Computing Preferred Answer Sets by Meta-Interpretation in Answer Set Programming∗

THOMAS EITER, WOLFGANG FABER
Institut für Informationssysteme 184/3, TU Wien
Favoritenstr. 9-11, 1040 Wien, Austria
(*e-mail:* {eiter,faber}@kr.tuwien.ac.at)

NICOLA LEONE
Department of Mathematics, University of Calabria
87030 Rende (CS), Italy
(*e-mail:* leone@unical.it)

GERALD PFEIFER
Institut für Informationssysteme 184/2, TU Wien
Favoritenstr. 9-11, 1040 Wien, Austria
(*e-mail:* pfeifer@dbai.tuwien.ac.at)

Abstract

Most recently, Answer Set Programming (ASP) has been attracting interest as a new paradigm for problem solving. An important aspect, for which several approaches have been presented, is the handling of preferences between rules. In this paper, we consider the problem of implementing preference handling approaches by means of meta-interpreters in Answer Set Programming. In particular, we consider the preferred answer set approaches by Brewka and Eiter, by Delgrande, Schaub and Tompits, and by Wang, Zhou and Lin. We present suitable meta-interpreters for these semantics using DLV, which is an efficient engine for ASP. Moreover, we also present a meta-interpreter for the weakly preferred answer set approach by Brewka and Eiter, which uses the weak constraint feature of DLV as a tool for expressing and solving an underlying optimization problem. We also consider advanced meta-interpreters, which make use of graph-based characterizations and often allow for more efficient computations. Our approach shows the suitability of ASP in general and of DLV in particular for fast prototyping. This can be fruitfully exploited for experimenting with new languages and knowledge-representation formalisms.

1 Introduction

Handling preference information plays an important role in applications of knowledge representation and reasoning. In the context of logic programs and related formalisms, numerous approaches for adding preference information have been proposed, including (Baader & Hollunder, 1995; Brewka, 1994; Brewka, 1996; Buccafurri *et al.*, 1999; Delgrande &

∗ This paper is a revised and extended version of a preliminary paper in: Alessandro Provetti and Tran Cao Son, editors, *Proc. AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, Stanford CA, March 2001, AAAI Press, ISBN 1-57735-129-0.

Schaub, 2000; Gelfond & Son, 1997; Marek & Truszczyński, 1993; Rintanen, 1998; **?**; Zhang & Foo, 1997). They have been designed for purposes such as capturing specificity or normative preference; see e.g. (Brewka & Eiter, 1999; Delgrande & Schaub, 2000; **?**) for reviews and comparisons.

Answer Set Programming (ASP) is gaining importance as a new paradigm for (declarative) problem solving. Basically, ASP is function-free disjunctive logic programming with negation as failure and strong negation under the Answer Set Semantics (Gelfond & Lifschitz, 1991). Several attempts have been made to combine ASP and preferential information, resulting in a number of different semantics for preferential ASP, such as those in (Brewka & Eiter, 1999; Wang *et al.*, 2000; Delgrande *et al.*, 2000a), on which we will focus in this paper; (Schaub & Wang, 2001) provides a unifying framework for them.

The following example is a classical situation for the use of preference information in non-monotonic reasoning.

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

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

This program has two answers sets: $A_1 = \{penguin, bird, \neg flies\}$ and $A_2 = \{penguin, bird, flies\}$, where we conclude $\neg flies$ and $flies$, respectively. 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, as $flies$ is concluded from (4), which has lower priority than (3) where we conclude $\neg flies$.

Interestingly, even if this example is very simple, various preference semantics arrive at different results. Furthermore, semantics which coincide on this example may well yield different results on other examples. Since evaluating a semantics on a number of benchmark examples, each of which possibly involving several rules, quickly becomes a tedious task, one would like to have a (quick) implementation of a semantics at hand, such that experimentation can be done using computer support. Exploring a (large) number of examples, which helps in assessing the behavior of a semantics, then can be performed significantly faster, easier, and less error prone than by manual evaluation.

In this paper, we address this issue and explore the implementation of preference semantics for logic programs by means of a powerful technique based on Answer Set Programming (ASP), which can be seen as a sort of *meta-programming in ASP*. There, a given logic program $\mathcal{P}$ with preferences is encoded by a suitable set of facts $F(\mathcal{P})$, which are added to a fixed "meta-program" P_I, such that the intended answer sets of $\mathcal{P}$ are determined by the answer sets of the logic program $P_I \cup F(\mathcal{P})$. The salient feature is that this P_I is universal, i.e., it is the same for all input programs $\mathcal{P}$.

Pursuing such a meta-programming approach has several advantages: On the one hand, leveraging the underlying "programming" language as much as possible facilitates quick prototype implementations for the preference semantics at hand. On the other hand, this novel application of answer set programming improves the understanding of how the

declarative nature of answer set semantics works, in that we see which methods are needed in order to express elements such as iterative and recursive definitions, respectively.

We recall that meta-interpretation is well-established in Prolog-style logic programming, but due to the differences in semantics, Prolog-style meta-interpreters noticeably differ from those presented in this paper. More specifically, Prolog-style meta-interpreters usually formalize resolution and its control, thereby exhibiting a rather procedural reading. Our ASP meta-interpreters, on the other hand, are formulated in a truly declarative way. This is more amenable to realizing interpreters for KR languages which have a purely declarative reading, and where syntactic elements such as the order of rules, or the order of literals in the body of a rule are not important.

However, the fundamental concept of ASP and Prolog-style meta-interpreters is the same: Representing programs using a formalism very close to the one they are written in, and providing a modular "interpreter" which evaluates the represented program by means of the common semantics, such that the results of the original and the interpreted programs are in correspondence. We will not go into further detail concerning the differences between Prolog-style and ASP-style meta-interpretation in this paper, but focus on presenting ASP-style meta-interpreters. It is worthwhile noting that meta-interpretation is not completely new in ASP; a similar technique was used previously in (Gelfond & Son, 1997) for defining the semantics of logic programming with defeasible rules (cf. Section 6).

In this paper we focus on three similar, yet different semantics for prioritized logic programs, namely the preference semantics by Brewka and Eiter (Brewka & Eiter, 1999), Wang, Zhou and Lin (Wang *et al.*, 2000), and Delgrande, Schaub and Tompits (Delgrande *et al.*, 2000a), which we refer to as B-preferred, W-preferred, and D-preferred answer set semantics, respectively. We present ASP meta-programs P_{I_B}, P_{I_W}, and P_{I_D} such that the answer sets of $P_{I_B} \cup F(\mathcal{P})$, $P_{I_W} \cup F(\mathcal{P})$, and $P_{I_D} \cup F(\mathcal{P})$ correspond (modulo a simple projection function) precisely to the B-, W-, and D-preferred answer sets of $\mathcal{P}$. This way, by running $F(\mathcal{P})$ together with the corresponding meta-program on the DLV system (Faber *et al.*, 1999; Faber & Pfeifer, since 1996), we compute the preferred answer sets of $\mathcal{P}$ in a simple and elegant way. For B-preferred answer sets we also provide an alternate meta-program P_{I_g}, which implements a graph-based algorithm that deterministically checks preferredness of an answer set and, in general, is more efficient than P_{I_B}. Note that by suitable adaption of the meta-programs, other ASP engines such as Smodels (Niemelä *et al.*, 2000) can be used as well.

The B-preferred answer set semantics refines previous approaches for adding preferences to default rules by Brewka (Brewka, 1994; Brewka, 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 semantics is that the definition was guided by two general principles which, as argued, a preference semantics should satisfy. As shown in (Brewka & Eiter, 1999), B-preferred answer sets satisfy these principles, while almost all other semantics do not. The W- and D-preferred answer set semantics increasingly strengthen the B-preferred answer set semantics (Schaub & Wang, 2001).

Since, in general, programs having an answer set may lack a preferred answer set, in (Brewka & Eiter, 1999) a relaxed notion, called weakly preferred answer sets, is defined.

We implement this semantics by means of a meta-program $P_{I_{\text{weak}}}$, which takes advantage of the *weak constraints* feature (Buccafurri *et al.*, 1997) of DLV.

Overall, the work reported in this paper is important in several respects:

- We put forward the use of ASP for experimenting new semantics by means of a meta-interpretation technique. The declarativity of logic programs (LPs) provides a new and elegant way of writing meta-interpreters, which is very different from Prolog-style meta-interpretation. Thanks to the high expressiveness of (disjunctive) LPs and DLV's weak constraints, meta-interpreters can be written in a simple and declarative fashion.
- The description of the meta-programs implementing the various preference semantics also has a didactic value: it is a good example how meta-interpreters can be built using ASP. In particular, we also present a core meta-program for plain extended logic programs under answer set semantics, which may be used as a basic building block for the construction of other meta-programs.
- Furthermore, the meta-interpreters provide an actual implementation of preferred and weakly preferred answer sets and allow for easy experimentation of 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) has been reported in (Delgrande *et al.*, 2000b), where programs are mapped 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. Weak constraints make the encoding of weakly preferred answer sets extremely simple and elegant, while that task would have been much more cumbersome otherwise. The meta-interpreters and a suite of example are available at `http://www.dlvsystem.com/preferred/`.

The experience reported in this paper confirms the benefits 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, it fosters *simple and very fast prototyping*, which is useful e.g. in the process of designing and experimenting new languages.

The structure of the remainder of this paper is as follows: In the next section, we provide preliminaries of extended logic programming and answer set semantics. We then develop in Section 3 a basic meta-interpreter program for extended logic programs under the answer set semantics. After that, we consider in Section 4 meta-interpreter programs for B-preferred, W-preferred, and D-preferred answer answer sets. Section 5 is devoted to the refinement of preferred answer sets to weakly preferred answer sets. A discussion of related work is provided in Section 6. Finally, in Section 7 we summarize our results and draw some conclusions.

2 Preliminaries: Logic Programs

Syntax. Logic Programs (LPs) 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 `not` l, where l is a classical literal. Unless stated otherwise, by *literal* we mean a classical literal.

A *rule* r is a formula

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

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 "`:-`" is the *head*, the part to the right is the *body* of r; we omit "`:-`" 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 a rule where $n = 0$.

A *datalog program (LP)* P is a finite set of rules. We call P *positive*, if P is `not` -free (i.e. $\forall r \in P : B^-(r) = \emptyset$); and *normal*, if P is `v` -free (i.e. $\forall r \in P : |H(r)| \leq 1$).

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, $l \geq 1$ is the *priority level* and $w \geq 1$ is the *weight* among the level. The intuition underlying levels and weights is that weak constraints of lower levels are more important than those of higher levels, and among the same level weak constraints with greater weights are more important. 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. $WC(P)$ denotes the set of weak constraints in P. Weak constraints are motivated by having a convenient means to deal with optimization problems such as the Traveling Salesperson problem, or express actions wit costs, cf. (Eiter *et al.*, 2002).

As usual, a term (atom, rule,...) is *ground*, if no variables appear in it.

Semantics. Answer sets for LPs with weak constraints are defined by extending consistent answer sets for LPs as introduced in (Gelfond & Lifschitz, 1991; Lifschitz, 1996). We proceed in three steps: we first define answer sets (1) of ground positive programs, then (2) of arbitrary 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, $Ground(P) = \bigcup_{r \in P} Ground(r)$. 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 program P, if

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

$B(r) \subseteq I$ implies $H(r) \cap I \neq \emptyset$ for every $r \in P$. An interpretation X is an *answer set* for P if it is a minimal set (wrt. set inclusion) which is closed under P.

