$
(3) $\neg flies$:- `not` $flies, peng.$
(4) $flies$:- `not` $\neg flies, bird.$

This program has two answers sets: $A_1 = \{peng, bird, \neg flies\}$ *and* $A_2 = \{peng, bird, flies\}$. *Assume that rule* (i) *has higher priority than* (j) *iff* $i < j$ *(i.e., rule (1) has the highest priority and rule (4) the lowest). Then,* A_2 *is no longer intuitive: indeed,* $flies$ *is concluded from (4); on the other hand, (3) has higher priority than (4), and thus* $\neg flies$ *should be concluded.*

In (Brewka & Eiter 1999), the authors presented an approach to preferred answer sets which refines previous approaches for adding preferences to default rules in (Brewka 1994; 1996). It is defined for answer sets of extended logic programs (Gelfond & Lifschitz 1991) and is generalized to Reiter's default logic in (Brewka & Eiter 2000). An important aspect of this approach is that the definition of preferred answer sets was guided by two general principles which, as argued, a preference semantics should satisfy. As shown in (Brewka & Eiter 1999), preferred answer sets satisfy the principles, while almost all other semantics do not. Since, in general, programs having an answer set may lack a preferred answer set, also a relaxed notion of weakly preferred answer sets was defined in (Brewka & Eiter 1999).

In this paper, we address the implementation of preferred and weakly preferred answer sets, announced as future work in (Brewka & Eiter 1999), on top of the `DLV` system. To this end, we use a powerful technique, based on Answer Set Programming (ASP), which can be seen as a sort of *meta-programming in ASP*. A similar technique has been applied previously in (Gelfond & Son 1997) for defining the semantics of logic programming with defeasible rules (cf. Related Work).

For implementing the preferred answer set semantics, we provide a disjunctive logic program (DLP) P_{I_p} (the "meta-program") such that, given an arbitrary prioritized ground logic program $\mathcal{P}$ encoded by a suitable set of facts $F(\mathcal{P})$, the answer sets of $P_{I_p} \cup F(\mathcal{P})$ correspond (modulo a simple projection function) precisely to the preferred answer sets of $\mathcal{P}$. This way, by running $P_{I_p} \cup F(\mathcal{P})$ on the `DLV` system (Faber, Leone, & Pfeifer 1999; Faber & Pfeifer since 1996), we compute the preferred answer sets of $\mathcal{P}$ in a simple and elegant way.

For implementing weakly preferred answer sets, we adopt a similar approach, where the meta-interpreter additionally uses the *weak constraints* feature (Buccafurri, Leone, & Rullo 1997) of `DLV`.

The work reported here is important in several ways:

- We put forward the use of ASP for experimenting new

*This work was supported by FWF (Austrian Science Funds) under the projects P11580-MAT, P13871-INF, and Z29-INF.

semantics by means of a meta-interpretation technique. The declarativity of DLPs provides a new, elegant way of writing meta-interpreters, which is very different from Prolog-style meta-interpretation. Thanks to the high expressiveness of DLP and DLV's weak constraints, meta-interpreters can be written in a simple and declarative fashion.

• The description of the "meta-programs" for implementing preferred and weakly preferred answer sets has also a didactic value: it is a good example on the way how meta-interpreters can be built using ASP.

• Furthermore, the meta-interpreters provided are relevant per se, since they provide an actual implementation of preferred and weakly preferred answer sets and allow for easy experimenting with these semantics in practice. To our knowledge, this is the first implementation of weakly preferred answer sets. An implementation of preferred answer sets (also on top of DLV) is reported in (Delgrande, Schaub, & Tompits 2000), by mapping programs into the framework of compiled preferences (Delgrande & Schaub 1997). Our implementation, as will be seen, is an immediate translation of the definition of preferred answer sets into DLV code, and similar for weakly preferred answer sets. Weak constraints make the encoding of weakly preferred answer sets extremely simple and elegant, while that task would have been much more cumbersome else.

In summary, the experience reported in this paper confirms the power of ASP. It suggests the use of the DLV system as a high-level abstract machine to be employed also as a powerful tool for experimenting with new semantics and novel KR languages.

It is worthwhile noting that the meta-interpretation approach presented here does not aim at efficiency; rather, this approach fosters *simple and very fast prototyping*, which is useful e.g. in the process of designing and experimenting new languages.

Disjunctive LPs and Preferred Answer Sets

Disjunctive LPs

Syntax. DLPs use a function-free first-order language. As for terms, strings starting with uppercase (resp., lowercase) letters denote variables (resp., constants). A (*positive* resp. *negative*) *classical literal* l is either an atom a or a negated atom $\neg a$, respectively; its *complementary* literal, denoted $\neg l$, is $\neg a$ and a, respectively. A (*positive* resp., *negative*) *negation as failure (NAF) literal* ℓ is of the form l or $\texttt{not}\ l$, where l is a classical literal. Unless stated otherwise, by *literal* we mean a classical literal.

A *disjunctive rule* (*rule*, for short) r is a formula

$$a_1 \ \texttt{v} \ \cdots \ \texttt{v} \ a_n \ \texttt{:-} \ b_1, \cdots, b_k, \ \texttt{not} \ b_{k+1}, \cdots, \ \texttt{not} \ b_m. \quad (1)$$

where all a_i and b_j are classical literals and $n \geq 0$, $m \geq k \geq 0$. The part to the left of "$\texttt{:-}$" is the *head*, the part to the right is the *body* of r, where "$\texttt{:-}$" is omitted if $m = 0$. We let $H(r) = \{a_1, \ldots, a_n\}$ be the set of head literals and $B(r) = B^+(r) \cup B^-(r)$ the set of body literals, where $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ are the sets of positive and negative body literals, respectively. An *integrity constraint* is rule with empty head ($n = 0$).

A *weak constraint* r is an expression of the form

$$:\sim b_1, \cdots, b_k, \texttt{not} \ b_{k+1}, \cdots, \ \texttt{not} \ b_m. \, [w : l]$$

where every b_i is a literal and $l \geq 1$ is the *priority level* and $w \geq 1$ the *weight* among the level. Both l and w are integers and set to 1 if omitted. The sets $B(r)$, $B^+(r)$, and $B^-(r)$ are defined by viewing r as an integrity constraint.

