Manifold Answer-Set Programs and Their Applications*

Wolfgang Faber[1] and Stefan Woltran[2]

[1] University of Calabria, Italy
wf@wfaber.com
[2] Vienna University of Technology, Austria
woltran@dbai.tuwien.ac.at

Abstract. In answer-set programming (ASP), the main focus usually is on computing answer sets which correspond to solutions to the problem represented by a logic program. Simple reasoning over answer sets is sometimes supported by ASP systems (usually in the form of computing brave or cautious consequences), but slightly more involved reasoning problems require external postprocessing. Generally speaking, it is often desirable to use (a subset of) brave or cautious consequences of a program P_1 as input to another program P_2 in order to provide the desired solutions to the problem to be solved. In practice, the evaluation of the program P_1 currently has to be decoupled from the evaluation of P_2 using an intermediate step which collects the desired consequences of P_1 and provides them as input to P_2. In this work, we present a novel method for representing such a procedure within a *single* program, and thus within the realm of ASP itself. Our technique relies on rewriting P_1 into a so-called *manifold program*, which allows for accessing all desired consequences of P_1 within a single answer set. Then, this manifold program can be evaluated jointly with P_2 avoiding any intermediate computation step. For determining the consequences within the manifold program we use *weak constraints*, which is strongly motivated by complexity considerations. As applications, we present encodings for computing the ideal extension of an abstract argumentation framework and for computing world views of a particular class of epistemic specifications.

1 Introduction

In the last decade, *Answer Set Programming* (ASP) [1, 2], also known as A-Prolog [3, 4], has emerged as a declarative programming paradigm. ASP is well suited for modelling and solving problems which involve common-sense reasoning, and has been fruitfully applied to a wide variety of applications including diagnosis (e.g. [5]), data integration (e.g. [6]), configuration (e.g. [7]), and many others. Moreover, the efficiency of the latest tools for processing ASP programs (so-called ASP solvers) reached a state that makes them applicable for problems of practical importance [8]. The basic idea

* This work was supported by the Vienna Science and Technology Fund (WWTF), grant ICT08-028, and by M.I.U.R. within the Italia-Austria internazionalization project "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione" and the PRIN project LoDeN. A preliminary version of this paper appeared in the Proceedings of the the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009), pp. 115–128, Springer LNAI 5753, 2009.

of ASP is to compute answer sets of a logic program from which the solutions of the problem encoded by the program can be obtained.

However, frequently one is interested not only in the solutions per se, but rather in reasoning tasks that have to take some or even all solutions into account. As an example, consider the problem of database repair, in which a given database instance does not satisfy some of the constraints imposed in the database. One can attempt to modify the data in order to obtain a consistent database by changing as little as possible. This will in general yield multiple possibilities and can be encoded conveniently using ASP (see, e.g., [9]). However, usually one is not interested in the repairs themselves, but in the data which is present in *all* repairs. For the ASP encoding, this means that one is interested in the elements which occur in all answer sets; these are also known as *cautious consequences*. Indeed, ASP systems provide special interfaces for computing cautious consequences by means of query answering. But sometimes one has to do more, such as answering a complex query over the cautious consequences (not to be confused with complex queries over answer sets). So far, ASP solvers provide no support for such tasks. Instead, computations like this have to be done outside ASP systems, which hampers usability and limits the potential of ASP.

In this work, we tackle this limitation by providing a technique, which transforms an ASP program P into a *manifold program* M_P which we use to identify all consequences of a certain type (we consider here the well-known concepts of brave and cautious consequence, but also definite consequence [10]) within a *single* answer set. The main advantage of the manifold approach is that the resulting program can be extended by additional rules representing a query over the brave (or cautious, definite) consequences of the original program P, thereby using ASP itself for this additional reasoning. In order to identify the consequences, we use *weak constraints* [11], which are supported by the ASP-solver DLV [12]. Weak constraints have been introduced to prefer a certain subset of answer sets via penalization. Their use for computing consequences is justified by a complexity-theoretic argument: One can show that computing consequences is complete for the complexity classes $\mathrm{FP}_{||}^{\mathrm{NP}}$ or $\mathrm{FP}_{||}^{\Sigma_2^P}$ (depending on the presence of disjunction), for which also computing answer sets for programs with weak constraints is complete[3], which means that an equivalent compact ASP program without these extra constructs does not exist, unless the polynomial hierarchy collapses. In principle, other preferential constructs similar to weak constraints could be used as well for our purposes, as long as they meet these complexity requirements.

We discuss three particular applications of the manifold approach. First, we specify an encoding which decides the SAT-related *unique minimal model problem*, which is closely related to closed-world reasoning [13]. The second problem stems from the area of argumentation (cf. [14] for an overview) and concerns the computation of the ideal extension [15] of an argumentation framework. For both problems we make use of manifold programs of well-known encodings (computing all models of a CNF-formula

[3] The first of these results is fairly easy to see, for the second, it was shown [11] that the related decision problem is complete for the class Θ_2^P or Θ_3^P, from which the $\mathrm{FP}_{||}^{\mathrm{NP}}$ and $\mathrm{FP}_{||}^{\Sigma_2^P}$ results can be obtained. Also note that frequently cited NP, Σ_2^P, and co-NP, Π_2^P completeness results hold for brave and cautious query answering, respectively, but not for computing brave and cautious consequences.

for the former application, computing all admissible extensions of an argumentation framework for the latter) in order to compute consequences. Extensions by a few more rules then directly provide the desired solutions, requiring little effort in total. As a final application, we consider an encoding for (a certain subclass of) epistemic specifications as introduced by Gelfond [16]. In a nutshell, these specifications are extensions of ASP programs, which may include modal atoms to allow for reasoning over answer-sets within the object language, and thus are closely related to some of the ideas we present here. Epistemic specifications (already introduced in 1991 [17]) have received increasing interest only over the last years (see, e.g. [18–21]) but nowadays get more and more recognized as an highly expressive and important extension of standard answer-set programming.

Organization and Main Results. After introducing the necessary background in the next section, we

- introduce in Section 3 the concept of a manifold program for rewriting propositional programs in such a way that all brave (resp. cautious, definite) consequences of the original program are collected into a single answer set;
- lift the results to the non-ground case (Section 4); and
- present applications for our technique in Section 5. In particular, we provide ASP encodings for computing the ideal extension of an argumentation framework and for computing world views of a particular class of epistemic specifications.

The paper concludes with a brief discussion of related and further work.

2 Preliminaries

In this section, we review the basic syntax and semantics of ASP with weak constraints, following [12], to which we refer for a more detailed definition.