(2) Let P^I be the *Gelfond-Lifschitz reduct* of a 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.

Example 2
The program

$$a \vee b. \qquad b \vee c. \qquad d \vee \neg d \ :- \ a, c.$$

has three answer sets: $\{a, c, d\}$, $\{a, c, \neg d\}$, and $\{b\}$.

(3) Given a program P with weak constraints, we are interested in the answer sets of the part without weak constraints which minimize the sum of weights of the violated constraints in the highest priority level, and among them those which minimize the sum of weights of the violated constraints in the next lower level, etc. This is expressed by an objective function for P and an answer set A:

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

where w^P_{max} and l^P_{max} denote the maximum weight and maximum level of a weak constraint in P, respectively; $N_i^{A,P}$ 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.

Intuitively, $f_P(i)$ is a factor which guarantees that the impact of violated weak constraints of level i is less than the impact of violated weak constraints of lower levels. More technically, note that $|WC(P)| \cdot w^P_{max} + 1$ is greater than the sum of all weights in the program, and is therefore guaranteed to be greater than any sum of weights of a single level.

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

Example 3
Let us extend the program from Example 2 by the following three weak constraints:

$$:\sim \ a, c. \ [2:1] \qquad :\sim \ \neg d. \ [1:1] \qquad :\sim \ b. \ [3:1]$$

The resulting program P_3 has the single optimal answer set $A_3 = \{a, c, d\}$ with weight 2 in level 1.

ASP systems like DLV (and virtually all others) provide a built-in binary predicate which defines a total order on the constants. In DLV, this predicate is denoted $<$; in this paper, we denote it by lt, in order to avoid confusion with the rule ordering $<$ in programs which will be introduced in Section 4.

3 Meta-Interpreting Answer Set Programs

In this section we show how a normal propositional answer set program can be encoded for and interpreted by a generic meta-interpreter based on the following idea:

We provide a representation $F(P)$ of an arbitrary normal propositional program P [2] as a set of facts and combine these facts with a generic answer set program P_{I_a} such that $\mathcal{AS}(P) = \{\pi(A) \mid A \in \mathcal{AS}(F(P) \cup P_{I_a})\}$, where π is a simple projection function.

3.1 Representing An Answer Set Program

First we translate the propositional answer set program P into a set of facts $F(P)$ as follows:

1. For each rule c :- $a_1, \ldots, a_m$, not $b_1, \ldots,$ not b_n. of the program P, $F(P)$ contains the following facts:

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

   where r is a unique rule identifier.
2. For each pair of complementary literals ℓ and $\neg\ell$ occurring in the program P we explicitly add a fact $compl(\ell, \neg\ell)$. We denote $\neg\ell$ by neg_ℓ in programs.

Example 4
The program of the bird & penguin example is represented by the following facts representing the rules and complementary literals:

$rule(r1).$ $head(penguin, r1).$
$rule(r2).$ $head(bird, r2).$
$rule(r3).$ $head(neg_flies, r3).\ pbl(penguin, r3).\ nbl(flies, r3).$
$rule(r4).$ $head(flies, r4).\ pbl(bird, r4).\ nbl(neg_flies, r4).$
$compl(flies, neg_flies).$

3.2 Basic Meta-interpreter program

Several meta-interpreters that we will present in the following sections consist of two parts: a meta-interpreter program P_{I_a} for representing an answer set, and another one for checking preferredness. In this section we will provide the first part which is common to many meta-interpreters shown in this paper.

Representing an answer set. We define a predicate $in_AS(.)$ which is true for the literals in an answer set of P. A literal is in an answer set if it occurs in the head of a rule whose positive body is definitely true and whose negative body is not false.

$$in_AS(X) \text{ :- } head(X, R), pos_body_true(R), \text{ not } neg_body_false(R).$$

[2] We assume that integrity constraints :- C. are written as equivalent rules bad :- C, not bad. where bad is a predicate not occurring in P.

The positive part of a body is true, if all of its literals are in the answer set. Unfortunately we cannot encode such a universal quantification in one rule. We can identify a simple case: If there are no positive body literals, the body is trivially true.

$$\begin{aligned} pos_body_exists(R) &\ \texttt{:-}\ pbl(X,R). \\ pos_body_true(R) &\ \texttt{:-}\ rule(R), \texttt{not}\ pos_body_exists(R). \end{aligned}$$

However, if positive body literals exist, we will proceed iteratively. To this end, we use the built-in constant ordering lt for defining a successor relation on the positive body literals of each rule, and to identify the first and last literal, respectively, of a positive rule body in this total order. Technically, it is sufficient to define auxiliary relations as follows.

$$\begin{aligned} pbl_inbetween(X,Y,R) &\ \texttt{:-}\ pbl(X,R), pbl(Y,R), pbl(Z,R), lt(X,Z), lt(Z,Y). \\ pbl_notmax(X,R) &\ \texttt{:-}\ pbl(X,R), pbl(Y,R), lt(X,Y). \\ pbl_notmin(X,R) &\ \texttt{:-}\ pbl(X,R), pbl(Y,R), lt(Y,X). \end{aligned}$$

This information can be used to define the notion of the positive body being true up to (w.r.t. the built-in order) some positive body literal. If the positive body is true up to the last literal, the whole positive body is true.

$$\begin{aligned} pos_body_true_upto(R,X) &\ \texttt{:-}\ pbl(X,R), \texttt{not}\ pbl_notmin(X,R), in_AS(X). \\ pos_body_true_upto(R,X) &\ \texttt{:-}\ pos_body_true_upto(R,Y), pbl(X,R), in_AS(X), \\ &\qquad lt(Y,X), \texttt{not}\ pbl_inbetween(Y,X,R). \\ pos_body_true(R) &\ \texttt{:-}\ pos_body_true_upto(R,X), \texttt{not}\ pbl_notmax(X,R). \end{aligned}$$

The negative part of a body is false, if one of its literals is in the answer set.

$$neg_body_false(R) \ \texttt{:-}\ nbl(X,R), in_AS(X).$$

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

$$\texttt{:-}\ compl(X,Y), in_AS(X), in_AS(Y).$$

The rules described above (referred to as P_{I_a} in the sequel) are all we need for representing answer sets. Each answer set of $P_{I_a} \cup F(P)$ represents an answer set of P. Let π be defined by $\pi(A) = \{\ell \mid in_AS(\ell) \in A\}$. Then we can state the following:

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

Proof
(i) $\pi(A)$ must be a consistent set of literals from P, since for each ℓ s.t. $in_AS(\ell) \in A$, $head(\ell, r)$ must hold for some rule r, which, by construction of $F(P)$, only holds for $\ell \in B_P$, and since the constraint $\texttt{:-}\ compl(\ell, \neg\ell), in_AS(\ell), in_AS(\neg\ell).$ must be be satisfied by A (again by construction of $F(P)$) $\{in_AS(\ell), in_AS(\neg\ell)\} \not\subseteq A$ and hence $\{\ell, \neg\ell\} \not\subseteq \pi(A)$ for all $\ell \in B_P$.

Thus, to show that $\pi(A) \in \mathcal{AS}(P)$, it suffices to show that (α) $\pi(A)$ is closed under

$P^{\pi(A)}$, and that (β) $\pi(A) \subseteq T^{\infty}_{P^{\pi(A)}}$ must hold, where $T_{P^{\pi(A)}}$ is the standard T_P operator for $P = P^{\pi(A)}$. Let, for convenience, denote $Q = Ground(P_{I_a} \cup F(P))$.

As for (α), we show that if $r \in P^{\pi(A)}$ such that $B(r) \subseteq \pi(A)$, then there is a rule $h_r \in Q$ such that $H(h_r) = \{in_AS(h)\}$ where $H(r) = \{h\}$ and h_r is applied in A, i.e., $A \models B(h_r)$ and $in_AS(h) \in A$. Let r stem from a rule $r' \in P$. Then, let h_r be the (unique) rule in Q such that $H(h_r) = \{in_AS(h)\}$ (note that $H(r) = H(r') = \{h\}$) and $head(h, r') \in B(h_r)$. Since $B(r) = B^+(r') \subseteq \pi(A)$, we have for each $\ell \in B(r)$ that $in_AS(\ell) \in A$ and, by construction of $F(P)$, that $pbl(\ell, r') \in A$. Since lt induces a linear ordering of $B^+(r')$, it follows by an inductive argument along it that $pos_body_true_upto(r', \ell) \in A$ holds for each $\ell \in B^+(r')$ and that $pos_body_true(r') \in A$. Furthermore, since $B^-(r') \cap \pi(A) = \emptyset$, it holds that $A \not\models B(n_r)$ for each rule $n_r \in Q$ such that $H(n_r) = neg_body_false(r')$. Therefore, $neg_body_false(r') \notin A$. Since $head(h, r') \in A$ by construction of $F(P)$, it follows that $A \models B(h_r)$. Thus, h_r is applied in A, and hence $H(h_r) = in_AS(h) \in A$. This proves (α).

As for (β), it suffices to show that if $in_AS(\ell) \in T^i_{Q^A} \setminus T^{i-1}_{Q^A}$, i.e., $in_AS(\ell)$ is added to A in the i-th step of the least fixpoint iteration for T_{Q^A}, $i \geq 1$, then $\ell \in T^{\infty}_{P^{\pi(A)}}$ holds. Addition of $in_AS(\ell)$ implies that $\ell = H(r)$ for some $r \in P$ such that $neg_body_false(r) \notin A$ and $pos_body_true(r) \in A$. This implies $in_AS(\ell') \in T^{i-1}_{Q^A}$ for each $\ell' \in B^+(r)$ and $in_AS(\ell') \notin A$ for each $\ell' \in B^-(r)$. By an inductive argument, we obtain $B^+(r) \subseteq T^{\infty}_{P^{\pi(A)}}$; therefore, $\ell \in T^{\infty}_{P^{\pi(A)}}$ holds. This proves (β).

(ii) For any $A \in \mathcal{AS}(P)$, let A' be defined as follows (lt is the total order on constants defined in DLV):

$$
\begin{aligned}
A' = {} & F(P) \cup \{in_AS(x) \mid x \in A\} \cup \{pos_body_true(r) \mid \forall x \in B^+(r) : x \in A\} \cup \\
& \{pos_body_true_upto(r, x) \mid x \in B^+(r) \wedge x \in A \wedge \forall y \in B^+(r) : lt(y, x) \rightarrow y \in A\} \cup \\
& \{neg_body_false(r) \mid B^-(r) \cap A \neq \emptyset\} \cup \{pos_body_exists(r) \mid B^+(r) \neq \emptyset\} \cup \\
& \{pbl_notmin(x, r) \mid x \in B^+(r) \wedge \exists y \in B^+(r) : lt(y, x)\} \cup \\
& \{pbl_notmax(x, r) \mid x \in B^+(r) \wedge \exists y \in B^+(r) : lt(x, y)\} \cup \\
& \{pbl_inbetween(x, y, r) \mid x, y \in B^+(r) \wedge (\exists z \in B^+(r) : lt(x, z) \wedge lt(z, y))\}
\end{aligned}
$$

Observe that the set of literals defined by the five last sets (call it A_{stat}) do not depend on A since they have to occur in all answer sets of $P_{I_a} \cup F(P)$. The definitions of A_{stat} directly reflect the corresponding rule structure in P_{I_a} and $F(P)$. Since the inclusion of literals $in_AS(x)$ into A' is determined by the condition $\pi(A') = A$, it is easy to see that all rules defining neg_body_false are satisfied; in the case of $pos_body_true_upto$ and pos_body_true, rule satisfaction can be seen by a constructive argument along the order lt.