A *disjunctive datalog program (DLP) with weak constraints* P (simply, *program*) is a finite set of rules and weak constraints. $WC(P)$ denotes the set of weak constraints in P. We call P *wc-free*, if $WC(P) = \emptyset$; *positive*, if P is $\texttt{not}$-free (i.e. $\forall r \in \mathcal{P} : B^-(r) = \emptyset$); and *normal*, if P is $\texttt{v}$-free (i.e. $\forall r \in \mathcal{P} : |H(r)| \leq 1$).

As usual, a term (atom, rule,...) is *ground*, if no variables appear in it. A ground program is also called *propositional*.

Semantics Answer sets for DLPs with weak constraints are defined by extending consistent answer sets for DLPs as introduced in (Gelfond & Lifschitz 1991; Lifschitz 1996). We proceed in three steps: we first define answer sets (1) of ground positive wc-free programs, then (2) of arbitrary wc-free ground programs, and (3) finally (optimal) answer sets of ground programs with weak constraints. As usual, the (optimal) answer sets of a non-ground program P are those of its ground instantiation $Ground(P)$, defined below.

For any program P, let U_P be its Herbrand universe and B_P be the set of all classical ground literals from predicate symbols in P over the constants of U_P; if no constant appears in P, an arbitrary constant is added to U_P. For any clause r, let $Ground(r)$ denote the set of its ground instances. Then, $Ground(P) = \bigcup_{r \in \mathcal{P}} Ground(r)$. Note that P is ground iff $P = Ground(P)$. An interpretation is any set $I \subseteq B_P$ of ground literals. It is consistent, if $I \cap \{\neg l \mid l \in I\} = \emptyset$.

In what follows, let P be a ground program.

(1) A consistent[1] interpretation $I \subseteq B_P$ is called *closed* under a positive wc-free program P, if $B(r) \subseteq I$ implies $H(r) \cap I \neq \emptyset$ for every $r \in P$. A set $X \subseteq B_P$ is an *answer set* for P if it is a minimal set (w.r.t. set inclusion) closed under P.

(2) Let P^I be the *Gelfond-Lifschitz reduct* of a wc-free program P w.r.t. $I \subseteq B_P$, i.e., the program obtained from P by deleting

- all rules $r \in P$ such that $B^-(r) \cap I \neq \emptyset$, and
- all negative body literals from the remaining rules.

Then, $I \subseteq B_P$ is an answer set of P iff I is an answer set of P^I. By $\mathcal{AS}(P)$ we denote the set of all answer sets of P.

(3) Given a program P with weak constraints, we are interested in the answer sets of the wc-free part which minimise the sum of weights of the violated constraints in the highest priority level, and among them those which minimise the sum of weights of the violated constraints in the

[1] We only consider *consistent answer sets*, while in (Lifschitz 1996) also the (inconsistent) set B_P may be an answer set.

next lower level, etc. This is expressed by an objective function for P and an answer set A:

$$
\begin{aligned}
&f_{\mathcal{P}}(1) = 1 \\
&f_{\mathcal{P}}(n) = f_{\mathcal{P}}(n-1) \cdot |WC(P)| \cdot w^{\mathcal{P}}_{max} + 1, \quad n > 1 \\
&H^{\mathcal{P}}_{A} \quad = \textstyle\sum_{i=1}^{l^{\mathcal{P}}_{max}} (f_{\mathcal{P}}(i) \cdot \sum_{N \in N_i^{A,\mathcal{P}}} w_N)
\end{aligned}
$$

where $w^{\mathcal{P}}_{max}$ and $l^{\mathcal{P}}_{max}$ denote the maximum weight and maximum level of a weak constraint in P, respectively; $N_i^{A,\mathcal{P}}$ denotes the set of weak constraints in level i which are violated by A; and, w_N denotes the weight of the weak constraint N. Note that $|WC(P)| \cdot w^{\mathcal{P}}_{max} + 1$ is greater than the sum of all weights in the program, and therefore guaranteed to be greater than any sum of weights of a single level. If weights in level i are multiplied by $f_{\mathcal{P}}(i)$, it is sufficient to calculate the sum of these updated weights, such that the updated weight of a violated constraint of a greater level is always greater than any sum of updated weights of violated constraints of lower levels.

Then, A is an *(optimal) answer set* of P, if $A \in \mathcal{AS}(\mathcal{P} \setminus WC(P))$ and $H^{\mathcal{P}}_{A}$ is minimal over $\mathcal{AS}(\mathcal{P} \setminus WC(P))$. Let $\mathcal{OAS}(\mathcal{P})$ denote the set of all optimal answer sets.

Preferred Answer Sets

We recall and adapt the definitions of (Brewka & Eiter 1999) as needed in the current paper. Throughout the rest of this section, programs are tacitly assumed to be propositional.

Definition 1 *A prioritized (propositional) program is a pair $\mathcal{P} = (P, <)$ where P is a normal logic program without constraints, and $<$ is a strict partial order on P, i.e., an irreflexive ($a \not< a$, for all a) and transitive relation.*

Informally, $r_1 < r_2$ means "r_1 has higher priority than r_2". For any $\mathcal{P} = (P, <)$, the answer sets of $\mathcal{P}$ are those of P; their collection is denoted by $\mathcal{AS}(\mathcal{P})$ $(= \mathcal{AS}(P))$.

Definition 2 *A full prioritization of a prioritized program $\mathcal{P} = (P, <)$ is any pair $\mathcal{P}' = (P, <')$ where $<'$ is a total order of P refining $<$, i.e., $r_1 < r_2$ implies $r_1 <' r_2$, for all $r_1, r_2 \in P$. The set of all full prioritizations of $\mathcal{P}$ is denoted by $\mathcal{FP}(\mathcal{P})$. We call $\mathcal{P}$ fully prioritized, if $\mathcal{FP}(\mathcal{P}) = \{\mathcal{P}\}$.*

Fully prioritized programs $\mathcal{P} = (P, <)$ are also referred to as ordered sets $\mathcal{P} = \{r_1, \ldots, r_n\}$ of rules where $r_i < r_j$ iff $i < j$.