An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $\alpha(p) = n \geq 0$ and each t_i is either a variable or a constant. A *literal*[4] is either an atom a or its negation $\text{not } a$.

A *(disjunctive) rule* r is of the form

$$a_1 \vee \cdots \vee a_n \text{ :- } b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, and where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms.

The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$, and the *body* of r is the set $B(r) = \{b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m\}$. Furthermore, $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. We will sometimes denote a rule r as $H(r)$:- $B(r)$.

A *weak constraint* [11] is an expression wc of the form

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

[4] For keeping the framework simple, we do not consider strong negation in this paper. However, the formalism can easily be adapted to deal with them.

where $m \geq k \geq 0$ and $b_1, \ldots, b_m$ are literals, while $weight(wc) = w$ (the *weight*) and l (the *level*) are positive integer constants or variables. For convenience, w and/or l may be omitted and are set to 1 in this case. The sets $B(wc)$, $B^+(wc)$, and $B^-(wc)$ are defined as for rules. We will sometimes denote a weak constraint wc as $:\sim B(wc)$.

A *program* P is a finite set of rules and weak constraints. We will often use semicolons for separating rules and weak constraints in order to avoid ambiguities. With $Rules(P)$ we denote the set of rules in P and $WC(P)$ denotes the set of weak constraints in P. w^P_{max} and l^P_{max} denote the maximum weight and maximum level over $WC(P)$, respectively. A program (rule, atom) is *propositional* or *ground* if it does not contain variables. A program is called *strong* if $WC(P) = \emptyset$, and *weak* otherwise.

For any program P, let U_P be the set of all constants appearing in P (if no constant appears in P, an arbitrary constant is added to U_P); let B_P be the set of all ground literals constructible from the predicate symbols appearing in P and the constants of U_P; and let $Ground(P)$ be the set of rules and weak constraints obtained by applying, to each rule and weak constraint in P all possible substitutions from the variables in P to elements of U_P. U_P is usually called the *Herbrand Universe* of P and B_P the *Herbrand Base* of P.

A ground rule r is *satisfied* by a set I of ground atoms iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program P, if each $r \in P$ is satisfied by I. For non-ground P, I satisfies P iff I satisfies $Rules(Ground(P))$. A ground weak constraint wc is *violated* by I, iff $B^+(wc) \subseteq I$ and $B^-(wc) \cap I = \emptyset$; it is satisfied otherwise.

Following [22], a set $I \subseteq B_P$ of atoms is an *answer set* for a strong program P iff it is a subset-minimal set that satisfies the *reduct*

$$P^I = \{H(r) \text{:- } B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Ground(P)\}.$$

A set of atoms $I \subseteq B_P$ is an *answer set* for a weak program P iff I is an answer set of $Rules(P)$ and $H^{Ground(P)}(I)$ is minimal among all the answer sets of $Rules(P)$, where the penalization function $H^P(I)$ for weak constraint violation of a ground program P is defined as follows:

$$\begin{array}{l} H^P(I) = \sum_{i=1}^{l^P_{max}} \left(f_P(i) \cdot \sum_{w \in N^P_i(I)} weight(w) \right) \\ f_P(1) \ = 1, \ \text{and} \\ f_P(n) \ = f_P(n-1) \cdot |WC(P)| \cdot w^P_{max} + 1 \ \text{ for } n > 1. \end{array}$$

where $N^P_i(I)$ denotes the set of weak constraints of P in level i violated by I. For any program P, we denote the set of its answer sets by $AS(P)$. In this paper, we use only weak constraints with weight and level 1, for which $H^{Ground(P)}(I)$ amounts to the number of weak constraints violated in I.

A ground atom a is a *brave* (sometimes also called credulous or possible) consequence of a program P, denoted $P \models_b a$, if $a \in A$ holds for at least one $A \in AS(P)$. A ground atom a is a *cautious* (sometimes also called skeptical or certain) consequence of a program P, denoted $P \models_c a$, if $a \in A$ holds for all $A \in AS(P)$. A ground atom a is a *definite* consequence [10] of a program P, denoted $P \models_d a$, if a is a cautious consequence of P and $AS(P) \neq \emptyset$. The sets of all brave, cautious, definite consequences of a program P are denoted as $BC(P)$, $CC(P)$, $DC(P)$, respectively.

3 Propositional Manifold Programs

In this section, we present a translation which essentially creates a copy of a given strong propositional program for each of (resp. for a subset of) its atoms. Thus, we require several copies of the alphabet used by the given program.

Definition 1. *Given a set I of literals, a collection $\mathcal{I}$ of sets of literals, and an atom a, define $I^a = \{p^a \mid \text{atom } p \in I\} \cup \{\text{not } p^a \mid \text{not } p \in I\}$ and $\mathcal{I}^a = \{I^a \mid I \in \mathcal{I}\}$.*

The actual transformation to a manifold is given in the next definition. We copy a given program P for each atom a in a given set S, whereby the transformation guarantees the existence of an answer set by enabling the copies conditionally.

Definition 2. *For a strong propositional program P and $S \subseteq B_P$, define its* manifold *(w.r.t. S) as*

$$P^{tr}_S = \bigcup_{r \in P} \{H(r)^a \text{ :- } \{c\} \cup B(r)^a \mid a \in S\} \cup \{c \text{ :- not } i \ ; \ i \text{ :- not } c\}.$$

We assume $B_P \cap B_{P^{tr}_S} = \emptyset$, that is, all symbols in P^{tr}_S are assumed to be fresh.

Example 1. Consider

$$\Phi = \{p \vee q \text{ :- } \ ; \ r \text{ :- } p \ ; \ r \text{ :- } q\}$$

for which we have $AS(\Phi) = \{\{p, r\}, \{q, r\}\}$, and thus $BC(\Phi) = \{p, q, r\}$ and $CC(\Phi) = DC(\Phi) = \{r\}$. When forming the manifold for $B_\Phi = \{p, q, r\}$, we obtain

$$\Phi^{tr}_{B_\Phi} = \left\{ \begin{array}{l} p^p \vee q^p \text{ :- } c \ ; \ r^p \text{ :- } c, p^p \ ; \ r^p \text{ :- } c, q^p \ ; \ c \text{ :- not } i \ ; \\ p^q \vee q^q \text{ :- } c \ ; \ r^q \text{ :- } c, p^q \ ; \ r^q \text{ :- } c, q^q \ ; \ i \text{ :- not } c \ ; \\ p^r \vee q^r \text{ :- } c \ ; \ r^r \text{ :- } c, p^r \ ; \ r^r \text{ :- } c, q^r \end{array} \right\}$$

Note that given a strong program P and $S \subseteq B_P$, the construction of P^{tr}_S can be done in polynomial time (w.r.t. the size of P). The answer sets of the transformed program consist of all combinations (of size $|S|$) of answer sets of the original program (augmented by c) plus the special answer set $\{i\}$ which we shall use to indicate inconsistency of P.