To see that A' is minimal and the only answer set of $P_{I_a} \cup F(P)$ s.t. $\pi(A') = A$, an argument similar to the one in (β) of the proof for (i) can be applied: If another answer set A'' exists s.t. $\pi(A'') = A$, each $in_AS(x)$ s.t. $x \in A$ must be added in some stage of $T^i_{Q^{A'}}$, and in some stage $T^j_{Q^{A''}}$ of the standard least fixpoint operator. But then, by an inductive argument, $T^{\infty}_{Q^{A'}} = T^{\infty}_{Q^{A''}}$ must hold. □

The meta-interpreter program P_{I_a} has the benign property that a standard class of programs, namely the class of stratified programs, which are easy to evaluate, is also interpreted efficiently through it. Recall that a normal propositional program P is *stratified*, if there is a function λ which associates with each atom a in P an integer $\lambda(a) \geq 0$, such

that each rule $r \in P$ with $H(r) = \{h\}$ satisfies $\lambda(h) \geq \lambda(\ell)$ for each $\ell \in B^+(r)$ and $\lambda(h) > \lambda(\ell)$ for each $\ell \in B^-(r)$. Denote for any program P by P_{ir} the set of rules in $Ground(P_{I_a} \cup F(P))$ whose representation literals (i.e., literals over $rule$, $head$, pbl, nbl) are satisfied by $F(P)$. Then we have:

Proposition 1
Let P be a stratified normal propositional program. Then, P_{ir} is locally stratified (i.e., stratified if viewed as a propositional program).

This can be seen by constructing from a stratification mapping λ for P a suitable stratification mapping λ' for P_{ir}. Locally stratified programs are handled efficiently by competitive ASP systems.

4 Preferred Answer Sets

In this section, we first introduce the underpinnings common to all three semantics for preferred answer sets we are considering in this paper and then provide the individual definitions.

4.1 Prioritized Programs

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 (*prioritized program*)
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 (*full prioritization*)
A full prioritization of a prioritized program $\mathcal{P} = (P, <)$ is any $(P, <')$ 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 $\mathcal{FP}(\mathcal{P}) = \{\mathcal{P}\}$.

4.2 B-Preferred Answer Sets

B-preferred answer sets have been introduced in (Brewka & Eiter, 1999) as a refinement of previous approaches in (Brewka, 1994; Brewka, 1996). We first define B-preferred answer sets for a particular class of fully prioritized programs. A program $\mathcal{P}$ is *prerequisite-free*, if $B^+(r) = \emptyset$ for every rule $r \in \mathcal{P}$ holds. Furthermore, a literal ℓ (resp., a set X of literals) *defeats* a rule r of the form (1), if $\ell \in B^-(r)$ (resp., $X \cap B^-(r) \neq \emptyset$). A rule r' defeats a rule r if $H(r')$ defeats r.

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:

$$S_0 = \emptyset$$

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

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

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

An answer set A $(= S)$ divides the rules of P in Definition 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 $C_{\mathcal{P}}^B$, then the inductive construction guarantees that indeed all zombies are defeated by generating rules with higher preference.

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

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

Example 5
Reconsider the bird & penguin example. Let us first check whether $A_1 = \{penguin, bird, \neg flies\}$ is a B-preferred answer set. We determine the dual reduct ${}^{A_1}\mathcal{P}$ which consists of the following rules:

$$\begin{array}{ll} (1) & penguin. \\ (2) & bird. \\ (3) & \neg flies \ \texttt{:-} \ \texttt{not}\ flies. \\ (4) & flies \ \texttt{:-} \ \texttt{not}\ \neg flies. \end{array}$$

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

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

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

Example 6
Consider the following program:

$$\begin{array}{ll} (1) & c \ \texttt{:-} \ \texttt{not}\ b\,. \\ (2) & b \ \texttt{:-} \ \texttt{not}\ a\,. \end{array}$$

where $(1) < (2)$. Its single answer set is $A = \{b\}$. However, $C^B_{{}^{A}\mathcal{P}}(A) = \{c, b\}$ and thus A is not B-preferred. Indeed, no B-preferred answer set exists.

4.2.1 Adapting the Meta-Interpreter

Now we can extend the meta-interpreter for answer set programs from Section 3 to cover prioritized answer set programs.

Representing a prioritized program. A prioritized program $\mathcal{P} = (P, <)$ is represented by a set of facts $F(\mathcal{P})$ which contains $F(P)$ plus, for each rule preference $r < r'$ that belongs to the transitive reduction of $<$, a fact $pr(r, r')$.

Example 7
In the case of our bird & penguin example, we add the following three facts:

$$pr(r1, r2). \quad pr(r2, r3). \quad pr(r3, r4).$$

Checking preferredness. According to Definition 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, refining $<$:

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

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 $C^B_{\mathcal{P}'}$ from Definition 3 where $\mathcal{P}' = {}^X\mathcal{P}$. To this end, we do not compute the sets S_i as in Definition 3 – clearly one rule can contribute at most one element to $C^B_{\mathcal{P}'}$ 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 $C^B_{\mathcal{P}'}$; for later occurrences of duplicates $S_i = S_{i-1}$ will always hold.

In Definition 3, a condition is stated, which holds whenever $H(r_i)$ is not added, 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 δ *global defeat* (by the answer set).

Definition 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 only rules with a true positive body w.r.t. the answer set have to be considered. The encoding is then straightforward:

$$\begin{aligned}
lit(X,Y) \;&\texttt{:-}\; head(X,Y), pos_body_true(Y),\\
&\quad\; \texttt{not}\; defeat_local(Y), \texttt{not}\; in_AS(X).\\
lit(X,Y) \;&\texttt{:-}\; head(X,Y), pos_body_true(Y),\\
&\quad\; \texttt{not}\; defeat_local(Y), \texttt{not}\; defeat_global(Y).\\
defeat_local(Y) \;&\texttt{:-}\; nbl(X,Y), lit(X,Y1), pr(Y1,Y).\\
defeat_global(Y) \;&\texttt{:-}\; nbl(X,Y), in_AS(X).
\end{aligned}$$

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

$$in_CP(X) \;\texttt{:-}\; lit(X,Y).$$

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

$$\begin{aligned}
&\texttt{:-}\; in_CP(X), \texttt{not}\; in_AS(X).\\
&\texttt{:-}\; in_AS(X), \texttt{not}\; in_CP(X).
\end{aligned}$$

This completes the meta-interpreter program P_{I_B}. A compact listing of it (without showing P_{I_a} explicitly) is given in Figure 1. The following result states that it works correctly.

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

Proof

```
% 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), pos_body_true(Y),
               not defeat_local(Y), not in_AS(X).
   lit(X,Y) :- head(X,Y), pos_body_true(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).
   :- in_AS(X), not in_CP(X).  % this constraint is redundant
```

Fig. 1. Meta-Interpreter P_{I_B} for B-Preferred Answer Sets (without P_{I_a})

Let $Q = P_{I_B} \cup F(\mathcal{P})$ and $Q_a = P_{I_a} \cup F(\mathcal{P})$. By well-known results about splitting a logic program (Lifschitz & Turner, 1994), we obtain that for each answer set A of Q, its restriction A_a to the predicates of Q_a is an answer set of Q_a, and that A is an answer set of $(P_{I_B} \setminus Q_a) \cup A_a$.

(i) Suppose that $A \in \mathcal{AS}(Q)$. Then, A_a is an answer set of Q_a, and thus, by Theorem 1, $S = \pi(A_a)$ $(= \pi(A))$ is an answer set of P. Furthermore, by the three clauses that define and constrain the predicate pr in P_{I_B}, A defines a total ordering $<'$ on P such that $r <' r'$ is equivalent to $pr(r, r')$. Consider the dual reduct ${}^S\mathcal{P} = ({}^SP, {}^S{<'})$. Since S is an answer set of P, it remains to show that $C^B_{{}^S\mathcal{P}}(S) = S$ holds. Define the sets S_i, $0 \leq i \leq n$, along the ordering ${}^S{<'}$ as follows. Denote by $r_i^{{}^S\mathcal{P}} \in P$ the least rule r under $<'$ such that its dual reduct ${}^Sr$ (i.e., $H({}^Sr) = H(r)$ and $B({}^Sr) = B^-(r)$) is the i-th rule in ${}^S\mathcal{P}$ under ${}^S{<'}$, where $1 \leq i \leq n$. Then, let $S_0 = \emptyset$ and let $S_i = \{\ell \mid lit(\ell, r) \in A$, where $r = r_j^{{}^S\mathcal{P}}$ for some $j \leq i\}$, for all $i = 1, \ldots, n$.

By an inductive argument on $i = 0, \ldots, n$, we obtain that the sequence S_0, …, S_n satisfies the condition of Definition 3. Indeed, this is true for $i = 0$. Suppose it is true for $i - 1$ and consider i. Let $r = r_i^{{}^S\mathcal{P}}$ where $H(r) = \{h\}$. Then, $pos_body_true(r) \in A$ holds. If $h \in S_{i-1}$ holds, then the condition in Definition 3 anyway holds. Thus, suppose $h \notin S_{i-1}$ and first that $S_i = S_{i-1}$. Then, both rules with head $lit(h, r)$ in Q are not applied in A. This means that either (α) $defeat_local(r) \in A$ or (β) $\{in_AS(h), defeat_global(r)\} \subseteq A$ must hold. Case (α) implies that, by the induction hypothesis, there is some literal $\ell \in B^-(r)$ such that $\ell \in S_{i-1}$; that is, S_{i-1} defeats r. Case (β) means that $H(r) \subseteq S$ and that S defeats r. Thus, S_i satisfies Definition 3 in this case.

Otherwise, suppose we have $h \in S_i \setminus S_{i-1}$. Thus, $lit(h, r) \in A$, which means that one of the two rules with head $lit(h, r)$ in Q is applied in A. Thus, we have (α) $defeat_local(r) \notin A$, which, by the induction hypothesis, means that r is not defeated by S_{i-1}, and that (β) either $in_AS(h) \notin A$, hence $H(r) \not\subseteq S$, or $defeat_global(r) \notin A$, which, by the definition

of $defeat_global$, means that S does not defeat r. Thus, the condition of Definition 3 is satisfied. This proves the claim on the sequence $S_0, \ldots, S_n$.

As easily seen, $S_n = \{\ell \mid \exists r : lit(\ell, r) \in A\}$. Therefore, from the rule defining in_CP in P_{I_B}, we obtain that $in_CP(\ell) \in A$ holds iff $\ell \in S_n$, for any constant ℓ. Thus, from the last two constraints of P_{I_B}, we infer that $C^B_{sp}(S) = S$ must hold; in other words, $S = \pi(A)$ is a B-preferred answer set of $\mathcal{P}$.

(ii) Suppose that $A \in \mathcal{BPAS}(\mathcal{P})$, and let $\mathcal{P}' = (P, <')$ be a full prioritization of $\mathcal{P}$, such that $A \in \mathcal{BPAS}(\mathcal{P}')$. Then, we obtain an answer set A' of Q as follows:

- On the predicates defined in $P_{I_a} \cup F(\mathcal{P})$, A' coincides with the answer set of $P_{I_a} \cup F(P)$ corresponding to A as in item (ii) of Theorem 1;
- pr is defined according to $<'$, i.e., $pr(r, r')$ iff $r <' r'$;
- $lit(h, r)$ is true iff r is a rule from P such that r is applied in A;
- $defeat_local(r)$ is true iff some rule $r' <' r$ exists such that $H(r') \cap B^-(r) \neq \emptyset$ and r' is applied in A;
- $defeat_global(r)$ is true iff $A \cap B^-(r) \neq \emptyset$;
- $in_CP(\ell)$ is true iff $\ell \in A$.