We define preferred answer sets first for a particular class of fully prioritized programs. Call a program $\mathcal{P}$ *prerequisite-free*, if $B^+(r) = \emptyset$ for every rule $r \in \mathcal{P}$ holds. Furthermore, a literal ℓ (resp., a set $X \subseteq B_P$ of literals) *defeats* a rule r of the form (1), if $\ell \in B^-(r)$ (resp., $X \cap B^-(r) \neq \emptyset$).

Definition 3 *Let $\mathcal{P} = \{r_1, \ldots, r_n\}$ be a fully prioritized and prerequisite-free program. For any set $S \subseteq B_P$ of literals, the sequence $S_i \subseteq B_P$, $0 \leq i \leq n$, is defined as follows: $S_0 = \emptyset$ and*

$$
S_i = \begin{cases} S_{i-1}, & \text{if } (\alpha)\ S_{i-1} \text{ defeats } r_i, \text{ or } (\beta) \\ & H(r_i) \subseteq S \text{ and } S \text{ defeats } r_i, \\ S_{i-1} \cup H(r_i), & \text{otherwise,} \end{cases}
$$

for all $i = 1, \ldots, n$. The set $C_{\mathcal{P}}(S)$ is defined by

$$
C_{\mathcal{P}}(S) = \begin{cases} S_n, & \text{if } S_n \text{ is consistent,} \\ B_P & \text{otherwise.} \end{cases}
$$

An answer set A $(= S)$ divides the rules of P in Def. 3 into three groups: *generating rules*, which are applied and contribute in constructing 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; and *zombies*, which are the rules not applicable in A whose consequences do not belong to A. Only zombies have the potential to render an answer set non-preferred. This is the case if some zombie is not "killed" by a generating rule of higher priority. If A is a fixpoint of $C_{\mathcal{P}}$, then the inductive construction guarantees that indeed all zombies are defeated by generating rules with higher preference.

Definition 4 *Let $\mathcal{P}$ be a fully prioritized and prerequisite-free program, and let $A \in \mathcal{AS}(\mathcal{P})$. Then A is a preferred answer set of $\mathcal{P}$ if and only if $C_{\mathcal{P}}(A) = A$.*

In the case where P is not prerequisite-free, a kind of dual Gelfond-Lifschitz reduct is computed as follows.

Definition 5 *Let $\mathcal{P} = (P, <)$ be a fully prioritized program, and let $X \subseteq B_P$. Then ${}^X\mathcal{P} = ({}^XP, {}^X<)$ is the fully prioritized program such that:*

- *${}^XP$ is the set of rules obtained from P by deleting*
  1. *every $r \in \mathcal{P}$ such that $B^+(r) \not\subseteq X$, and*
  2. *all positive body literals from the remaining rules.*
- *${}^X<$ is inherited from $<$ by the map $f : {}^XP \longrightarrow P$ (i.e., $r'_1\ {}^X< r'_2$ iff $f(r'_1) < f(r'_2)$), where $f(r')$ is the first rule in P w.r.t. $<$ such that r' results from r by step 2.*

The definition of ${}^X<$ must respect possible clashes of rule priorities, as step 2 may produce duplicate rules in general.

Definition 6 *Let $\mathcal{P} = (P, <)$ be a prioritized program and $A \subseteq B_P$. If $\mathcal{P}$ is fully prioritized, then A is a preferred answer set of $\mathcal{P}$ iff A is a preferred answer set of ${}^A\mathcal{P}$; otherwise, A is a preferred answer set of $\mathcal{P}$ iff A is a preferred answer set for some $\mathcal{P}' \in \mathcal{FP}(\mathcal{P})$. By $\mathcal{PAS}(\mathcal{P})$ we denote the set of all preferred answer sets of $\mathcal{P}$.*

Example 2 *Reconsider the bird & penguin example. Let us first check whether $A_1 = \{peng, bird, \neg flies\}$ is a preferred answer set. We determine the dual reduct ${}^{A_1}\mathcal{P}$ which consists of the following rules:*

(1) $peng.$
(2) $bird.$
(3) $\neg flies$:- not $flies.$
(4) $flies$:- not $\neg flies.$

The order ${}^{A_1}<$ coincides with $<$ as in Def. 5. Now, let us determine $A_{1,4}$ $(= S_4)$, by constructing the sequence $A_{1,i}$, for $0 \leq i \leq 4$: $A_{1,0} = \emptyset$, $A_{1,1} = \{peng\}$, $A_{1,2} = \{peng, bird\}$, $A_{1,3} = \{peng, bird, \neg flies\}$, and $A_{1,4} = A_{1,3}$. Thus, $A_{1,4} = \{peng, bird, \neg flies\} = A_1$ and $C_{{}^{A_1}\mathcal{P}}(A_1) = A_1$; hence, the answer set A_1 is preferred.

Next consider the answer set $A_2 = \{peng, bird, flies\}$. The dual reducts ${}^{A_2}\mathcal{P}$ and ${}^{A_1}\mathcal{P}$ coincide, and thus $A_{2,4} = A_1$, which means $C_{{}^{A_2}\mathcal{P}}(A_2) \neq A_2$. Hence, A_2 is not preferred, and A_1 is the single preferred answer set of $\mathcal{P}$.

The following example shows that not every prioritized program which has an answer set has also a preferred one.

Example 3 *Consider the following program:*

(1) c :- not b.
(2) b :- not a.

where $(1) < (2)$. *Its single answer set is* $A = \{b\}$. *However,* $C_{\mathcal{A}_{\mathcal{P}}}(A) = \{c, b\}$ *and thus* A *is not preferred.*

Weakly Preferred Answer Sets

The concept of weakly preferred answer set relaxes the priority ordering as little as necessary to obtain a preferred answer set, if no answer set is preferred. It can be seen as a conservative approximation of a preferred answer set.

Definition 7 *Let* $<_1$ *and* $<_2$ *be total orderings of the same finite set* M. *The distance from* $<_1$ *to* $<_2$, *denoted* $d(<_1, <_2)$, *is the number of pairs* $m, m' \in M$ *such that* $m <_1 m'$ *and* $m' <_2 m$.[2]

Clearly, $d(<_2, <_1)$ defines a metric on the set of all total orderings of M. For example, the distance between $a <_1 b <_1 c$ and $c <_2 a <_2 b$ is $d(<_1, <_2) = d(<_2, <_1) = 2$. Note that $d(<_1, <_2)$ amounts to the smallest number of successive switches of neighbored elements which are needed to transform $<_1$ into $<_2$. This is precisely the number of switches executed by the well-known bubble-sort algorithm.