Proposition 1. *For a strong propositional program P and a set $S \subseteq B_P$, $AS(P^{tr}_S) = A \cup \{\{i\}\}$, where*

$$A = \{\bigcup_{i=1}^{|S|} A_i \cup \{c\} \mid \langle A_1, \ldots, A_{|S|} \rangle \in \prod_{a \in S} AS(P)^a\}.$$

Note that $\prod$ denotes the Cartesian product in Proposition 1.

Example 2. For Φ of Example 1, we obtain that $AS(\Phi^{tr}_{B_\Phi})$ consists of $\{i\}$ plus (copies of $\{q, r\}$ are underlined for readability)

$$\begin{array}{l} \{c, p^p, r^p, p^q, r^q, p^r, r^r\}, \{c, \underline{q^p, r^p}, \underline{q^q, r^q}, \underline{q^r, r^r}\}, \\ \{c, \underline{q^p, r^p}, p^q, r^q, p^r, r^r\}, \{c, p^p, r^p, \underline{q^q, r^q}, p^r, r^r\}, \{c, p^p, r^p, p^q, r^q, \underline{q^r, r^r}\}, \\ \{c, \underline{q^p, r^p}, \underline{q^q, r^q}, p^r, r^r\}, \{c, \underline{q^p, r^p}, p^q, r^q, \underline{q^r, r^r}\}, \{c, p^p, r^p, \underline{q^q, r^q}, \underline{q^r, r^r}\}. \end{array}$$

Using this transformation, each answer set encodes an association of an atom with some answer set of the original program. If an atom a is a brave consequence of the original program, then a witnessing answer set exists, which contains the atom a^a. The idea is now to prefer those atom-answer set associations where the answer set is a witness. We do this by means of weak constraints and penalize each association where the atom is not in the associated answer set, that is, where a^a is not in the answer set of the transformed program. Doing this for each atom means that an optimal answer set will not contain a^a only if there is no answer set of the original program that contains a, so each a^a contained in an optimal answer set is a brave consequence of the original program.

Definition 3. *Given a strong propositional program P and $S \subseteq B_P$, let*

$$P_S^{bc} = P_S^{tr} \cup \{:\sim \text{not } a^a \mid a \in S\} \cup \{:\sim i\}$$

Observe that all weak constraints are violated in the special answer set $\{i\}$, while in the answer set $\{c\}$ (which occurs if the original program has an empty answer set) all but $:\sim i$ are violated.

Proposition 2. *Given a strong propositional program P and $S \subseteq B_P$, for any $A \in AS(P_S^{bc})$, $\{a \mid a^a \in A\} = BC(P) \cap S$.*

This result would also hold without including $:\sim\ i$ in P_S^{bc}. It has been included for clarity and for making the encoding more uniform with respect to the encoding for definite consequences, which will be presented below.

Example 3. For the program Φ as given Example 1,

$$\Phi_{B_\Phi}^{bc} = \Phi_{B_\Phi}^{tr} \cup \{:\sim \text{not } p^p \ ;\ :\sim \text{not } q^q \ ;\ :\sim \text{not } r^r \ ;\ :\sim i\}.$$

We obtain that $AS(\Phi_{B_\Phi}^{bc}) = \{A_1, A_2\}$, where

$$A_1 = \{c, p^p, r^p, q^q, r^q, p^r, r^r\};$$
$$A_2 = \{c, p^p, r^p, q^q, r^q, q^r, r^r\},$$

as these two answer sets are the only ones that violate no weak constraint. We can observe that $\{a \mid a^a \in A_1\} = \{a \mid a^a \in A_2\} = \{p, q, r\} = BC(\Phi)$.

Concerning cautious consequences, we first observe that if a program is inconsistent (in the sense that it does not have any answer set), each atom is a cautious consequence. But if P is inconsistent, then P_S^{tr} will have only $\{i\}$ as an answer set, so we will need to find a suitable modification in order to deal with this in the correct way. In fact, we can use a similar approach as for brave consequences, but penalize those associations where an atom is contained in its associated answer set. Any optimal answer set will thus contain a^a for an atom only if a is contained in each answer set. If an answer set containing i exists, it is augmented by all atoms a^a, which also causes all weak constraints to be violated.

Definition 4. *Given a strong propositional program P and $S \subseteq B_P$, let*

$$P_S^{cc} = P_S^{tr} \cup \{ :\sim a^a \mid a \in S\} \cup \{a^a \text{ :- } i \mid a \in S\} \cup \{ :\sim i\}$$

Proposition 3. *Given a strong propositional program P and $S \subseteq B_P$, for any $A \in AS(P_S^{cc})$, $\{a \mid a^a \in A\} = CC(P) \cap S$.*

Similar to P_S^{bc}, this result also holds without including $:\sim i$.

Example 4. Recall program Φ from Example 1. We have

$$\Phi_{B_\Phi}^{cc} = \Phi_{B_\Phi}^{tr} \cup \{ :\sim p^p \ ; \ :\sim q^q \ ; \ :\sim r^r \ ; \ p^p \text{ :- } i \ ; \ q^q \text{ :- } i \ ; \ r^r \text{ :- } i \ ; \ :\sim i\}.$$

We obtain that $AS(\Phi_{B_\Phi}^{cc}) = \{A_3, A_4\}$, where

$$A_3 = \{c, q^p, r^p, p^q, r^q, p^r, r^r\};$$
$$A_4 = \{c, q^p, r^p, p^q, r^q, q^r, r^r\},$$

as these two answer sets are the only ones that violate only one weak constraint, namely $:\sim r^r$. We observe that $\{a \mid a^a \in A_3\} = \{a \mid a^a \in A_4\} = \{r\} = CC(\Phi)$.

We next consider the notion of definite consequences. Different to cautious consequences, we do not add the annotated atoms to the answer set containing i. However, this answer set should never be among the optimal ones unless it is the only one. Therefore we inflate it by new atoms i^a, all of which incur a penalty. This guarantees that this answer set will incur a higher penalty ($|B_P| + 1$) than any other ($\leq |B_P|$).