Note that A' satisfies the last two constraints in P_{I_B} by virtue of Theorem 1, since A is an answer set of $\mathcal{P}$.

By the splitting result of (Lifschitz & Turner, 1994), for showing that A' is an answer set of Q, we only need to show the following. Let A'' be the restriction of A' to the predicates in $P_{I_a} \cup F(\mathcal{P})$ and pr; then, A' is an answer set of the program Q_1, which contains A'' plus all clauses c in Q which involve the predicates lit, $defeat_local$, $defeat_global$, in_CP and such that A'' satisfies all literals in $B(c)$ on the predicates in P_{I_a} and pr.

As easily seen, Q_1 is locally stratified, and a stratification λ exists on the atoms occurring in Q_1 such that

- $\lambda(lit(\ell', r')) < \lambda(lit(\ell, r))$ and $\lambda(defeat_local(r')) < \lambda(defeat_local(r))$, for all constants ℓ, ℓ' and rules $r, r' \in P$ such that $r' <' r$;
- $\lambda(defeat_local(r)) < \lambda(lit(\ell, r))$, for all literals ℓ and rules $r \in P$;
- $\lambda(in_CP(\ell)) = 1 + \max\{\lambda(lit(\ell, r)) \mid head(\ell, r) \in A''\}$; and
- $\lambda(a) = 0$, for all other atoms a in Q_1.

Then, along λ, we can verify that Q_1 has a stratified model S which coincides with A', i.e., for $i \geq 0$, we have for all atoms a with $\lambda(a) \leq i$ that $a \in A'$ iff $a \in S$.

For $i = 0$, this is immediate from the definition of A'. Suppose the statement is true for i, and consider $i + 1$. Let a be an atom such that $\lambda(a) = i + 1$. Suppose first that $a \in A'$, and consider the possible cases:

• If $a = defeat_local(r)$, then some $r' <' r$ exists such that $H(r') \cap B^-(r) \neq \emptyset$ and r' is applied in A. By definition, $lit(\ell, r') \in A'$ holds, and by the induction hypothesis, $lit(\ell, r') \in S$. Thus, $defeat_local(r) \in S$.

• Next, if $a = in_CP(\ell)$, then by definition of A', $\ell \in A$. Since A is a B-preferred answer set of $\mathcal{P}'$, it follows that $lit(\ell, r) \in A'$ for some r such that $\lambda(lit(\ell, r)) \leq i$. Thus, by the induction hypothesis, $lit(\ell, r) \in S$, which implies $in_CP(\ell) \in S$.

• Finally, if $a = lit(\ell, r)$, then by definition of A', r is applied in A. Thus, we have by construction $head(\ell, r) \in S$, $pos_body_true(r) \in S$, $defeat_global(r) \notin S$, and all these

atoms rank lower than $lit(\ell, r)$. Furthermore, $defeat_local(r) \notin S$ must hold; otherwise, as $\lambda(defeat_local(r)) < \lambda(lit(\ell, r))$, by the induction hypothesis, some rule $r' <' r$ would exist which is applied in A such that $H(r') \cap B^-(r) \neq \emptyset$, which would contradict that r is applied in A. This means, however, that the second rule with head $lit(\ell, r)$ in Q_1 is applied, and thus $lit(\ell, r) \in S$.

Thus, $a \in A'$ implies $a \in S$. Conversely, suppose that $a \in S$, and again consider the possible cases:

• If $a = defeat_local(r)$, then it follows that $lit(\ell, r') \in S$ for some $r' <' r$ such that $H(r') \cap B^-(r) \neq \emptyset$. By the induction hypothesis, $lit(\ell, r') \in A'$, which by definition of A' means that r' is applied in A; since $H(r') \cap B^-(r) \neq \emptyset$, by definition of A' we have $defeat_local(r) \in A'$.

• Next, if $a = in_CP(\ell)$, then $lit(\ell, r) \in S$ exists such that $\lambda(lit(\ell, r)) \leq i$. By the induction hypothesis and the definition of A', we have that r is applied in A. Therefore, $\ell \in A$, which by definition means $in_CP(\ell) \in A'$.

• Finally, if $a = lit(\ell, r)$, then $head(\ell, r) \in A'$, $pos_body_true(r) \in A'$, $defeat_local(r) \notin A'$, and either ($\alpha$) $in_AS(\ell) \notin A'$, or (β) $defeat_global(r) \notin A'$. In case ($\alpha$), by definition of A' we have $\ell \notin A$. This implies, however, that A is not a B-preferred answer set of $\mathcal{P}$, which is a contradiction. Thus, (β) must apply. By the induction hypothesis, we obtain $defeat_global(r) \notin S$. Hence, $B^-(r) \cap A = \emptyset$, which means that r is applied in A; hence, $lit(\ell, r) \in A'$ by definition.

This shows that $a \in S$ implies $a \in A'$, which concludes the induction. We thus have shown that $S = A'$. Since A' satisfies the last two constraints of P_{I_B}, it follows that S is the stratified model of Q_1 and A' is an answer set Q_1. Hence, A' is an answer set of Q. This proves the result. □

Example 8

For the bird & penguin example, $P_{I_B} \cup F(\mathcal{P})$ has one answer set, containing $in_AS(bird)$, $in_AS(penguin)$, and $in_AS(\neg flies)$.

We note that the last constraint in P_{I_B} is in fact redundant and can be dropped; this is possible since the fixpoint condition $C^B_{\mathcal{P}}(A) = A$ in the Definition 4 can be equivalently replaced by a weaker condition as follows.

Proposition 2

Let $\mathcal{P} = (P, <)$ be a fully prioritized and prerequisite-free program, and let $A \in \mathcal{AS}(\mathcal{P})$. Then, $A \in \mathcal{BPAS}(\mathcal{P})$ iff $C^B_{\mathcal{P}}(A) \subseteq A$.

Proof

It suffices to show that $C^B_{\mathcal{P}}(A) \subseteq A \wedge C^B_{\mathcal{P}}(A) \neq A$ raises a contradiction. Assume that the condition holds. Then, some $\ell \in A \setminus C^B_{\mathcal{P}}(A)$ must exist, which means that a generating rule r w.r.t. A must exist such that $H(r) = \{\ell\}$ and $A \models B(r)$. According to Definition 3, (α) $\vee$ (β) must hold for A_r, otherwise $\ell \in C^B_{\mathcal{P}}(A)$ would hold. Now (β) cannot hold, since A cannot defeat r because $A \models B(r)$. Thus (α) must hold. This implies that some $\ell' \in A_{r-1}$ defeats r such that $\ell' \notin A$. Since $A_{r-1} \subseteq C^B_{\mathcal{P}}(A)$, $C^B_{\mathcal{P}}(A) \not\subseteq A$ follows. This is a contradiction. □

4.2.2 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 has been 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 in 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 propositional 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 $A \in \mathcal{BPAS}(\mathcal{P}')$ if $A \in \mathcal{BPAS}(\mathcal{P})$; "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 invalidated by removing any other source r' before. 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 *stages* for modeling the iterations through Steps 2–5. Since the number of steps is bounded by the number of rules in $\mathcal{P}$, it is possible to avoid introducing new symbols for these stages and reuse the rule-IDs for this purpose instead:

$$stage(T) \quad :- \quad rule(T).$$

Similar as in Section 3.2, we employ the built-in total order lt on constants for ordering the stages. For example, if we have $rule(r1)$ and $rule(s1)$, then we have $stage(r1)$ and $stage(s1)$, and $r1$ is "earlier" than $s1$ since $lt(r1, s1)$ holds in DLV's built-in order lt. It is worthwhile noting that any total order on the stage symbols would work, and that this ordering is different from the ordering of the rules given in the input (which is partial in general). The first (least) stage 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 defined as follows (label "i" is not of interest and thus omitted):

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

% Use rules ids as stages.
  stage(S) :- rule(S).

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

% Other steps in the algorithm: Remove non-z nodes and, under some
% conditions, also z-nodes.
  nosource(R,S) :- pr(R1,R), stage(S), not remove(R1,S).
  remove(R,S1)  :- rule(R), not nosource(R,S), stage(S), stage(S1),
                   lt(S,S1), not z(R), not remove(R,S).
  remove(R,S1)  :- rule(R), not nosource(R,S), stage(S), stage(S1),
                   lt(S,S1), z(R), nbl(X,R), s(X,S).

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

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

Fig. 2. Meta-Interpreter P_{I_g} for B-Preferred Answer Sets Using Deterministic Preferredness Checking (without P_{I_a})

$$g(R) \;\texttt{:-}\; rule(R), pos_body_true(R), \texttt{not}\ neg_body_false(R).$$
$$z(R) \;\texttt{:-}\; rule(R), pos_body_true(R), head(X,R), \texttt{not}\ in_AS(X).$$

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 node in G, and a predicate $remove(R,S)$ which means that at stage S, the vertex R is no longer in G:

$$nosource0(R) \;\texttt{:-}\; pr(R1,R).$$
$$remove(R,S) \;\texttt{:-}\; rule(R), \texttt{not}\ nosource0(R), \texttt{not}\ z(R), stage(S).$$

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

$$\begin{aligned} nosource(R,S) \;\texttt{:-}\;& pr(R1,R), stage(S), \texttt{not}\ remove(R1,S) \\ remove(R,S1) \;\texttt{:-}\;& rule(R), \texttt{not}\ nosource(R,S), stage(S), stage(S1), lt(S,S1) \\ & \texttt{not}\ z(R), \texttt{not}\ remove(R,S). \\ remove(R,S1) \;\texttt{:-}\;& rule(R), \texttt{not}\ nosource(R,S), stage(S), stage(S1), lt(S,S1) \\ & z(R), nbl(X,R), s(X,S). \end{aligned}$$

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 $s(X,St)$, which informally means that X belongs to set S at stage St, and add the rule:

$$s(X,St) \;\texttt{:-}\; remove(R,St), g(R), head(X,R).$$

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:

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

The resulting meta-interpreter program P_{I_g} (without P_{I_a}) is shown in Figure 2. Note that P_{I_g} is in general also more efficient than P_{I_B}, 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 there), we can state the following result:

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

Proof (sketch)
(i) As in the proof for Theorem 2 we employ the notion of splitting a program (Lifschitz & Turner, 1994). Let $Q = P_{I_g} \cup F(\mathcal{P})$ and $Q_a = P_{I_a} \cup F(\mathcal{P})$. Then for each answer set A of Q, its restriction A_a to the predicates of Q_a is an answer set of Q_a and $A \in \mathcal{AS}((P_{I_B} \setminus Q_a) \cup A_a)$. By Theorem 1, $T = \pi(A_a) = \pi(A)$ is an answer set of P.