Definition 8 *Let* $\mathcal{P} = (P, <)$ *be a prioritized program and let* $A \in \mathcal{AS}(\mathcal{P})$. *The preference violation degree of* A *in* $\mathcal{P}$, *denoted* $pvd_{\mathcal{P}}(A)$, *is the minimum distance from any full prioritization of* $\mathcal{P}$ *to any fully prioritized program* $\mathcal{P}' = (P, <')$ *such that* A *is a preferred answer set of* $\mathcal{P}'$, *i.e.,*

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

The preference violation degree of $\mathcal{P}$, $pvd(\mathcal{P})$, *is defined by* $pvd(\mathcal{P}) = \min\{pvd_{\mathcal{P}}(A) \mid A \in \mathcal{AS}(P)\}$.

Now the weakly preferred answer sets are those answer sets which minimize preference violation.

Definition 9 *Let* $\mathcal{P} = (P, <)$ *be a prioritized program. Then,* $A \in \mathcal{AS}(P)$ *is a weakly preferred answer set of* $\mathcal{P}$ *iff* $pvd_{\mathcal{P}}(A) = pvd(\mathcal{P})$.

Example 4 *In the bird & penguin example,* A_1 *is the unique preferred answer set of* $\mathcal{P}$. *Clearly, every preferred answer set* A *of any prioritized program* $\mathcal{P}$ *has* $pvd_{\mathcal{P}}(A) = 0$, *and thus* A *is a weakly preferred answer set of* $\mathcal{P}$. *Thus,* A_1 *is the single weakly preferred answer set of the program.*

Example 5 *Reconsider the program in Example 3. Its answer set* $A = \{b\}$ *is not preferred. Switching the two rules, the resulting prioritized program* $\mathcal{P}'$ *has* $C_{\mathcal{A}_{\mathcal{P}'}}(A) = \{b\}$, *thus* A *is preferred for* $\mathcal{P}'$. *Hence* $pvd_{\mathcal{P}}(A) = pvd(\mathcal{P}) = 1$ *and* A *is a weakly preferred answer set of* $\mathcal{P}$.

Example 6 *Consider the following program* $\mathcal{P}$:

(1) a :- not c.
(2) c :- not b.
(3) $\neg d$:- not b.
(4) b :- not $\neg b, a$.

[2] The definition in (Brewka & Eiter 1999) uses ordinals and deals with possibly infinite M. Ours is equivalent on finite M.

$\mathcal{P}$ *has the answer sets* $A_1 = \{a, b\}$ *and* $A_2 = \{c, \neg d\}$. *Imposing* $(i) < (j)$ *iff* $i < j$, *none is preferred. We have* $pvd_{\mathcal{P}}(A_1) = 2$: *(2) and (3) are zombies in the dual reduct which are only defeatable by (4), which must be moved in front of them; this takes two switches. On the other hand,* $pvd_{\mathcal{P}}(A_2) = 1$: *the single zombie (1) in the dual reduct is defeated if (2) is moved in front of it (here, (4) is a dead rule). Hence,* $pvd(\mathcal{P}) = 1$, *and* A_2 *is the single weakly preferred answer set of* $\mathcal{P}$.

For further background and examples on preferred and weakly preferred answer sets, we refer to (Brewka & Eiter 1999).

From Preferred Answer Sets to DLP

All translations in this paper rely on a kind of meta-interpretation technique: We give a program P_{I_p} and a representation $F(\mathcal{P})$ of an arbitrary propositional prioritized program $\mathcal{P}$ as a set of facts, such that $\mathcal{PAS}(\mathcal{P}) = \{\pi(A) \mid A \in \mathcal{AS}(P_{I_p} \cup F(\mathcal{P}))\}$, where π is a simple projection function.

Representing a prioritized program

The translation $F(\mathcal{P})$ is as follows.

- For each rule

  $$c \text{ :- } a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n.$$

  of the program $\mathcal{P}$, $F(\mathcal{P})$ contains the following facts:

  $$rule(r).\ head(c, r).\ pbl(a_1, r).\ \ldots\ pbl(a_m, r).\ nbl(b_1, r).\ \ldots\ nbl(b_n, r).$$

- For each pair of complementary literals $\ell, \neg\ell$ occurring in the program, we add a fact $compl(\ell, \neg\ell)$.
- For each rule preference $r < r'$ that belongs to the transitive reduction of $<$, we add a fact $pr(r, r')$. to $F(\mathcal{P})$.

Example 7 *The program of the bird & penguin example is represented by the following facts:*

$rule(r1)$. $head(peng, r1)$.
$rule(r2)$. $head(bird, r2)$.
$rule(r3)$. $head(neg\_flies, r3)$.
$pbl(peng, r3)$. $nbl(flies, r3)$.
$rule(r4)$. $head(flies, r4)$.
$pbl(bird, r4)$. $nbl(neg\_flies, r4)$.
$compl(flies, neg\_flies)$.
$pr(r1, r2)$. $pr(r2, r3)$. $pr(r3, r4)$.

Meta-interpreter program

The meta-interpreter P_{I_p} consists of two parts: one for representing an answer set (P_{I_a}), and another one for checking preferredness.