Definition 5. *Given a strong propositional program P and $S \subseteq B_P$, let*

$$P_S^{dc} = P_S^{tr} \cup \{ :\sim a^a;\ i^a \text{ :- } i;\ :\sim i^a \mid a \in S\} \cup \{ :\sim i\}$$

Proposition 4. *Given a strong propositional program P and $S \subseteq B_P$, for any $A \in AS(P_S^{dc})$, $\{a \mid a^a \in A\} = DC(P) \cap S$.*

Example 5. Recall program Φ from Example 1. We have

$$\Phi_{B_\Phi}^{dc} = \Phi_{B_\Phi}^{tr} \cup \{ :\sim p^p \ ; \ :\sim q^q \ ; \ :\sim r^r \ ; \ i^p \text{ :- } i \ ; \ i^q \text{ :- } i \ ; \ i^r \text{ :- } i \ ; \ :\sim i^p \ ; \ :\sim i^q \ ; \ :\sim i^r \ ; \ :\sim i\}.$$

As in Example 4, A_3 and A_4 are the only ones that violate only one weak constraint, namely $:\sim r^r$, and thus are the answer sets of $\Phi_{B_\Phi}^{dc}$.

Obviously, one can compute all brave, cautious, or definite consequences of a program by choosing $S = B_P$. We also note that the programs from Definitions 3, 4 and 5 yield multiple answer sets. However each of these yields the same atoms a^a, so it is sufficient to compute one of these. The programs could be extended in order to admit only one answer set by suitably penalizing all atoms a^b ($a \neq b$). To avoid interference with the weak constraints already used, these additional weak constraints would have to pertain to a different level.

4 Non-Ground Manifold Programs

We now generalize the techniques introduced in Section 3 to non-ground strong programs. The first step in Section 3 was to define the notion of annotation. There, we annotated propositional atoms with propositional atoms. Also in the non-ground case, we want to annotate atoms with atoms in some way, but it is not immediately clear what kind of atoms should be used for annotations — ground atoms or non-ground atoms?

The first thought would be to annotate using ground atoms, since after all the goal is to produce a copy of the program for each possible ground consequence. This would amount mean annotating each predicate (and thus also each atom) with ground atoms of some subset of the Herbrand Base. For example, annotating the rule $p(X,Y)$:- $q(X,Y)$ with the set $\{r(a), r(b)\}$ would yield the annotated rules $p^{r(a)}(X,Y)$:- $q^{r(a)}(X,Y)$ and $p^{r(b)}(X,Y)$:- $q^{r(b)}(X,Y)$. The tacit assumption here is that $r(a)$ and $r(b)$ are the only two ground instances of predicate r which are of interest.

Since we want to keep our description as general as possible, we assume annotation using the full Herbrand Base. In this scenario it makes sense to annotate with non-ground atoms, in order to ease readability and reduce the size of the (non-ground) manifold program. In particular, the arguments of these non-ground atoms should be mutually different variables, in order to represent all possible ground instances of the atom. The idea is that we can then use the standard grounding definition also on the annotations.

In the example given earlier, we would annotate using $r(Z)$. In order to be able to fall back on the regular grounding defined for non-annotated programs, we will annotate using only the predicate r and extend the arguments of p, yielding the rule $\mathrm{d}_p^r(X,Y,Z)$:- $\mathrm{d}_q^r(X,Y,Z)$ (we use predicate symbols d_p^r and d_q^r rather than p^r and q^r just for pointing out the difference between annotation by predicates versus annotation by ground atoms).

This notation is quite general, as it can restrict the annotations to ground atoms of special interest by adding appropriate atoms to the rule body. In our example, this amounts to writing $p^r(X,Y,Z)$:- $q^r(X,Y,Z), rdom(Z)$ where the predicate $rdom$ identifies the instances of r for which annotations should be produced. In the following, recall that $\alpha(p)$ denotes the arity of a predicate p.

Definition 6. *Given an atom $a = p(t_1, \ldots, t_n)$ and a predicate q, let a_q^{tr} be the atom $\mathrm{d}_p^q(t_1, \ldots, t_n, X_1, \ldots, X_{\alpha(q)})$ where $X_1, \ldots, X_{\alpha(q)}$ are fresh variables and d_p^q is a new predicate symbol with $\alpha(\mathrm{d}_p^q) = \alpha(p) + \alpha(q)$. Furthermore, given a set $\mathcal{L}$ of literals, and a predicate q, let $\mathcal{L}_q^{tr}$ be $\{a_q^{tr} \mid \text{atom } a \in \mathcal{L}\} \cup \{\mathrm{not}\ a_q^{tr} \mid \mathrm{not}\ a \in \mathcal{L}\}$.*

Note that we assume that even though the variables $X_1, \ldots, X_{\alpha(q)}$ are fresh, they will be the same for each occurrence of a_q^{tr}. We define the manifold program in analogy to Definition 2, the only difference being the different way of annotating.

Definition 7. *Given a strong program P and a set S of predicates, define its* manifold *as*

$$P_S^{tr} = \bigcup_{r \in P} \{H(r)_q^{tr} \text{ :- } \{c\} \cup B(r)_q^{tr} \mid q \in S\} \cup \{c \text{ :- } \mathrm{not}\ i \ ;\ i \text{ :- } \mathrm{not}\ c\}.$$

Example 6. Consider program

$$\Psi = \{p(X) \vee q(X) \text{ :- } r(X) \ ; \ r(a) \text{ :- } \ ; \ r(b) \text{ :- } \}$$

for which

$$\begin{aligned} AS(\Psi) = \{ & \{p(a), p(b), r(a), r(b)\}, \\ & \{p(a), q(b), r(a), r(b)\}, \\ & \{q(a), p(b), r(a), r(b)\}, \\ & \{q(a), q(b), r(a), r(b)\}\}. \end{aligned}$$

Hence, we have $BC(\Psi) = \{p(a), p(b), q(a), q(b), r(a), r(b)\}$ and moreover $CC(\Psi) = DC(\Psi) = \{r(a), r(b)\}$. Forming the manifold for $S = \{p\}$, we obtain

$$\Psi_S^{tr} = \left\{ \begin{array}{l} \mathrm{d}_p^p(X, X_1) \vee \mathrm{d}_q^p(X, X_1) \text{ :- } \mathrm{d}_r^p(X, X_1), c \ ; \\ \mathrm{d}_r^p(a, X_1) \text{ :- } c \ ; \ \mathrm{d}_r^p(b, X_1) \text{ :- } c \ ; \ c \text{ :- not } i \ ; \ i \text{ :- not } c \end{array} \right\}$$

$AS(\Psi_S^{tr})$ consists of $\{i\}$ plus 16 answer sets, corresponding to all combinations of the 4 answer sets in $AS(\Psi)$.