We can now loosely argue that the deterministic variant of FULL-ORDER with input T creates at most n (where $n = |P|$) intermediate values for the set S (not counting the initialization to $\emptyset$), referred to as $S_1, \ldots, S_n$, and implicitly creates (cumulative) sets $R_1, \ldots, R_n$ of removed rules. It can be seen that there is a one-to-one mapping of rule labels $r_1, \ldots, r_n$, ordered by the built-in total order lt, to $S_1, \ldots, S_n$ and $R_1, \ldots, R_n$ via $stage$. Now, $R_1 = \{r \mid remove(r, r_1) \in A\}$ and $S_1 = \{h \mid s(h, r_1) \in A\}$ by the definition of $nosource0$, $remove$ and s. We can proceed by induction and assume that $R_i = \{r \mid remove(r, r_i) \in A\}$ and $S_i = \{h \mid s(h, r_i) \in A\}$ for $1 \leq i < n$. Then, it can be seen that $R_{i+1} = \{r \mid remove(r, r_{i+1}) \in A\}$ and $S_{i+1} = \{h \mid s(h, r_{i+1}) \in A\}$ hold by definition of the predicates $nosource0$, $remove$ and s. Observe further that $R_n = \{r \mid removed(r) \in A\}$, and that the graph G is empty iff $R_n = P$. Since A satisfies the final constraint in P_{I_g}, $R_n = P$ is guaranteed to hold. Therefore the algorithm outputs "yes" and so $\pi(A) \in \mathcal{BPAS}(\mathcal{P})$ holds.
(ii) For $A \in \mathcal{BPAS}(\mathcal{P})$ we can construct an answer set A' of Q, such that (again by notion of splitting) A'' is the restriction of A' to the predicates defined by $Q_a = P_{I_a} \cup F(\mathcal{P})$ and A' is an answer set of the program Q_1, which contains A'' plus all clauses c in $Ground(Q)$ which involve the predicates g, z, $nosource0$, $remove$, s, $removed$ such that A'' satisfies all literals in $B(c)$ involving predicates defined in Q_a. Q_1 is locally stratified by a stratification defined as follows (where $r_1, \ldots, r_n$ are the rule identifiers ordered by the built-in total order lt):

- $\lambda(removed(r)) = 2 \times n + 1$
- $\lambda(nosource(r, r_i)) = 2 \times i + 1$
- $\lambda(remove(r, r_i)) = \lambda(s(h, r_i)) = 2 \times i$
- $\lambda(g(r)) = \lambda(z(r)) = 1$
- $\lambda(a) = 0$ for all other atoms a in Q_1

Since $\pi(A'') = \pi(A') = A$, it is easy to see that A' is an extension to A'' (which is an answer set of $P_{I_a} \cup F(\mathcal{P})$) that is fully determined by λ. A' must furthermore satisfy the final constraint in P_{I_g}. Due to these facts, such an A' can be effectively constructed. □

Example 9
Consider the program in Example 12 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 $s(c, ri)$ are 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.

An alternate definition of B-preferred answer sets is provided by (Schaub & Wang, 2001); a meta-interpreter program following that definition can be developed using techniques similar to the ones employed in P_{I_B}, P_{I_g}, and the interpreters in the following sections.

4.3 W-Preferred Answer Sets

The semantics we have seen in Section 4 is but one way to assign a meaning to prioritized logic programs. In this section we will introduce a related approach due to Wang, Zhou and Lin (Wang *et al.*, 2000) following the presentation in (Schaub & Wang, 2001).

Definition 7 (*W-preferred answer set, $\mathcal{WPAS}$*)
Let $\mathcal{P} = (\{r_1, \ldots, r_n\}, <)$ be a prioritized 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:

$$S_0 = \emptyset$$

$$S_i = S_{i-1} \cup \left\{ H(r) \;\middle|\; \begin{array}{ll} \text{I.} & r \in \mathcal{P} \text{ is active wrt. } (S_{i-1}, S), \text{ and} \\ \text{II.} & \text{there is no rule } r' \in \mathcal{P} \text{ with } r' < r \text{ such that} \\ & \text{(a) } r' \text{ is active wrt. } (S, S_{i-1}), \text{ and} \\ & \text{(b) } H(r') \notin S_{i-1}. \end{array} \right\}$$

for all $i = 1, \ldots, n$, where a rule r is *active* wrt. the pair (X, Y) if $B^+(r) \subseteq X$ and $B^-(r) \cap Y = \emptyset$.

The set $C_{\mathcal{P}}^W(S)$ is defined by

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

and S of $\mathcal{P}$ is W-preferred if $C_{\mathcal{P}}^W(S) = S$. The set of all W-preferred answer sets of $\mathcal{P}$ is denoted by $\mathcal{WPAS}(\mathcal{P})$. In this paper we will only consider consistent W-preferred answer sets.

```
% Part 1: Guess a consistent set S.

  lit(L) :- head(L,_).   lit(L) :- pbl(L,_).   lit(L) :- nbl(L,_).

  in_S(L) v notin_S(L) :- lit(L).
  :- compl(X,Y), in_S(X), in_S(Y).

% Part 2: Handle preferences.

  pr(X,Z) :- pr(X,Y), pr(Y,Z).
  :- pr(X,X).

% Part 3: Stage IDs.

  stage(S) :- rule(S).

% Part 4: Evaluate positive bodies.

  pos_body_false_S(R) :- rule(R), pbl(X,R), not in_S(X).
  pos_body_false_Si(R,Si) :- pbl(L,R), stage(Si), not in_Si(L,Si).
  pos_body_false_S0(R) :- pbl(L,R).

% Part 5: Evaluate negative bodies.

  neg_body_false_S(R) :- rule(R), nbl(X,R), in_S(X).
  neg_body_false_Si(R,Si) :- nbl(L,R), in_Si(L,Si).

% Part 6: Determine active rules.

  active(R,Si) :- rule(R), stage(Si),
                  not pos_body_false_Si(R,Si), not neg_body_false_S(R).

  active_Si(R,Si) :- rule(R), stage(Si), not pos_body_false_S(R),
                     not neg_body_false_Si(R,Si).
  active_S0(R) :- rule(R), not pos_body_false_S0(R), not neg_body_false_S(R).

% Part 7: Check for preferred generating rules.

  head_not_in_Si(R,Si) :- stage(Si), head(H,R), not in_Si(H,Si).

  preferred_generating_rule_exists(R,Si) :- pr(R1,R), active_Si(R1,Si),
                                            head_not_in_Si(R1,Si).
  preferred_generating_rule_exists_S0(R) :- pr(R1,R), not pos_body_false_S(R1).

% Part 8: Compute Si.

  in_Si(H,Si) :- head(H,R), active(R,Sj), stage(Sj), stage(Si), lt(Sj,Si),
                 not preferred_generating_rule_exists(R,Sj).
  in_Si(H,Si) :- head(H,R), active_S0(R), stage(Si),
                 not preferred_generating_rule_exists_S0(R).

% Part 9: Verify "stability".

  in_PAS(L) :- in_Si(L,_).
  :- in_PAS(L), not in_S(L).
  :- in_S(L), not in_PAS(L).
```

Fig. 3. Meta-Interpreter P_{1_W} for W-Preferred Answer Sets

4.3.1 Adapting the Meta-Interpreter

In Figure 3 we provide a meta-interpreter for W-preferred answer sets which closely follows Definition 7 and consists of three parts: The first guesses a consistent literal set A (Part 1 below), the second proceeds in stages of rule application according to the definition (Parts 2–8 below), and the final one verifies the "stability" condition $C_{\mathcal{P}}^{W}(A) = A$ (Part 9).

Part 1 [Guess a consistent set S] By means of the first three rules we extract all literals occurring in the input program $\mathcal{P}$ into a new predicate lit. Then we guess all possible subsets S of lit by means of the disjunctive rule such that $in_S(X)$ is true iff $X \in S$. The constraint, finally, ensures that the set S is consistent.

Part 2 [Handle Preferences] To complete the preference relation we transitively close the pr predicate and we also verify that it is irreflexive. The constraint is violated (in that case $P_{I_W} \cup F(\mathcal{P})$ admits no answer set) only if a rule is preferred to itself – $pr(X, X)$.

Part 3 [Stage IDs] Similar to Definition 7, in which we have used the indices $1, \ldots, n$ of the rules $r_1, \ldots, r_n$, we reuse the IDs of the rules in $\mathcal{P}$ and the arbitrary built-in total order lt over these IDs as IDs of the consecutive stages $S_1, \ldots, S_n$ of Definition 7. In this way, the rule IDs and the order lt effectively represent the sequence $S_1, \ldots, S_n$ without introducing new symbols.

Part 4 [Evaluate positive bodies] According to Definition 7 we need to evaluate the positive and negative bodies of the rules in $\mathcal{P}$ in two ways to verify whether a rule is active wrt. (S_i, S) and (S, S_i), respectively.

The predicates $pos_body_false_S(R)$ and $pos_body_false_Si(R, Si)$ represent the sets of all rules R whose bodies are false according to the set S we have guessed in Part 1 and the set S_i, respectively, where S_i is represented by the predicate in_Si (with $S_i = \{L \mid in_Si(L, Si)\}$). $pos_body_false_S0(R)$ covers the base case for $S_0 = \emptyset$ where the positive body is false w.r.t. S_0 if some positive body literal exists.

Part 5 [Evaluate negative bodies] Analogously to the case of positives bodies in Part 4, we define predicates which determine the falsity of negative rule bodies w.r.t. S and S_i. Note that no negative body can be false w.r.t. $S_0 = \emptyset$, so there is no special predicate for S_0.

Part 6 [Determine active rules] Now we need to define those rules that are active wrt. (S_i, S) and (S, Si). The former is handled by the first rule, the latter by the second and third rules. Again, the third rule covers the special case for the initial stage $S_0 = \emptyset$.

Part 7 [Check for preferred generating rules] The rule defining the predicate $preferred_generating_rule_exists(R, Si)$ checks whether a rule, which is preferred to R, exists such that it is active wrt. (S, S_i) and its head does not occur in S_i. Here we use $head_not_in_Si(R, Si)$ as an auxiliary predicate that checks whether the head of the rule R is in the set S_i. The third rule once more covers the base case $(S, S_0) = (S, \emptyset)$, where the body of the rule can be simplified.

Part 8 [Compute S_i] To compute S_i, we have to include the head of all rules which are active wrt. (S_{i-1}, S) and where no preferred rule exists which is active wrt. (S, S_{i-1}) and whose head does not already occur in S_{i-1}. Also here we need a specialized rule for the base-case where $S_{i-1} = S_0$.

Part 9 [Verify "stability"] Finally, we define a predicate in_PAS as the union of S_i, for $1 \leq i \leq n$, and check the "stability" condition of Definition 7, i.e., we check whether the relations in_S and in_PAS are equal. Any difference between these two will lead to a violation of one of the two constraints and thus a corresponding answer set for $P_{I_W} \cup F(\mathcal{P})$ cannot exist.

We provide the following theorem that states the correctness of the meta-interpreter program P_{I_W}, where $\pi'(S) = \{\ell \mid in_PAS(\ell) \in S\}$.

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

Proof
For the proof of the result, the same techniques as in the proofs of Theorems 2 and 3 can be used. A major difference is that P_{I_W} does not start from answer sets generated by P_{I_a}. Rather, the answer sets of the rules in Part 1 of P_{I_W} and $F(\mathcal{P})$, denoted by $P_1 \cup F(\mathcal{P})$, correspond one to one to the consistent sets S of literals occurring in P. Therefore the splitting is done on the literals $head$, pbl, nbl, $compl$, lit, in_S, $notin_S$.