Representing an answer set We define a predicate $in\_AS(.)$ which is true for the literals in an answer set of $\mathcal{P}$. A literal is in an answer set, if it occurs in the head of a rule whose body is not false.

$$in\_AS(X) \text{ :- } head(X, Y),\ \texttt{not}\ pos\_body\_false(Y),\ \texttt{not}\ neg\_body\_false(Y).$$

The positive part of a body is false, if one of its literals is not in the answer set, while the negative part of a body is false, if one of its literals is in the answer set.

$$
\begin{aligned}
pos\_body\_false(Y) &:- pbl(X,Y), \texttt{not}\ in\_AS(X).\\
neg\_body\_false(Y) &:- nbl(X,Y), in\_AS(X).
\end{aligned}
$$

Each answer set needs to be consistent; we thus add an integrity constraint which rejects answer sets containing complementary literals.

$$:- compl(X,Y), in\_AS(X), in\_AS(Y).$$

This program (call it P_{I_a}) is all we need for representing answer sets in P_{I_p}. Each answer set of P_{I_a} plus $F(\mathcal{P})$ represents an answer set of $\mathcal{P}$. Let π be defined by $\pi(A) = \{\ell \mid in\_AS(\ell) \in A\}$. Then we can state the following:

Proposition 1 *Let $\mathcal{P} = (P,<)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{AS}(P_{I_a} \cup F(\mathcal{P}))$ then $\pi(A) \in \mathcal{AS}(\mathcal{P})$, and (ii) for each $A \in \mathcal{AS}(\mathcal{P})$, there exists a single $A' \in \mathcal{AS}(P_{I_a} \cup F(\mathcal{P}))$ such that $\pi(A') = A$.*

Checking preferredness According to Def. 6, we have to create all fully prioritized programs $\mathcal{FP}(\mathcal{P})$ of $\mathcal{P}$ to determine its preferred answer sets. To this end, we add code to guess a total order on the rules which refines $<$:

$$
\begin{aligned}
pr(X,Y)\ \texttt{v}\ pr(Y,X) &:- rule(X),\ rule(Y),\ X\ != Y.\\
pr(X,Z) &:- pr(X,Y),\ pr(Y,Z).\\
&:- pr(X,X).
\end{aligned}
$$

The rules state the axioms of totality, transitivity, and irreflexivity of a total order. Note that it would be possible to replace the disjunctive guessing rule by two rules involving unstratified negation. However, the disjunctive version is more readable.

Next we build the set $C_{\mathcal{P}}$ from Def. 3 where $\mathcal{P}' =^X \mathcal{P}$. To this end, we do not compute the sets S_i as in the definitions – clearly one rule can contribute at most one element to $C_{\mathcal{P}}$ and we represent this fact using the predicate $lit(.,.)$. We first observe that duplicate rules arising in the dual reduct $\mathcal{P}$ need no special care, since only the first occurrence of a rule from $\mathcal{P}'$ is relevant for the value of $C_{\mathcal{P}'}$; for later occurrences of duplicates always $S_i = S_{i-1}$ will hold.

In Def. 3, a condition when $H(r_i)$ is not added is stated, while $lit(.,.)$ represents the opposite, so we negate the condition: (β) is actually itself a conjunction $\gamma \wedge \delta$, so the condition we are interested in is

$$\neg(\alpha \vee (\gamma \wedge \delta)) \equiv (\neg\alpha \wedge \neg\gamma) \vee (\neg\alpha \wedge \neg\delta).$$

We call condition α *local defeat* (by rules of higher priority) and γ *global defeat* (by the answer set).

Def. 3 applies only to prerequisite-free programs, so for the general case we also have to include the definition of the dual Gelfond-Lifschitz reduct, which amounts to stating that a rule having a prerequisite ℓ not in the answer set must not be considered. The encoding is then straightforward:

$$
\begin{aligned}
lit(X,Y) &:- head(X,Y), \texttt{not}\ pos\_body\_false(Y),\\
&\qquad \texttt{not}\ defeat\_local(Y), \texttt{not}\ in\_AS(X).\\
lit(X,Y) &:- head(X,Y), \texttt{not}\ pos\_body\_false(Y),\\
&\qquad \texttt{not}\ defeat\_local(Y), \texttt{not}\ defeat\_global(Y).\\
defeat\_local(Y) &:- nbl(X,Y), lit(X,Y1), pr(Y1,Y).\\
defeat\_global(Y) &:- nbl(X,Y), in\_AS(X).
\end{aligned}
$$

The set $C_{\mathcal{P}}$ is the union of all literals in $lit(.,.)$:

$$in\_CP(X) :- lit(X,Y).$$

Finally, according to Def. 4, a preferred answer set A must satisfy $A = C_{\mathcal{P}'}(A)$, so we formulate integrity constraints which discard answer sets violating this condition:

$$
\begin{aligned}
&:- in\_CP(X), \texttt{not}\ in\_AS(X).\\
&:- in\_AS(X), \texttt{not}\ in\_CP(X).
\end{aligned}
$$

The last constraint is in fact redundant and can be dropped, because $S \in \mathcal{AS}(\mathcal{P})$ (the first constraint is not violated) and $C_{\mathcal{P}} \subseteq S$ jointly imply that $C_{\mathcal{P}} = S$. Indeed, suppose $C_{\mathcal{P}} \neq S$ werer true. Then, some $\ell \in S \setminus C_{\mathcal{P}}$ must exist, which means that a generating rule r w.r.t. S must exist such that $H(r) = \ell$ and $S \models B(r)$. According to Def. 3, $(\alpha) \vee (\beta)$ must hold for S_r, otherwise $\ell \in C_{\mathcal{P}}$ would hold. Now (β) cannot hold, since S cannot defeat r because $A \models B(r)$. Thus (α) must hold. This implies that some $\ell' \in S_{r-1}$ defeats r such that $\ell' \notin S$. Sine $S_{r-1} \subseteq C_{\mathcal{P}}$, it follows $C_{\mathcal{P}} \nsubseteq S$. This is a contradiction.

This completes the meta-interpreter P_{I_p}. A compact listing of it is given in the Appendix. We have:

Proposition 2 *Let $\mathcal{P} = (P,<)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{AS}(P_{I_p} \cup F(\mathcal{P}))$ then $\pi(A) \in \mathcal{PAS}(\mathcal{P})$, and (ii) for each $A \in \mathcal{PAS}(\mathcal{P})$, there exists some $A' \in \mathcal{AS}(P_{I_p} \cup F(\mathcal{P}))$ such that $\pi(A') = A$.*

Example 8 *For the bird & penguin example, $P_{I_p} \cup F(\mathcal{P})$ has one answer set, which contains $in\_AS(peng)$, $in\_AS(bird)$, and $in\_AS(\neg flies)$.*

From Weakly Preferred Answer Sets to DLP^w

The transition from an interpreter for preferred answer sets to one for weakly preferred answer sets is simple – just a few clauses have to be added and one has to be slightly altered.