Now we are able to generalize the encodings for brave, cautious, and definite consequences. These definitions are direct extensions of Definitions 3, 4, and 5, the differences are only due to the non-ground annotations. In particular, the diagonalization atoms a^a should now be written as $\mathrm{d}_p^p(X_1, \ldots, X_{\alpha(p)}, X_1, \ldots, X_{\alpha(p)})$ which represent the set of ground instances of $p(X_1, \ldots, X_{\alpha(p)})$, each annotated by itself. So, a weak constraint $:\sim \ \mathrm{d}_p^p(X_1, \ldots, X_{\alpha(p)}, X_1, \ldots, X_{\alpha(p)})$ gives rise to $\{:\sim \ \mathrm{d}_p^p(c_1, \ldots, c_{\alpha(p)}, c_1, \ldots, c_{\alpha(p)}) \mid c_1, \ldots, c_{\alpha(p)} \in U\}$ where U is the Herbrand base of the program in question, that is one weak constraint for each ground instance annotated by itself.

Definition 8. *Given a strong program P and a set S of predicate symbols, let*

$$\begin{aligned} P_S^{bc} &= P_S^{tr} \cup \{:\sim \text{not } \Delta_q \mid q \in S\} \cup \{:\sim i\} \\ P_S^{cc} &= P_S^{tr} \cup \{:\sim \Delta_q \ ; \ \Delta_q \text{ :- } i \mid q \in S\} \cup \{:\sim i\} \\ P_S^{dc} &= P_S^{tr} \cup \{:\sim \Delta_q \ ; \ I_q \text{ :- } i \ ; \ :\sim I_q \mid q \in S\} \cup \{:\sim i\} \end{aligned}$$

where $\Delta_q = \mathrm{d}_q^q(X_1, \ldots, X_{\alpha(q)}, X_1, \ldots, X_{\alpha(q)})$ and $I_q = i_q(X_1, \ldots, X_{\alpha(q)})$.

Proposition 5. *Given a strong program P and a set S of predicates, for an arbitrary $A \in AS(P_S^{bc})$, (resp., $A \in AS(P_S^{cc})$, $A \in AS(P_S^{dc})$), the set $\{p(c_1, \ldots, c_{\alpha(p)}) \mid \mathrm{d}_p^p(c_1, \ldots, c_{\alpha(p)}, c_1, \ldots, c_{\alpha(p)}) \in A\}$ is the set of brave (resp., cautious, definite) consequences of P with a predicate in S.*

Example 7. Consider again Ψ and $S = \{p\}$ from Example 6. We obtain

$$\Psi_S^{bc} = \Psi_S^{tr} \cup \{:\sim \text{not } \mathrm{d}_p^p(X_1, X_1) \ ; \ :\sim i\}$$

and we can check that $AS(\Psi_S^{bc})$ consists of the sets

$$R \cup \{\mathrm{d}_p^p(a,a), \mathrm{d}_p^p(b,b), \mathrm{d}_q^p(a,b), \mathrm{d}_q^p(b,a)\},$$
$$R \cup \{\mathrm{d}_p^p(a,a), \mathrm{d}_p^p(b,b), \mathrm{d}_p^p(a,b), \mathrm{d}_q^p(b,a)\},$$
$$R \cup \{\mathrm{d}_p^p(a,a), \mathrm{d}_p^p(b,b), \mathrm{d}_q^p(a,b), \mathrm{d}_p^p(b,a)\},$$
$$R \cup \{\mathrm{d}_p^p(a,a), \mathrm{d}_p^p(b,b), \mathrm{d}_p^p(b,a), \mathrm{d}_p^p(b,a)\};$$

where $R = \{\mathrm{d}_r^p(a,a), \mathrm{d}_r^p(a,b), \mathrm{d}_r^p(b,a), \mathrm{d}_r^p(b,b)\}$. For each A of these answer sets we obtain $\{p(t) \mid \mathrm{d}_p^p(t,t) \in A\} = \{p(a), p(b)\}$ which corresponds exactly to the brave consequences of Ψ with a predicate of $S = \{p\}$.

For cautious consequences, we have

$$\Psi_S^{cc} = \Psi_S^{tr} \cup \{:\sim \mathrm{d}_p^p(X_1, X_1) \ ; \ \mathrm{d}_p^p(X_1, X_1) \text{:-} \ i \ ; \ :\sim i\}$$

and we can check that $AS(\Psi_S^{cc})$ consists of the sets

$$R \cup \{\mathrm{d}_q^p(a,a), \mathrm{d}_q^p(b,b), \mathrm{d}_q^p(a,b), \mathrm{d}_q^p(b,a)\},$$
$$R \cup \{\mathrm{d}_q^p(a,a), \mathrm{d}_q^p(b,b), \mathrm{d}_p^p(a,b), \mathrm{d}_q^p(b,a)\},$$
$$R \cup \{\mathrm{d}_q^p(a,a), \mathrm{d}_q^p(b,b), \mathrm{d}_q^p(a,b), \mathrm{d}_p^p(b,a)\},$$
$$R \cup \{\mathrm{d}_q^p(a,a), \mathrm{d}_q^p(b,b), \mathrm{d}_p^p(b,a), \mathrm{d}_p^p(b,a)\};$$

where $R = \{\mathrm{d}_r^p(a,a), \mathrm{d}_r^p(a,b), \mathrm{d}_r^p(b,a), \mathrm{d}_r^p(b,b)\}$, as above. For each A of these answer sets we obtain $\{p(t) \mid \mathrm{d}_p^p(t,t) \in A\} = \emptyset$ and indeed there are no cautious consequences of Ψ with a predicate of $S = \{p\}$.

Finally, for definite consequences,

$$\Psi_S^{dc} = \Psi_S^{tr} \cup \{:\sim \mathrm{d}_p^p(X_1, X_1) \ ; \ i_p(X_1) \text{:-} \ i \ ; \ :\sim i_p(X_1) \ ; \ :\sim i\}.$$

It is easy to see that $AS(\Psi_S^{dc}) = AS(\Psi_S^{cc})$ and so $\{p(t) \mid \mathrm{d}_p^p(t,t) \in A\} = \emptyset$ for each answer set A of Ψ_S^{dc}, and indeed there is also no definite consequence of Ψ with a predicate of $S = \{p\}$.

These definitions exploit the fact that the semantics of non-ground programs is defined via their grounding with respect to their Herbrand Universe. So the fresh variables introduced in the manifold will give rise to one copy of a rule for each ground atom.