As for (i), let $A_1^{\mathcal{P}}$ be an answer set of $P_1 \cup F(\mathcal{P})$ which corresponds to a consistent set of literals S of P. We show by induction on $j = 1, \ldots, n$ that the sets S_j of Definition 7 correspond to the sets $\{h \mid in_Si(h, r_j) \in A_{1'}^{\mathcal{P}}\}$ where $A_{1'}^{\mathcal{P}}$ is the (single) answer set of the rules in Parts 2–8 of P_{I_W} and $A_1^{\mathcal{P}}$, denoted by $P_2 \cup A_1^{\mathcal{P}}$. Recall that the r_i are the rule labels which identify the sets S_i: therefore, the S_i are totally ordered via the built-in total order lt ("less than") on the rule labels r_i by $lt(S_i, S_j)$ iff $lt(r_i, r_j)$. In what follows, we may assume that $lt(r_i, r_j)$ (equivalently, $lt(S_i, S_j)$) iff $i < j$, i.e., the ordering is given by indices.

To see the correspondence, it can be verified from inspection of the rules in Parts 4 and 5 that $active_S0(r)$ holds, by the third rule of Part 6, exactly for all rules r which are active w.r.t. $(\emptyset, S) = (S_0, S)$ and that $preferred_generating_rule_exists_S0(r)$ holds, by the third rule of Part 7, iff a rule r' exists such that $r' < r$ and r' is active w.r.t. $(S, \emptyset) = (S, S_0)$. Given that S_1 is the first element in the sequence $S_1, \ldots, S_n$, the first rule in Part 8 cannot contribute to generate $in_Si(h, r_1) \in A_{1'}^{\mathcal{P}}$. Thus, each such literal must be generated by application of the second rule in Part 8; since $H(r') \not\subseteq \emptyset$ trivially holds for any rule r', this means that the conditions I and II in Definition 7 must be satisfied; therefore $h \in S_1$ is equivalent to $in_Si(h, r_1) \in A_{1'}^{p}$, which shows the correspondence in the induction base.

Assume now that S_j corresponds to $\{h \mid in_Si(h, r_j) \in A_{1'}^{p}\}$. where $j < n$. Then, we verify that also S_{j+1} corresponds to $\{h \mid in_Si(h, r_{j+1}) \in A_{1'}^{p}\}$. First, we observe that for any $in_Si(h, r_{j'}) \in A_{1'}^{p}$, where $j' \leq j$, also $in_Si(h, r_{j+1}) \in A_{1'}^{p}$ holds because

of the construction of the rules in Part 8. For literals which do not occur in S_j but in S_{j+1}, observe that by the induction hypothesis, $active_Si(r, s_j)$ holds, via the first rule in Part 4 and the second rule in Part 5, exactly for rules r which are active w.r.t. (S, S_j), and thus $preferred_generating_rule_exists(r, s_j)$ holds, via the first and second rule of Part 7, iff a rule r' exists such that $r' < r$ and r' is active w.r.t. (S, S_j) and $H(r') \notin S_j$. Furthermore, $active(r, s_j)$ holds, via the second rule in Part 4 and the first rule in Part 5, exactly for rules r which are active w.r.t. (S_j, S). Since $in_Si(h, r_{j+1}) \in A^p_{1'}$ such that $in_Si(h, r_j) \notin A^p_{1'}$ must be derived by the first rule of Part 8, we thus have that $in_Si(h, r_{j+1}) \in A^{\mathcal{P}}_{1'}$ and $in_Si(h, r_j) \notin A^{\mathcal{P}}_{1'}$ iff $h \in S_{j+1} \setminus S_j$. Therefore, S_{j+1} corresponds to $\{h \mid in_Si(h, r_{j+1}) \in A^p_{1'}\}$, which completes the inductive argument.

Finally, concerning Part 9, in_PAS encodes S_n, i.e. $\{l \mid in_PAS(l) \in A^{\mathcal{P}}_{1'}\} = S_n$. The constraints guarantee that in any answer set the criterion $C^W_{\mathcal{P}}(S) = S_n = S$ is met (recall that inconsistent preferred answer sets are not considered in this paper). In total, we have shown that $A \in \mathcal{AS}(P_{I_W} \cup F(\mathcal{P}))$ implies $\pi'(A) \in \mathcal{WPAS}(\mathcal{P})$, which proves item (i).

As for (ii), let Q_1 be the split program obtained from the ground program by dropping rules from Part 1 and $F(\mathcal{P})$ and adding an answer set A'' of the dropped rules, while keeping only those rules of which the bodies agree with A''. Then, Q_1 is (analogous to the program in the proof of Theorem 2) locally stratified. A possible stratification for Q_1 would be as follows, where $r_1, r_2, \ldots, r_n$ are the rule IDs ordered by lt:

- $\lambda(in_PAS(l)) = 3 \times n + 2$
- $\lambda(preferred_generating_rule_exists(r, r_i)) = 3 \times i + 2$
- $\lambda(active(r, r_i)) = 3 \times i + 2$
- $\lambda(head_not_in_Si(r, r_i)) = 3 \times i + 1$
- $\lambda(active_Si(r, r_i)) = 3 \times i + 1$
- $\lambda(pos_body_false_Si(r, r_i)) = 3 \times i + 1$
- $\lambda(neg_body_false_Si(r, r_i)) = 3 \times i$
- $\lambda(in_Si(h, r_i)) = 3 \times i$
- $\lambda(preferred_generating_rule_exists_S0(r)) = 2$
- $\lambda(pos_body_false_S(r)) = 1$
- $\lambda(active_S0(r)) = 1$
- $\lambda(a) = 0$ for all other atoms a in Q_1

Based on this information, we can readily apply the rules of Q_1 level by level and effectively construct $A' \in \mathcal{AS}(P_{I_W} \cup F(\mathcal{P}))$ such that $\pi'(A') = A$. Specifically, we can verify that Q_1 has the answer set A' such that, for each $j \in \{1, \ldots, n\}$, $in_Si(h, S_j) \in A'$ holds iff $h \in S_j$ holds and thus $in_PAS(h) \in A'$ iff $h \in A$ holds, so $\pi'(A') = A$. □

4.4 D-Preferred Answer Sets

Another way to assign a meaning to prioritized logic programs has been introduced by Delgrande, Schaub and Tompits (Delgrande *et al.*, 2000a). For our presentation we again follow (Schaub & Wang, 2001).

Definition 8 (*D-preferred answer set, $\mathcal{DPAS}$*)

Let $\mathcal{P} = (\{r_1, \ldots, r_n\}, <)$ be a prioritized 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:

$$S_0 = \emptyset$$

$$S_i = S_{i-1} \cup \left\{ H(r) \;\middle|\; \begin{array}{ll} \text{I.} & r \in \mathcal{P} \text{ is active wrt. } (S_{i-1}, S), \text{ and} \\ \text{II.} & \text{there is no rule } r' \in \mathcal{P} \text{ with } r' < r \text{ such that} \\ & \text{(a) } r' \text{ is active wrt. } (S, S_{i-1}), \text{ and} \\ & \text{(b) } r' \notin rule(S_{i-1}). \end{array} \right\}$$

for all $i = 1, \ldots, n$, where again a rule r is *active* wrt. the pair (X, Y) if $B^+(r) \subseteq X$ and $B^-(r) \cap Y = \emptyset$, and $rule(X)$ denotes those rules in $\mathcal{P}$ that have been effectively used to derive literals in X.

The set $C^D_{\mathcal{P}}(S)$ is defined by

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

and an answer set A of $\mathcal{P}$ is D-preferred if $C^D_{\mathcal{P}}(A) = A$. The set of all D-preferred answer sets of $\mathcal{P}$ is denoted by $\mathcal{DPAS}(\mathcal{P})$.

The basic difference between D-preferred and W-preferred answer sets is that the former requires that a higher-ranked rule has been used to actually derive some literal, while for the latter it is sufficient that the literal appears in the head of such a rule.

In fact one can show that the three approaches we have shown get increasingly restrictive in that each approach admits only a subset of the (preferred) answer sets of the previous approach. The following theorem is due to (Schaub & Wang, 2001):

Theorem 5 (*Schaub & Wang, 2001*)
Let $\mathcal{P} = (P, <)$ be a propositional prioritized program. Then, we have:

$$\mathcal{DPAS}(\mathcal{P}) \subseteq \mathcal{WPAS}(\mathcal{P}) \subseteq \mathcal{BPAS}(\mathcal{P}) \subseteq \mathcal{AS}(\mathcal{P})$$

4.4.1 Adapting the Meta-Interpreter

The changes from P_{I_W} to P_{I_D} are relatively small, and we have marked those lines where P_{I_D} differs by a vertical bar in Figure 4.

Instead of tracking literals by means of in_Si we need to track which rule has been used to derive a particular literal, which is done by means of a predicate $in_rule_Si(H, R, Si)$ specifying that in the state denoted by Si the literal H has been derived by means of the rule R.

Similarly, we replace $head_not_in_Si$ by a new predicate $rule_not_generating_in_Si$ that tells whether a specific rule has been actually applied, not just whether the head of this rule has been derived (possibly from a different rule).

in_Si, finally, is a simple projection of in_rule_Si to obtain the union of all S_is for use in the stability check. We have the following result (recall $\pi'(S) = \{\ell \mid in_PAS(\ell) \in S\}$).

Theorem 6
Let $\mathcal{P} = (P, <)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{AS}(P_{I_D} \cup F(\mathcal{P}))$

```
% Part 1: Guess a consistent set S.

  lit(L) :- head(L,_).   lit(L) :- pbl(L,_).   lit(L) :- nbl(L,_).

  in_S(L) v notin_S(L) :- lit(L).
  :- compl(X,Y), in_S(X), in_S(Y).

% Part 2: Handle preferences.

  pr(X,Z) :- pr(X,Y), pr(Y,Z).
  :- pr(X,X).

% Part 3: Stage IDs.

  stage(S) :- rule(S).

% Part 4: Evaluate positive bodies.

  pos_body_false_S(R) :- rule(R), pbl(X,R), not in_S(X).
  pos_body_false_Si(R,Si) :- pbl(L,R), stage(Si), not in_Si(L,Si).
  pos_body_false_S0(R) :- pbl(L,R).

% Part 5: Evaluate negative bodies.

  neg_body_false_S(R) :- rule(R), nbl(X,R), in_S(X).
  neg_body_false_Si(R,Si) :- nbl(L,R), in_Si(L,Si).

% Part 6: Determine active rules.

  active(R,Si) :- rule(R), stage(Si),
                  not pos_body_false_Si(R,Si), not neg_body_false_S(R).

  active_Si(R,Si) :- rule(R), stage(Si), not pos_body_false_S(R),
                     not neg_body_false_Si(R,Si).
  active_S0(R) :- rule(R), not pos_body_false_S0(R), not neg_body_false_S(R).

% Part 7: Check for preferred generating rules.

| rule_not_generating_in_Si(R,Si) :- stage(Si), head(H,R),
|                                    not in_rule_Si(H,R,Si).

| preferred_generating_rule_exists(R,Si) :- pr(R1,R), active_Si(R1,Si),
|                                            rule_not_generating_in_Si(R1,Si).
  preferred_generating_rule_exists_S0(R) :- pr(R1,R), not pos_body_false_S(R1).

% Part 8: Compute Si.

| in_rule_Si(H,R,Si) :- head(H,R), active(R,Sj), stage(Sj), stage(Si),
|                       lt(Sj,Si), not preferred_generating_rule_exists(R,Sj).
| in_rule_Si(H,R,Si) :- head(H,R), active_S0(R), stage(Si),
|                       not preferred_generating_rule_exists_S0(R).

| in_Si(H,Si) :- in_rule_Si(H,_,Si).

% Part 9: Verify "stability".

  in_PAS(L) :- in_Si(L,_).
  :- in_PAS(L), not in_S(L).
  :- in_S(L), not in_PAS(L).
```

Fig. 4. Meta-Interpreter P_{I_D} for D-Preferred Answer Sets

then $\pi'(A) \in \mathcal{DPAS}(\mathcal{P})$, and (ii) for each $A \in \mathcal{DPAS}(\mathcal{P})$, there exists some $A' \in \mathcal{AS}(P_{I_D} \cup F(\mathcal{P}))$ such that $\pi'(A') = A$.