For weakly preferred answer sets, we have to generate a second total ordering (called $pr1$), which needs not be compatible with the input partial order.

$$
\begin{aligned}
pr1(X,Y)\ \texttt{v}\ pr1(Y,X) &:- rule(X),\ rule(Y),\ X\ != Y.\\
pr1(X,Z) &:- pr1(X,Y),\ pr1(Y,Z).\\
&:- pr1(X,X).
\end{aligned}
$$

This ordering should be used to determine the preferred answer sets. Since the given totalization of the input ordering occurs just in one rule of the original program, we just have to update this rule:

$$defeat\_local(Y) :- nbl(X,Y),\ lit(X,Y1),\ pr1(Y1,Y).$$

Finally, we want to keep only those orderings which minimize the differences to some totalization of an input ordering. To this end, we state a weak constraint, where each difference in the orderings gets a penalty of one (we don't need the levelling concept here).

$$:\sim pr(X,Y), pr1(Y,X).\ [1:1]$$

In this way, each answer set A will be weighted with $pvd_{\mathcal{P}}(A)$, and the optimal answer sets minimize this number, which corresponds exactly to Defs. 7, 8, and 9.

Call the resulting interpreter P_{I_w} (a compact listing is given in the Appendix). We have:

Proposition 3 *Let $\mathcal{P} = (P, <)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{OAS}(P_{I_w} \cup F(\mathcal{P}))$ then $\pi(A) \in \mathcal{WAS}(\mathcal{P})$, and (ii) for each $A \in \mathcal{WAS}(\mathcal{P})$, there exists some $A' \in \mathcal{OAS}(P_{I_w} \cup F(\mathcal{P}))$ such that $\pi(A') = A$.*

Example 9 *Reconsider Ex. 3, which does not have any preferred answer set. $P_{I_w} \cup F(\mathcal{P})$ has one optimal answer set (with weight 1 in level 1) containing $in\_AS(b)$, $pr(r1, r2)$, and $pr1(r2, r1)$, which is consistent with Ex. 5.*

Example 10 *Reconsider Ex. 6, which does not have any preferred answer set either. $P_{I_w} \cup F(\mathcal{P})$ has one optimal answer set (with weight 1 in level 1) containing $in\_AS(c)$, $in\_AS(\neg d)$, $pr(r1, r2)$, and $pr1(r2, r1)$, where the pair $(r1, r2)$ is the only difference between pr and $pr1$, consistent with Ex. 6.*

Deterministic Preferredness Checking

The method we provided above non-deterministically generates, given a prioritized program $\mathcal{P} = (P, <)$ and an answer set of $\mathcal{P}$, all full prioritizations of $\mathcal{P}$ and tests them.

In (Brewka & Eiter 1999) a graph-based algorithm was described which checks preferredness of an answer set A deterministically without refining $<$ to a total order. In general, this method is much more efficient.

This approach works as follows: A labeled directed graph $G(\mathcal{P}, A)$ is constructed, whose vertices are the rules P, and an edge leads from r to r' if $r < r'$. Each vertex r is labeled "g" if r is generating w.r.t A, "z" if it is a zombie, and "i" (for irrelevant) otherwise. The following algorithm then performs a kind of topological sorting for deciding whether an answer set A is preferred, and outputs a suitable full prioritization of $\mathcal{P}$:

Algorithm FULL-ORDER

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

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

Method:

Step 1. Construct the graph $G = G(\mathcal{P}, A)$, and initialize $S := \emptyset$, $<' := \emptyset$.

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 with no incoming edge, such that either r is not labeled "z" or r is defeated by S. If no such r exists, then output "no" and halt.

Step 4. If r is labeled "g", then set $S := S \cup H(r)$.

Step 5. Remove r from G, and continue at Step 2.

A discussion of this algorithm is given in (Brewka & Eiter 1999). Note that it is non-deterministic in Step 3. A deterministic variant of it can be used for merely deciding preferredness of A: rather than some arbitrary source r, *all* sources r satisfying the condition are selected in Step 3 and then removed in parallel in Step 5. As easily seen, this is feasible since removability of a source r is monotone, i.e., can not be destroyed by removing any other source r' before. Thus, A is a preferred answer set iff the algorithm stops with the empty graph, i.e., all vertices are removed.

This deterministic algorithm can be readily encoded in DLV. The idea is to use time stamps for modeling the iterations through Steps 2–5. Since the number of steps is bounded by the number of rules in $\mathcal{P}$, we reuse rule-ids as time stamps:

$$time(T) :- rule(T).$$

Time stamps are ordered by DLV's built-in order $<$ on constants. The first (least) time stamp is used for the stage after the first run through Steps 2–5.

We use predicates g and z for rule labels "g" and "z", respectively, which are attached as follows (label "i" is not of interest and thus omitted):

$$\begin{aligned} g(R) &:- rule(R), \texttt{not}\ neg\_body\_false(R), \\ &\quad\ \texttt{not}\ pos\_body\_false(R). \\ z(R) &:- rule(R), \texttt{not}\ pos\_body\_false(R), \\ &\quad\ head(X, R), \texttt{not}\ in\_AS(X). \end{aligned}$$

Initially, only sources which are not zombies can be removed from the graph. We use a predicate $nosource0(R)$, which informally means that R is *not* a source of G, and a predicate $remove(R, T)$ which means that at time T, the vertex R is no longer in G:

$$\begin{aligned} nosource0(R) &:- pr(R1, R). \\ remove(R, T) &:- rule(R), \texttt{not}\ nosource0(R), \\ &\quad\ \texttt{not}\ z(R), time(T). \end{aligned}$$

At other stages of the iteration, we can remove all rules satisfying the condition of Step 3. We use a predicate $nosource(R, T)$ which expresses that R is not a source at time T.

$$\begin{aligned} nosource(R, T) &:- pr(R1, R), time(T), \\ &\quad\ \texttt{not}\ remove(R1, T). \\ remove(R, T1) &:- rule(R), \texttt{not}\ nosource(R, T), \\ &\quad\ time(T), time(T1), T < T1, \\ &\quad\ \texttt{not}\ z(R), \texttt{not}\ remove(R, T). \\ remove(R, T1) &:- rule(R), \texttt{not}\ nosource(R, T), \\ &\quad\ time(T), time(T1), T < T1, \\ &\quad\ z(R), nbl(X, R), s(X, T). \end{aligned}$$