In practice, ASP systems usually require rules to be safe, that is, that each variable occurs (also) in the positive body. The manifold for a set of predicates may therefore contain unsafe rules (because of the fresh variables). But this can be repaired by adding a *domain atom* $dom_q(X_1, \ldots, X_m)$ to a rule which is to be annotated with q. This predicate can in turn be defined by a rule $dom_q(X_1, \ldots, X_m)$:- $u(X_1), \ldots, u(X_m)$ where u is defined using $\{u(c) \mid c \in U_P\}$. One can also provide smarter definitions for dom_q by using a relaxation of the definition for q.

We also observe that ground atoms that are contained in all answer sets of a program need not be annotated in the manifold. Note that these are essentially the cautious consequences of a program and therefore determining all of those automatically before rewriting does not make sense. But for some atoms this property can be determined

by a simple analysis of the structure of the program. For instance, facts will be in all answer sets. In the sequel we will not annotate extensional atoms (those defined only by facts) in order to obtain more concise programs. One could also go further and omit the annotation of atoms which are defined using non-disjunctive stratified programs.

As an example, we present an ASP encoding for boolean satisfiability and then create its manifold program for resolving the following problem: Given a propositional formula in CNF φ, compute all atoms which are true in all models of φ. We provide a fixed program which takes a representation of φ as facts as input. To apply our method we first require a program whose answer sets are in a one-to-one correspondence to the models of φ. To start with, we fix the representation of CNFs. Let φ (over atoms A) be of the form $\bigwedge_{i=1}^{n} c_i$. Then, $D_\varphi = \{\mathrm{at}(a) \mid a \in A\} \cup \{\mathrm{cl}(i) \mid 1 \leq i \leq n\} \cup \{\mathrm{pos}(a,i) \mid \text{atom } a \text{ occurs positively in } c_i\} \cup \{\mathrm{neg}(a,i) \mid \text{atom } a \text{ occurs negatively in } c_i\}$. We construct program SAT as the set of the following rules.

$$\begin{aligned} t(X) &\text{ :- } \mathrm{not}\ f(X), \mathrm{at}(X); & f(X) &\text{ :- } \mathrm{not}\ t(X), \mathrm{at}(X);\\ ok(C) &\text{ :- } t(X), \mathrm{pos}(C,X); & ok(C) &\text{ :- } f(X), \mathrm{neg}(C,X);\\ &\text{ :- } \mathrm{not}\ ok(C), \mathrm{cl}(C). \end{aligned}$$

It can be checked that the answer sets of $\mathrm{SAT} \cup D_\varphi$ are in a one-to-one correspondence to the models (over A) of φ. In particular, for any model $I \subseteq A$ of φ there exists an answer set M of $\mathrm{SAT} \cup D_\varphi$ such that $I = \{a \mid t(a) \in M\}$. We now consider $\mathrm{SAT}^{cc}_{\{t\}}$ which consists of the following rules.

$$\begin{aligned} \mathrm{d}^t_t(X,Y) &\text{ :- } c, \mathrm{not}\ \mathrm{d}^t_f(X,Y), \mathrm{at}(X); & \mathrm{d}^t_f(X,Y) &\text{ :- } c, \mathrm{not}\ \mathrm{d}^t_t(X,Y), \mathrm{at}(X);\\ \mathrm{d}^t_{ok}(C,Y) &\text{ :- } c, \mathrm{d}^t_t(X,Y), \mathrm{pos}(C,X); & \mathrm{d}^t_{ok}(C,Y) &\text{ :- } c, \mathrm{d}^t_f(X,Y), \mathrm{neg}(C,X);\\ &\text{ :- } c, \mathrm{not}\ \mathrm{d}^t_{ok}(C,Y), \mathrm{cl}(C); & \mathrm{d}^t_t(X,X) &\text{ :- } i\ ;\\ c &\text{ :- } \mathrm{not}\ i; & i &\text{ :- } \mathrm{not}\ c;\\ &\text{ :}\sim \mathrm{d}^t_t(X,X); & &\text{ :}\sim i. \end{aligned}$$

Given Proposition 5, it is easy to see that, given some answer set A of $\mathrm{SAT}^{cc}_{\{t\}} \cup D_\varphi$, $\{a \mid \mathrm{d}^t_t(a,a) \in A\}$ is precisely the set of atoms which are true in all models of φ.

5 Applications

In this section, we put our technique to work and show how to use meta-reasoning over answer sets for three application scenarios. The first one is a well-known problem from propositional logic, and we will reuse the example from above. The second example takes a bit more background, but presents a novel method to compute ideal extensions for argumentation frameworks which was also implemented in the logic-programming based argumentation system ASPARTIX [23].[5] Finally, we address Michael Gelfond's epistemic specification, a powerful extension of standard ASP with modal atoms which allow for meta-reasoning over answer sets. In particular, we will consider a certain subclass which is directly amenable to manifolds.

[5] For a web frontend, see `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX`.

5.1 The Unique Minimal Model Problem

As a first example, we show how to encode the problem of deciding whether a given propositional formula φ has a unique minimal model. This problem is known to be in Θ_2^P and to be co-NP-hard (the exact complexity is an open problem). Let I be the intersection of all models of φ. Then φ has a unique minimal model iff I is also a model of φ. We thus use our example from the previous section, and define the program UNIQUE as $\mathrm{SAT}^{cc}_{\{t\}}$ augmented by rules

$$\begin{aligned} ok(C) \text{ :- } & \mathrm{d}^t_t(X,X), \mathrm{pos}(C,X); \\ ok(C) \text{ :- } & \text{not } \mathrm{d}^t_t(X,X), \mathrm{neg}(C,X); \\ \text{ :- } & \text{not } ok(C), \mathrm{cl}(C). \end{aligned}$$

We immediately obtain the following result.

Theorem 1. *For any CNF formula φ, it holds that φ has a unique minimal model, if and only if program* UNIQUE $\cup\, D_\varphi$ *has at least one answer set.*

A slight adaption of this encoding allows us to formalize CWA-reasoning [13] over a propositional knowledge base φ, since the atoms a in φ, for which the corresponding atoms $\mathrm{d}^t_t(a,a)$ are not contained in an answer set of $\mathrm{SAT}^{cc}_{\{t\}} \cup D_\varphi$, are exactly those which are added negated to φ for CWA-reasoning.