Proof
Given that both Definition 8 is very similar to Definition 7 (they differ only in II.b, where $H(r') \notin S_{i-1}$ is replaced by $r' \notin rule(S_{i-1})$, i.e., r' has not been used to derive some literal in S_{i-1}) and that also P_{I_W} is very similar to P_{I_D}, most of the arguments in the proof of Theorem 4 are also valid here.

We just highlight the essential difference: According to the difference between Definition 8 and Definition 7, a predicate $rule_not_generating_in_Si$ is used in P_{I_D} to store the rules which have not been applied to derive some literal in S_i, rather than $head_not_in_Si$ as in P_{I_W}, which is defined in the first rule and used in the second rule of Part 7.

For defining $rule_not_generating_in_Si$, an auxiliary predicate in_rule_Si is used (defined in Part 8), which is like in_Si but stores in addition the rule ID r which is used to derive a literal h in a set S_i. The predicate in_Si is then defined by a simple projection (third rule of Part 8). Note that the only point where $head_not_in_Si$ is used is in the definition of $rule_not_generating_in_Si$.

In the inductive step, the argument (in the proof of Theorem 4) that the predicate $preferred_generating_rule_exists(r, r_i)$ holds iff a rule r' exists such that $r' < r$ and r' is active w.r.t. (S, S_i) and $H(r') \notin S_i$, must accordingly be updated such that it reads "$preferred_generating_rule_exists(r, r_i)$ holds iff a rule r' exists such that $r' < r$ and r' is active w.r.t. (S, S_i) and $r' \notin rule(S_i)$."

Apart from these details, the proof remains the same. □

5 Weakly Preferred Answer Sets

The concept of a 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. So far, this approximation has only been defined for B-preferred answer sets (Brewka & Eiter, 1999), though similar extensions to the two other approaches we have seen in Sections 4.3 and 4.4 should be feasible.

Definition 9 (*distance*)
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 $m' <_2 m$.[3]

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.

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

Definition 10 (*preference violation degree, pvd*)
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.,

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

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 11 (*weakly preferred answer set, $w\mathcal{PAS}$*)
Let $\mathcal{P} = (P, <)$ be a prioritized program. Then, $A \in \mathcal{AS}(P)$ is a weakly preferred answer set of $\mathcal{P}$ iff $pvd_{\mathcal{P}}(A) = pvd(\mathcal{P})$. By $w\mathcal{PAS}(\mathcal{P})$ we denote the collection of all such weakly preferred answer sets of $\mathcal{P}$.

Example 10
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 11
Reconsider the program in Example 6. Its answer set $A = \{b\}$ is not preferred. Switching the priorities of the two rules, the resulting prioritized program $\mathcal{P}'$ has $C^B_{A\mathcal{P}'}(A) = \{b\}$, 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 12
Consider the following program $\mathcal{P}$:

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

$\mathcal{P}$ 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 defeasible 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 $\mathcal{P}$.

5.1 Adapting the Meta-Interpreter for Weakly Preferred Answer Sets

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 does not need to be compatible with the input partial order, and corresponds to $<_2$ in Definition 10.

$$\begin{array}{rcl} pr1(X,Y) \vee pr1(Y,X) & :- & rule(X),\ rule(Y),\ X != Y. \\ pr1(X,Z) & :- & pr1(X,Y),\ pr1(Y,Z). \\ & :- & pr1(X,X). \end{array}$$

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:

$$defeat_local(Y) \quad :- \quad nbl(X,Y),\ lit(X,Y1),\ pr1(Y1,Y).$$

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 leveling concept here).

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

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. 9, 10, and 11.

Let us call the resulting interpreter $P_{I_{\text{weak}}}$; a compact listing (without showing P_{I_a}) is given in Figure 5. We have the following result:

Theorem 7
Let $\mathcal{P} = (P, <)$ be a propositional prioritized program. Then, (i) if $A \in \mathcal{OAS}(P_{I_{\text{weak}}} \cup F(\mathcal{P}))$, then $\pi(A) \in w\mathcal{PAS}(\mathcal{P})$, and (ii) for each $A \in w\mathcal{PAS}(\mathcal{P})$, there exists some $A' \in \mathcal{OAS}(P_{I_{\text{weak}}} \cup F(\mathcal{P}))$ such that $\pi(A') = A$.

Proof
Let Q_0 be the set of all clauses in $P_{I_{\text{weak}}}$ except the two rules defining pr, the constraint $:\!-pr(X,X).$, and the weak constraint for minimization of violations. After renaming $pr1$ to pr, Q_0 is identical to the meta-interpreter program P_{I_B} minus the redundant constraint $:\!-in_AS(X), \texttt{not}\ in_CP(X)$. Thus, we infer from Theorem 2 that $\mathcal{AS}(Q_0 \cup F((P,\emptyset)))$ is in correspondence (i), (ii) as there to $\mathcal{BPAS}((P,\emptyset))$. Let Q_1 result from Q_0 by adding the rules defining pr and the constraint $:\!-pr(X,X).$ Then, $\mathcal{AS}(Q_1 \cup F(\mathcal{P}))$ is in a similar correspondence to the set of tuples $T = \{(A, <_1, <_2) \mid (P, <_1) \in \mathcal{FP}(\mathcal{P}), A \in \mathcal{BPAS}(P, <_2), (P, <_2) \in \mathcal{FP}((P,\emptyset))\}$. Adding the weak constraint to Q_1 (which results in $P_{I_{\text{weak}}}$), we thus have that $\mathcal{OAS}(P_{I_{\text{weak}}} \cup F(\mathcal{P}))$ is in similar correspondence to the set $T' = \{(A, <_1, <_2) \in T \mid \forall (A', <'_1, <'_2) \in T : d(<'_1, <'_2) \geq d(<_1, <_2)\}$, which in turn naturally corresponds to $w\mathcal{PAS}(\mathcal{P})$. More precisely, we can conclude that for each $A \in \mathcal{OAS}(P_{I_{\text{weak}}} \cup F(\mathcal{P}))$, there exists some tuple $(\pi(A), <_1, <_2)$ in T', which corresponds to $\pi(A) \in w\mathcal{PAS}(\mathcal{P})$; conversely, for each $A \in w\mathcal{PAS}(\mathcal{P})$, there exists some tuple $(A, <_1, <_2) \in T'$, which corresponds to some $A' \in \mathcal{OAS}(P_{I_{\text{weak}}} \cup F(\mathcal{P}))$ such that $\pi(A') = A$. This proves items (i) and (ii) of the theorem. □

```
% 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), pos_body_true(Y),
               not defeat_local(Y), not in_AS(X).
   lit(X,Y) :- head(X,Y), pos_body_true(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).
```

Fig. 5. Meta-Interpreter $P_{I_{\text{weak}}}$ for Weakly Preferred Answer Sets (without P_{I_a})

Example 13
Reconsider Example 6, which has no preferred answer set. $P_{I_{\text{weak}}} \cup F(\mathcal{P})$ 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 Example 11.

Example 14
Reconsider Example 12, which does not have any preferred answer set either. $P_{I_{\text{weak}}} \cup F(\mathcal{P})$ has one optimal answer set (having 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 Example 12.

While $P_{I_{\text{weak}}}$ is a straightforward encoding of the definition of weakly preferred answer set, and gives us an executable specification, it is quite inefficient on larger problem instances because of the large search space generated by the possible total orderings pr and $pr1$. To improve efficiency, we can use a variant of the graph algorithm FULL-ORDER from Section 4.2.2, based on the following observation. We may arrange the vertices which are removed from G, in this order, as a common prefix for orderings $<_1$ and $<_2$ in the definition of $pvd(A)$. We thus need to guess only optimal completions of $<_1$ and $<_2$ for those rules that remain in G on termination of FULL-ORDER. In particular, if G is empty, then $<_1$ and $<_2$ coincide and A is a preferred answer set, hence also weakly preferred. The meta-interpreter programs P_{I_g} and $P_{I_{\text{weak}}}$ can be combined to another meta-interpreter

program for computing weakly preferred answer sets, which conservatively extends the computation of preferred answer sets in the sense that guessing comes only into play if no preferred answer sets exist. However, we do not further discuss this issue here.

6 Related Work

Meta-interpretation of answer sets or answer set-like semantics has been considered by other authors as well, in different contexts. We briefly discuss (Gelfond & Son, 1997; Marek & Remmel, 2001; Delgrande *et al.*, 2000a; Delgrande *et al.*, 2001) which are related to our work.

Gelfond and Son. In (Gelfond & Son, 1997), the idea of meta-interpretation was used to define the semantics of a language $\mathcal{L}_0$ for rules with preferences. $\mathcal{L}_0$ is a multi-sorted logical language which has constants for individuals, definite rules, and default rules 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 formula

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

represents a default rule, where d is its name and $[l_1, \ldots, l_m]$ is Prolog-like list notation. Informally, it amounts to the rule $l_0 \texttt{:-} l_1, \ldots, l_m, \texttt{not}\ \neg l_0$ in extended logic programming. Moreover, the language allows to express conflicts between two default rules; both preferences and conflicts can be declared dynamically by means of rules.

The semantics of $\mathcal{L}_0$ is then defined in terms of a transformation of any program P in $\mathcal{L}_0$ into an extended logic program $t(P)$ whose answer sets are, roughly speaking, cast into answer sets of the program P.

However, there are some salient differences w.r.t. the approach of (Gelfond & Son, 1997) and the one presented here.

- First and foremost, the semantics of $\mathcal{L}_0$ is defined exclusively by means of a meta-interpreter, while our approach implements semantics which have been defined previously without meta-interpretation techniques.
- Secondly, 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, a traversal mechanism that exploits an implicit ordering, and default negation.
- Thirdly, in our approach we extend a general answer set 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.

Marek and Remmel. In a recent paper (Marek & Remmel, 2001), Marek and Remmel discussed the issue of a meta-interpreter for propositional normal logic programs in the context of the expressiveness of stable logic programming. They describe a function-free normal logic program $Meta1$, such that on input of a factual representation edb_Q of a 0-2

normal logic program Q (i.e., each clause in Q has 0 or 2 positive body literals), a projection of the answer sets of $Meta1 \cup edb_Q$ is in one-to-one correspondence to the answer sets of Q. The representation edb_Q is similar to our representation $F(Q)$, but explicitly records the position of positive body literals. The meta-interpreter $Meta1$ is similar to ours, but differs from ours in the following respects:

- Firstly, $Meta1$ encodes a simple guess and check strategy for the computation of a stable model S. It contains a pair of unstratified rules which guess for each atom a, whether a is in the stable model S or not. The remaining clauses mimic the computation of the minimal model of Q^S, using a special predicate $computed$, and constraints check whether S is reconstructed by it. On the other hand, our meta-interpreter P_{I_a} has no separate guessing and checking parts. Instead, stability of a model is effected by the stable semantics underlying the interpreter. Furthermore, P_{I_a} uses negation sensible to the structure of the program Q, and in essence preserves stratification (cf. Proposition 1).
- Secondly, because of its naive guess and check strategy, $Meta1$ is highly inefficient for programs Q which can be evaluated easily. In particular, even for positive programs Q, $Meta1$ explores an exponential search space, and computation of the unique stable model of Q may take considerable time. On the other hand, for stratified Q our meta-interpreter program P_{I_a} is, after propagation of the input facts, a locally stratified program and can be evaluated efficiently. Loosely speaking, P_{I_a} interprets a significant class of computational "easy" logic programs efficiently.
- Thirdly, $Meta1$ is only applicable for 0-2 normal logic programs. An extension to arbitrary normal logic programs is possible using similar techniques as in this paper, though.