According to Step 4, we must add the head $H(r)$ of a generating rule which is to be removed in Step 5, to the set S there. We represent this using a predicate $s(X, T)$, which informally means that X belongs to S at time point T, and add the rule:

$$s(X, T) :- remove(R, T), g(R), head(X, R).$$

Finally, according to Step 2 we have to check whether all rules have been removed in the processing of the graph G. This is done by using a predicate $removed$ for the projection of $remove$ to rules and the following rule plus a constraint:

$$\begin{aligned} removed(R) &:- remove(R, T). \\ &:- rule(R), \texttt{not}\ removed(R). \end{aligned}$$

The resulting meta-interpreter P_{I_g} is given in the Appendix. Note that P_{I_g} is in general also more efficient than

P_{I_p}, since unnecessary totalizations of the partial order can be avoided with P_{I_g}. By virtue of the results in (Brewka & Eiter 1999) (in particular, Lemma 7.2), we can state the follwing result:

Proposition 4 *Let $\mathcal{P} = (P, <)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{AS}(P_{I_g} \cup F(\mathcal{P}))$ then $\pi(A) \in \mathcal{PAS}(\mathcal{P})$, and (ii) for each $A \in \mathcal{PAS}(\mathcal{P})$, there exists some $A' \in \mathcal{AS}(P_{I_g} \cup F(\mathcal{P}))$ such that $\pi(A') = A$.*

Example 11 *Consider the program in Ex. 6 and assume priorities $(1) < (3)$, $(2) < (4)$, and $(4) < (3)$. Suppose preferredness of $A_2 = \{c, \neg d\}$ is checked. Then, the atoms $z(r1)$, $g(r2)$, and $g(r3)$ representing labels are derived, as well as $nosource0(r4)$ and $nosource0(r3)$. Both $r1$ and $r2$ are sources, but $r1$ is labeled "z", so only $remove(r2, ri)$ and $s(c, ri)$ is derived for $i = 1, \ldots, 4$. Thus, $nosource(ri, r1)$ is derived only for $i = 3$. Since $s(c, r1)$ holds, too, we can derive $remove(r1, ri)$ and $remove(r4, ri)$ for $i = 2, 3, 4$. Neither $s(a, ri)$ nor $s(b, ri)$ are derived since $g(r1)$ and $g(r4)$ do not hold. Finally, $remove(r3, ri)$ and $s(\neg d, ri)$ for $i = 3, 4$ are derived and $removed(ri)$ holds for $i = 1, \ldots, 4$, satisfying the final constraint introduced above. Thus, A_2 is a preferred answer set.*

Usage of the prototype

To experiment with the meta-interpreters we showed in this paper, a current version of DLV can be obtained through the DLV project page (Faber & Pfeifer since 1996), where you will also find detailed documentation and further information on the project. At `http://www.dbai.tuwien.ac.at/proj/dlv/preferred/` you will find downloadable versions of all meta-interpreters as well as several examples.

Let us assume that the code from Example 7 is stored in a file called "penguin" and the meta-interpreter for preferred answer sets in a file called "pas". Then the following command-line will compute and print the single preferred answer set for our running penguin example:

```
dlv pas penguin -filter=in_AS
```

and the output will look like

```
{in_AS(peng), in_AS(bird), in_AS(neg_flies)}
```

`-filter` is a special feature in DLV that allows us to print only those predicates for each answer that we are actually interested in. Without this option we would also see $head$, $comp$, $defeat\_local$ and all other auxiliary predicates in the current example.

Now let us consider weakly preferred answer sets. Assume that the program from Example 6 is converted to our internal representation using $head$, pbl and nbl and saved in a file called "weak" and that the meta-interpreter for weakly preferred answer sets is stored in a file called "wpas".

When we issue the command

```
dlv wpas weak -filter=in_AS
```

we obtain the following output:

```
Optimal answer set: {in_AS(c), in_AS(neg_d)}
Cost ([Weight:Level]): <[1:1]>
```

which matches the result that we described in Example 6. It is worth noting that the weight part of the cost of the optimal answer sets is always equal to $pvd(\mathcal{P})$.

Consequently, the command

```
dlv wpas penguin -filter=in_AS
```

computes the same result as shown in the first example, with a weight of 0:

```
Optimal answer set: {in_AS(peng), in_AS(bird),
                     in_AS(neg_flies)}
Cost ([Weight:Level]): <[0:1]>
```

Related Work

In (Gelfond & Son 1997) the idea of meta-interpretation has been used for defining the semantics of the language $\mathcal{L}_0$. However, there are some differences w.r.t. the approach presented here.

First of all, the semantics of $\mathcal{L}_0$ is defined only by means of a meta-interpreter, while our approach implements a semantics which has been defined previously without meta-interpretation techniques.

Second, the interpretation program in (Gelfond & Son 1997) uses lists for representing aggregations of literals and conditions on them, in particular "for all" conditions. Such lists cannot be used in datalog programs, as arbitrarily deep function nesting is required for the list concept. We avoid these aggregations by using rule identifiers and default negation.

Third, in our approach we extend a general answer sets meta-interpreter, thus clearly separating the representation of answer sets and prioritization. In the meta-interpreter presented in (Gelfond & Son 1997), this distinction is not obvious.

Extensions and Further Work

The techniques as presented here work on ground input – handling non-ground programs is also feasible, but unless function symbols are allowed at the code level (which is currently not the case in DLV), this is not completely straightforward. Extending our work to deal with such cases and creating a DLV frontend for prioritized program evaluation is an interesting research issue, as is investigating whether we can exploit a variant of the deterministic graph algorithm also for the computation of weakly preferred answer sets.

References

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

Brewka, G., and Eiter, T. 1999. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence* 109(1-2):297–356.

Brewka, G., and Eiter, T. 2000. Prioritizing Default Logic. In Hölldobler, S., ed., *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*. Kluwer. 27–45.

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

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

Buccafurri, F.; Leone, N.; and Rullo, P. 1997. Strong and Weak Constraints in Disjunctive Datalog. In Dix, J.; Furbach, U.; and Nerode, A., eds., *Proc. LPNMR'97*, LNAI 1265, 2–17. Springer.