5.2 Computing the Ideal Extension

Our second example is from the area of argumentation, where the problem of computing the ideal extension [15] of an abstract argumentation framework was recently shown to be complete for $\mathrm{FP}^{\mathrm{NP}}_{||}$ in [24]. Thus, this task cannot be compactly encoded via normal programs (under usual complexity theoretic assumptions). On the other hand, the complexity shows that employing disjunction is not necessary, if one instead uses weak constraints. We first give the basic definitions following [25].

Definition 9. *An* argumentation framework (AF) *is a pair $F = (A, R)$ where $A \subseteq \mathcal{U}$ is a set of arguments and $R \subseteq A \times A$. $(a, b) \in R$ means that a attacks b. An argument $a \in A$ is* defended *by $S \subseteq A$ (in F) if, for each $b \in A$ such that $(b, a) \in R$, there exists a $c \in S$, such that $(c, b) \in R$. An argument a is admissible (in F) w.r.t. a set $S \subseteq A$ if each $b \in A$ which attacks a is defended by S.*

Semantics for argumentation frameworks are given in terms of so-called extensions. The next definitions introduce two such notions which also underlie the concept of an ideal extension.

Definition 10. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is said to be* conflict-free (in F)*, if there are no $a, b \in S$, such that $(a, b) \in R$. A set S is an* admissible extension *of F, if S is conflict-free in F and each $a \in S$ is admissible in F w.r.t. S. The collection of admissible extensions is denoted by $adm(F)$. An admissible extension S of F is a* preferred extension *of F, if for each $T \in adm(F)$, $S \not\subset T$. The collection of preferred extensions of F is denoted by $pref(F)$.*

The original definition of ideal extensions is as follows [15].

Definition 11. *Let F be an AF. A set S is called* ideal *for F, if $S \in adm(F)$ and $S \subseteq \bigcap_{T \in pref(F)} T$. A maximal (w.r.t. set-inclusion) ideal set of F is called an* ideal extension *of F.*

It is known that each AF possesses a unique ideal extension. In [24], the following algorithm to compute the ideal extension of an AF $F = (A, R)$ is proposed. Let

$$X_F^- = A \setminus \bigcup_{S \in adm(F)} S \text{ and}$$
$$X_F^+ = \{a \in A \mid \forall b, c : (b, a), (a, c) \in R \Rightarrow b, c \in X_F^-\} \setminus X_F^-,$$

and define the AF $F^* = (X_F^+ \cup X_F^-, R^*)$ where the attack relation R^* is given as $R \cap \{(a, b), (b, a) \mid a \in X_F^+, b \in X_F^-\}$. F^* is a bipartite AF in the sense that R^* is a bipartite graph.

Proposition 6 ([24]). *The ideal extension of AF F is given by $\bigcup_{S \in adm(F^*)}(S \cap X_F^+)$.*

The set of all admissible atoms for a bipartite AF F can be computed in polynomial time using Algorithm 1 of [26]. This is basically a fixpoint iteration identifying arguments in X_F^+ that cannot be in an admissible extension: First, arguments in $X_0 = X_F^+$ are excluded, which are attacked by unattacked arguments (which are necessarily in X_F^-), yielding X_1. Now, arguments in X_F^- may be unattacked by X_1, and all arguments in X_1 attacked by such newly unattacked arguments should be excluded. This process is iterated until either no arguments are left or no more argument can be excluded. There may be at most $|X_F^+|$ iterations in this process.

We exploit this technique to formulate an ASP-encoding IDEAL. We first describe a program the answer sets of which characterize admissible extensions. Then, we use the brave manifold of this program in order to determine all arguments contained in some admissible extension. Finally, we extend this manifold program in order to identify F^* and to simulate Algorithm 1 of [26].

The argumentation frameworks will be given to IDEAL as sets of input facts. Given an AF $F = (A, R)$, let $D_F = \{a(x) \mid x \in A\} \cup \{\mathrm{r}(x, y) \mid (x, y) \in R\}$. The program ADM, given by the rules below, computes admissible extensions (cf. [27, 23]):

$$\begin{aligned} \mathrm{in}(X) &\text{ :- } \mathrm{not\ out}(X), a(X); \\ \mathrm{out}(X) &\text{ :- } \mathrm{not\ in}(X), a(X); \\ &\text{ :- } \mathrm{in}(X), \mathrm{in}(Y), \mathrm{r}(X, Y); \\ \mathrm{def}(X) &\text{ :- } \mathrm{in}(Y), \mathrm{r}(Y, X); \\ &\text{ :- } \mathrm{in}(X), \mathrm{r}(Y, X), \mathrm{not\ def}(Y). \end{aligned}$$

Indeed one can show that, given an AF F, the answer sets of $\mathrm{ADM} \cup D_F$ are in a one-to-one correspondence to the admissible extensions of F via the $\mathrm{in}(\cdot)$ predicate. In order to determine the brave consequences of ADM for predicate in, we form $\mathrm{ADM}^{bc}_{\{\mathrm{in}\}}$, and extend it by collecting all brave consequences of $\mathrm{ADM} \cup D_F$ in predicate $\mathrm{in}(\cdot)$, from which we can determine X_F^- (represented by $\mathrm{in}^-(\cdot)$), X_F^+ (represented by $\mathrm{in}^+(\cdot)$, using auxiliary predicate $\mathrm{not\_in}^+(\cdot)$), and R^* (represented by $\mathrm{q}(\cdot, \cdot)$).

$$
\begin{aligned}
\mathrm{in}(X) &\text{ :- } \mathrm{d}^{\mathrm{in}}_{\mathrm{in}}(X,X);\\
\mathrm{in}^-(X) &\text{ :- } a(X), \mathrm{not}\ \mathrm{in}(X);\\
\mathrm{in}^+(X) &\text{ :- } \mathrm{in}(X), \mathrm{not}\ \mathrm{not\_in}^+(X);\\
\mathrm{not\_in}^+(X) &\text{ :- } \mathrm{in}(Y), r(X,Y);\\
\mathrm{not\_in}^+(X) &\text{ :- } \mathrm{in}(Y), r(Y,X);\\
\mathrm{q}(X,Y) &\text{ :- } \mathrm{r}(X,Y), \mathrm{in}^+(X), \mathrm{in}^-(Y);\\
\mathrm{q}(X,Y) &\text{ :- } \mathrm{r}(X,Y), \mathrm{in}^-(X), \mathrm{in}^+(Y).
\end{aligned}
$$

In order to simulate Algorithm 1 of [26], we use the elements in X_F^+ for marking the iteration steps. To this end, we use an arbitrary order $<$ on ASP constants (all ASP systems provide such a predefined order) and define successor, infimum and supremum among the constants representing X_F^+ w.r.t. the order $<$.