All these considerations suggest the conclusion that the meta-interpreter $Meta1$ defined in (Marek & Remmel, 2001) is more of theoretical interest, which is fully compliant with the goals of that paper.

Delgrande, Schaub, and Tompits. Based on a seminal approach for adding priorities to default logic (Delgrande & Schaub, 1997), Delgrande *et al.* have developed the PLP framework for expressing priorities on logic programs (Delgrande *et al.*, 2000a; Delgrande *et al.*, 2001). In this framework, extended logic programs with preference information $r \prec r'$ between rules attached at the object level, are "compiled" into another extended logic program, such that the answer sets of the latter program amount to the preferred answer sets of the original program. The transformation uses a number of control predicates for the application of rules such that rule preferences are respected as intended in the application of the rules for constructing an answer set of a given program. More specifically, control atoms $ap(r)$ and $bl(r)$ state whether a rule r is applied or blocked, respectively, and atoms $ok(r)$ and $rdy(r)$ control the applicability of rules based on antecedent conditions reflecting the given order information. The framework provides the flexibility to modify the standard transformation, such that transformations for different preference semantics can be designed.

The PLP framework significantly differs from our work in the following respects:

- Firstly, PLP does not use a fixed meta-interpreter for evaluating varying programs,

given in a format which can be "processed" by a meta-interpreter. Rather, PLP performs a schematic program construction ad-hoc.

- Secondly, PLP aims at a tool for realizing preferences semantics by providing a suite of special predicates and a particular representation formalism. In contrast, our interest is in casting definitions from first principles to extended logic programs, in a way such that we obtain executable specifications. This way, variations of definitions can be experimented with more flexibly.
- Thirdly, similar as (Gelfond & Son, 1997), PLP has no obvious separation of answer sets and prioritization.

7 Conclusion

In this paper, we have considered the issue of building experimental prototypes for semantics of extended logic programs equipped with rule preferences, by using the technique of meta-interpretation. In the course of this, we have presented a suite of meta-interpreters for various such semantics, including a simple meta-interpreter for answer set semantics of plain extended logic programs itself. This meta-interpreter has benign computational properties, and can be used as a building block for meta-interpreters of other semantics.

While the focus of this paper has been on a propositional setting, it is possible to extend the techniques that we have presented for handling non-ground programs as well. However, unless function symbols are allowed at the code level (which is currently not the case in `DLV`), a technical realization is not completely straightforward. Extending our work to deal with such cases, for which the work reported in (Bonatti, 2001b; Bonatti, 2001a) might prove useful, and creating a `DLV` front-end for prioritized program evaluation are issues for further work.

We believe that the work that we have presented in this paper provides supportive evidence to the following items.

- Meta-interpretation can be a useful technique for building experimental prototype implementations of knowledge representation formalisms. In particular, we have shown this for some preferences formalisms extending the seminal answer set semantics.
- By the use of answer set programming, it is possible to cast definitions of the semantics of KR-formalisms quite naturally and appealingly into extended logic programs, which then, by usage of answer set programming engines, provide executable specifications. Note that, in this line, You *et al.* (You *et al.*, 1999) have shown how inheritance networks can be compiled to logic programs, and that, on the other hand, semantics of logic programs may be used for refining the semantics of inheritance networks.
- Adding optimization constructs to the basic language of extended logic programming, such as weak constraints in `DLV` and the constructs provided in Smodels (Niemelä *et al.*, 2000), is valuable for elegantly expressing semantics which are defined in terms of optimal values of cost functions. The semantics of weakly preferred answer sets provides a striking example; other examples can be found e.g. in the domain of diagnostic reasoning. Enhancing ASP engines by further constructs and

their efficient realization is thus important for increasing the usability of the ASP compilation and meta-interpreter approach.

Furthermore, the techniques and methods, which we have used in the design of the meta-interpreters, in particular the use of ordering relations, may prove useful for other researchers when designing ASP implementations of applications.

In conclusion, it appears that meta-interpretation, which is well-established in Prolog-style logic programming, is also a topic of interest in ASP, whose exploration also provides useful results for core ASP itself. We are confident that future work will provide further evidence for this view.

Acknowledgments.

The authors would like to thank their colleagues, the participants of the AAAI 2001 Spring Symposium on Answer Set Programming, and the anonymous reviewers for their comments and helpful suggestions for improvements. This work was supported by the Austrian Science Fund (FWF) under grants P13871-INF, P14781-INF, and Z29-INF and the European Commission under projects IST-2001-37004 WASP and IST-2002-33570 INFOMIX.

References

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

Bonatti, Piero A. (2001a). Prototypes for Reasoning with Infinite Stable Models and Function Symbols. *Pages 416–419 of:* Eiter, Thomas, Faber, Wolfgang, & Truszczyński, Mirosław (eds), *Proc. 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*. LNCS, no. 2173. Springer.

Bonatti, Piero A. (2001b). Reasoning with Infinite Stable Models. *Pages 603–610 of: Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*. Seattle, WA, USA: Morgan Kaufmann Publishers.

Brewka, G. (1994). Adding Priorities and Specificity to Default Logic. *Pages 247–260 of: Proceedings JELIA '94*. LNAI 838. Springer.

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

Brewka, G., & Eiter, T. (2000). Prioritizing Default Logic. *Pages 27–45 of:* Hölldobler, Steffen (ed), *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*. Kluwer. ISBN 0-7923-6261-6.

Brewka, Gerhard, & Eiter, Thomas. (1999). Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, **109**(1-2), 297–356.

Buccafurri, F., Leone, N., & Rullo, P. (1999). Semantics and Expressiveness of Disjunctive Ordered Logic. *Annals of Mathematics and Artificial Intelligence*, **25**(3-4), 311–337. Abstract in Proceedings KR '98.

Buccafurri, Francesco, Leone, Nicola, & Rullo, Pasquale. (1997). Strong and Weak Constraints in Disjunctive Datalog. *Pages 2–17 of:* Dix, Jürgen, Furbach, Ulrich, & Nerode, Anil (eds), *Proc. 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*. Lecture Notes in AI (LNAI), no. 1265. Dagstuhl, Germany: Springer Verlag.

Delgrande, J., & Schaub, T. (1997). Compiling Reasoning With and About Preferences into Default Logic. *Pages 168–174 of: Proceedings IJCAI '97*.

Delgrande, J., & Schaub, T. (2000). Expressing Preferences in Default Logic. *Artificial Intelligence*, **123**(1-2), 41–87.

Delgrande, Jim, Schaub, Torsten, & Tompits, Hans. (2000a). Logic Programs with Compiled Preferences. *Pages 392–398 of:* Horn, Werner (ed), *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI'2000)*. IOS Press.

Delgrande, Jim, Schaub, Torsten, & Tompits, Hans. (2000b). Prioritized Default Logic Revisited: A Compilation of Brewka and Eiter's Approach. Ojeda-Aciego, M., de Guzmán, I. P., Brewka, G., & Moniz Pereira, L. (eds), *Proceedings European Workshop on Logics in Artificial Intelligence – Journées Européennes sur la Logique en Intelligence Artificielle (JELIA 2000), Malaga, Spain, September 29–October 2, 2000*. LNCS, no. 1919. Springer.

Delgrande, Jim, Schaub, Torsten, & Tompits, Hans. (2001). plp: A Generic Compiler for Ordered Logic Programs. *Pages 411–415 of:* Eiter, Thomas, Faber, Wolfgang, & Truszczyński, Mirosław (eds), *Proc. 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*. LNCS, no. 2173. Springer.

Eiter, Thomas, Faber, Wolfgang, Leone, Nicola, Pfeifer, Gerald, & Polleres, Axel. 2002 (Sept.). Answer Set Planning under Action Costs. *Pages 186–197 of:* Flesca, Sergio, Greco, Sergio, Ianni, Giovambattista, & Leone, Nicola (eds), *Proc. 8th European Conference on Artificial Intelligence (JELIA)*. Lecture Notes in Computer Science, no. 2424.

Faber, Wolfgang, & Pfeifer, Gerald. (since 1996). `DLV` *homepage*. `http://www.dlvsystem.com/`.

Faber, Wolfgang, Leone, Nicola, & Pfeifer, Gerald. (1999). Pushing Goal Derivation in DLP Computations. *Pages 177–191 of:* Gelfond, Michael, Leone, Nicola, & Pfeifer, Gerald (eds), *Proc. 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*. Lecture Notes in AI (LNAI), no. 1730. El Paso, Texas, USA: Springer Verlag.

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

Gelfond, Michael, & Son, Tran Cao. (1997). Reasoning with Prioritized Defaults. *Pages 164–223 of: Proc. Workshop of Logic Programming and Knowledge Representation (LPKR '97)*. Springer.

Lifschitz, V., & Turner, H. (1994). Splitting a Logic Program. *Pages 23–37 of:* Van Hentenryck, Pascal (ed), *Proc. 11th International Conference on Logic Programming (ICLP'94)*. Santa Margherita Ligure, Italy: MIT Press.

Lifschitz, Vladimir. (1996). Foundations of Logic Programming. *Pages 69–127 of:* Brewka, G. (ed), *Principles of Knowledge Representation*. Stanford: CSLI Publications.

Marek, V.W., & Remmel, J.B. (2001). On the Expressibility of Stable Logic Programming. *Pages 107–120 of:* Eiter, Thomas, Faber, Wolfgang, & Truszczyński, Mirosław (eds), *Proc. 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*. LNCS, no. 2173. Springer.

Marek, V.W., & Truszczyński, M. (1993). *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer-Verlag.

Niemelä, Ilkka, Simons, Patrik, & Syrjänen, Tommi. 2000 (April). Smodels: A System for Answer Set Programming. Baral, Chitta, & Truszczyński, Mirosław (eds), *Proc. 8th International Workshop on Non-Monotonic Reasoning (NMR'2000)*.

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

Sakama, Chiaki, & Inoue, Katsumi. (2000). Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence*, **123**(1-2), 185–222.

Schaub, Torsten, & Wang, Kewen. (2001). A Comparative Study of Logic Programs with Preference. *Pages 597–602 of: Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*. Seattle, WA, USA: Morgan Kaufmann Publishers.

Wang, Kewen, Zhou, Lizhu, & Lin, Fangzhen. (2000). Alternating Fixpoint Theory for Logic Pro-

grams with Priority. *Pages 164–178 of: Computational Logic - CL 2000, First International Conference, Proceedings*. Lecture Notes in AI (LNAI), no. 1861. London, UK: Springer Verlag.

You, Jia-Huai, Wang, Xianchang, , & Yuan, Li Yan. (1999). Compiling Defeasible Inheritance Networks to General Logic Programs. *Artificial Intelligence*, **113**(1-2), 247–268.

Zhang, Yan, & Foo, Norman Y. (1997). Answer Sets for Prioritized Logic Programs. *Pages 69–83 of: Proceedings ILPS 97*.