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Semantics and Expressiveness of Disjunctive Ordered Logic. *Annals of Mathematics and Artificial Intelligence*. To appear. Abstract in Proc. KR '98.

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

Delgrande, J.; Schaub, T.; and Tompits, H. 2000. Prioritized Default Logic Revisited: A Compilation of Brewka and Eiter's Approach. In Ojeda-Aciego, M.; de Guzmán, I. P.; Brewka, G.; and Moniz Pereira, L., eds., *Proc. JELIA 2000*, LNCS 1919. Springer.

Faber, W.; Leone, N.; and Pfeifer, G. 1999. Pushing Goal Derivation in DLP Computations. *Proc. LPNMR'99)*, LNAI 1730. 177–191. Springer.

Faber, W., and Pfeifer, G. since 1996. `dlv` homepage. `<URL: http://www.dbai.tuwien.ac.at/proj/dlv/>`.

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

Gelfond, M., and Son, T. C. 1997. Reasoning with Prioritized Defaults. In *Proc. LPKR '97*, 164–223. LNCS 1471. Springer.

Lifschitz, V. 1996. Foundations of logic programming. In Brewka, G., ed., *Principles of Knowledge Representation*. Stanford: CSLI Publications. 69–127.

Marek, V., and Truszczyński, M. 1993. *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer.

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

Sakama, C., and Inoue, K. 1996. Representing Priorities in Logic Programs. In *Proc. JICSLP'96*, 82–96. MIT Press.

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

Appendix: Meta-Interpreter for Preferred Answer Sets

```
% Represent answer sets:
  in_AS(X) :- head(X,Y), not pos_body_false(Y),
                         not neg_body_false(Y).
  pos_body_false(Y) :- pbl(X,Y), not in_AS(X).
  neg_body_false(Y) :- nbl(X,Y), in_AS(X).
  :- compl(X,Y), in_AS(X), in_AS(Y).

% For full prioritization: refine pr to a total ordering.
  pr(X,Y) v pr(Y,X) :- rule(X), rule(Y), X != Y.
  pr(X,Z) :- pr(X,Y), pr(Y,Z).
  :- pr(X,X).

% Check dual reduct: Build sets S_i, use rule ids as indices i.
% lit(X,r) means that the literal x occurs in the set S_r.
   lit(X,Y) :- head(X,Y), not pos_body_false(Y),
               not defeat_local(Y), not in_AS(X).
   lit(X,Y) :- head(X,Y), not pos_body_false(Y),
               not defeat_local(Y), not defeat_global(Y).
   defeat_local(Y)  :- nbl(X,Y), lit(X,Y1), pr(Y1,Y).
   defeat_global(Y) :- nbl(X,Y), in_AS(X).

% Include literal into CP(.).
   in_CP(X) :- lit(X,Y).
   :- in_CP(X), not in_AS(X).
```

Appendix: Meta-Interpreter for Weakly Preferred Answer Sets

```
% Represent answer sets:
  in_AS(X) :- head(X,Y), not pos_body_false(Y),
                         not neg_body_false(Y).
  pos_body_false(Y) :- pbl(X,Y), not in_AS(X).
  neg_body_false(Y) :- nbl(X,Y), in_AS(X).
  :- compl(X,Y), in_AS(X), in_AS(Y).

% For full prioritization: Refine pr to a total ordering.
  pr(X,Y) v pr(Y,X) :- rule(X),rule(Y), X != Y.
  pr(X,Z) :- pr(X,Y), pr(Y,Z).
  :- pr(X,X).

% Weakly preferred answer sets: Create a total ordering pr1,
% as close to pr as possible.
  pr1(X,Y) v pr1(Y,X) :- rule(X),rule(Y), X != Y.
  pr1(X,Z) :- pr1(X,Y), pr1(Y,Z).
  :- pr1(X,X).

% Weak constraint: Minimize violations.
  :~ rule(X), rule(Y), pr(X,Y), pr1(Y,X). [1:1]

% Check dual reduct: Build sets S_i, use rule ids as indices i.
% lit(X,r) means that the literal x occurs in the set S_r.
   lit(X,Y) :- head(X,Y), not pos_body_false(Y),
               not defeat_local(Y), not in_AS(X).
   lit(X,Y) :- head(X,Y), not pos_body_false(Y),
               not defeat_local(Y), not defeat_global(Y).
   defeat_local(Y)  :- nbl(X,Y), lit(X,Y1), pr1(Y1,Y).
   defeat_global(Y) :- nbl(X,Y), in_AS(X).

% Include literal into CP(.).
   in_CP(X) :- lit(X,Y).
   :- in_CP(X), not in_AS(X).
```

Appendix: Meta-Interpreter for Preferred Answer Sets Using Deterministic Preferredness Checking

```
% Represent answer sets:
  in_AS(X) :- head(X,Y), not pos_body_false(Y),
                         not neg_body_false(Y).
  pos_body_false(Y) :- pbl(X,Y), not in_AS(X).
  neg_body_false(Y) :- nbl(X,Y), in_AS(X).
  :- compl(X,Y), in_AS(X), in_AS(Y).

% Label 'g' nodes and 'z' nodes (other labels are uninteresting):
  g(R) :- rule(R), not neg_body_false(R), not pos_body_false(R).
  z(R) :- rule(R), not pos_body_false(R), head(X,R), not in_AS(X).

% Use rules ids as time stamps.
  time(T) :- rule(T).

% Initial step of the algorithm: Consider global source nodes.
% Only non-z nodes can be removed.
  nosource0(R) :- pr(R1,R).
  remove(R,T)  :- rule(R), not nosource0(R), not z(R), time(T).

% Other steps in the algorithm: Remove non-z nodes and, under some
% conditions, also z-nodes.
  nosource(R,T) :- pr(R1,R), time(T), not remove(R1,T).
  remove(R,T1)  :- rule(R), not nosource(R,T), time(T), time(T1),
                   T < T1, not z(R), not remove(R,T).
  remove(R,T1)  :- rule(R), not nosource(R,T), time(T), time(T1),
                   T < T1, z(R), nbl(X,R), s(X,T).

% Add the head if a removed generating rule to the set T.
  s(X,T) :- remove(R,T), g(R), head(X,R).

% Check whether all rules are removed.
  removed(R) :- remove(R,T).
  :- rule(R), not removed(R).
```