$$
\begin{aligned}
\mathrm{nsucc}(X,Z) &\text{ :- } \mathrm{in}^+(X), \mathrm{in}^+(Y), \mathrm{in}^+(Z), X<Y, Y<Z;\\
\mathrm{succ}(X,Y) &\text{ :- } \mathrm{in}^+(X), \mathrm{in}^+(Y), X<Y, \mathrm{not}\ \mathrm{nsucc}(X,Y);\\
\mathrm{ninf}(Y) &\text{ :- } \mathrm{in}^+(X), \mathrm{in}^+(Y), X<Y;\\
\mathrm{nsup}(X) &\text{ :- } \mathrm{in}^+(X), \mathrm{in}^+(Y), X<Y;\\
\mathrm{inf}(X) &\text{ :- } \mathrm{in}^+(X), \mathrm{not}\ \mathrm{ninf}(X);\\
\mathrm{sup}(X) &\text{ :- } \mathrm{in}^+(X), \mathrm{not}\ \mathrm{nsup}(X).
\end{aligned}
$$

We now use this to iteratively determine arguments that are not in the ideal extension, using $\mathrm{nid}(\cdot,\cdot)$, where the first argument is the iteration step. In the first iteration (identified by the infimum) all arguments in X_F^+ which are attacked by an unattacked argument are collected. In subsequent iterations, all arguments from the previous steps are included and augmented by arguments that are attacked by an argument not attacked by arguments in X_F^+ that were not yet excluded in the previous iteration. Finally, arguments in the ideal extension are those that are not excluded from X_F^+ in the final iteration (identified by the supremum).

$$
\begin{aligned}
\mathrm{att}_0(X) &\text{ :- } \mathrm{q}(Y,X);\\
\mathrm{att}_i(J,Z) &\text{ :- } q(Y,Z), \mathrm{in}^+(Y), \mathrm{not}\ \mathrm{nid}(J,Y), \mathrm{in}^+(J);\\
\mathrm{ideal}(X) &\text{ :- } \mathrm{in}^+(X), \mathrm{sup}(I), \mathrm{not}\ \mathrm{nid}(I,X);\\
\mathrm{nid}(I,Y) &\text{ :- } \mathrm{succ}(J,I), \mathrm{nid}(J,Y);\\
\mathrm{nid}(I,Y) &\text{ :- } \mathrm{inf}(I), \mathrm{q}(Z,Y), \mathrm{in}^+(Y), \mathrm{not}\ \mathrm{att}_0(Z);\\
\mathrm{nid}(I,Y) &\text{ :- } \mathrm{succ}(J,I), q(Z,Y), \mathrm{in}^+(Y), \mathrm{not}\ \mathrm{att}_i(J,Z).
\end{aligned}
$$

If we put $\mathrm{ADM}^{bc}_{\{\mathrm{in}\}}$ and all of these additional rules together to form the program IDEAL, we obtain the following result:

Theorem 2. *Let F be an AF and $A \in AS(\mathrm{IDEAL} \cup D_F)$. Then, the ideal extension of F is given by $\{a \mid \mathrm{ideal}(a) \in A\}$.*

5.3 Epistemic Specifications

Epistemic Specifications have been defined in [16], and are an extension of programs as defined in Section 2 by the possible occurrence of epistemic operators K and M. In this paper, we will consider a simple class of epistemic specifications, which includes the main motivating example of [16][6].

A *simple epistemic literal* is one of $\mathrm{K}a$, $\neg\mathrm{K}a$, $\mathrm{M}a$ or $\neg\mathrm{M}a$, where a is an atom as in Section 2. A *simple epistemic specification* is a set of epistemic rules

$$a_1 \vee \cdots \vee a_n \text{ :- } B_1, \ldots, B_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m \tag{1}$$

where $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, $B_1, \ldots, B_k$ are atoms or simple epistemic literals and $a_1, \ldots, a_n, b_{k+1}, \ldots, b_m$ are atoms. We say that an atom a directly modally depends on an atom b if a is one of $a_1, \ldots, a_n$ and b occurs in a simple epistemic literal of $B_1, \ldots, B_k$ in a rule of the form (1). A simple epistemic specification is *modally acyclic* if no atom depends modally on itself in the transitive closure of the direct modal dependency relation. A specification is *one-step modal* if each atom a directly modally depends only on atoms which do not depend modally on other atoms.

Herbrand Universe and Base are defined as for standard logic programs, considering also atoms in simple epistemic literals (but no modal operators). In the context of epistemic specifications, collections of interpretations are called *world views*. Satisfaction of standard atoms by interpretations is defined as usual. Satisfaction of simple epistemic literals is defined with respect to world views: A world view W satisfies $\mathrm{K}a$, written $W \models \mathrm{K}a$, iff $\forall B \in W : a \in B$. W satisfies $\mathrm{M}a$, written $W \models \mathrm{M}a$, iff $\exists B \in W : a \in B$. Moreover, $W \models \neg\mathrm{K}a$ iff $W \not\models \mathrm{K}a$ and $W \models \neg\mathrm{M}a$ iff $W \not\models \mathrm{M}a$.

The *modal reduct* of a simple epistemic specification Π with respect to a world view W, denoted Π^W, is obtained by deleting all epistemic rules of Π of the form (1) where $W \not\models B_i$ for some simple epistemic literal B_i, and by deleting all simple epistemic literals of the remaining rules. Note that Π^W is a standard program without epistemic literals. W is a world view of Π iff $W = AS(\Pi^W)$.

Observe that standard programs without weak constraints are epistemic specifications, and their modal reduct is equal to the original program. These programs therefore have a single world view, the collection of the answer sets of the program.

A one-step modal epistemic specification Π can be split into two specifications Π_1 (the lower part) and Π_2 (the upper part), where $\Pi_1 \cap \Pi_2 = \emptyset$ and $\Pi_1 \cup \Pi_2 = \Pi$, such that Π_1 does not contain K or M, and no head atom of Π_2 occurs in Π_1. This is similar to, and in fact a special case of, splitting epistemic specifications as defined in [28].

In order to be able to compute world views of one-step modal epistemic specifications by means of manifold programs, we would like them to have a single world view. The reason is that it is not clear how to differentiate between a specification having multiple world views and a specification having a single world view that contains all sets of the multiple world views of the first specification. The issues are best explained by an example.

[6] Here we do not consider strong negation (except for negating epistemic operators) in order to keep the framework simple. It can be extended without major efforts to incorporate also strong negation.

Example 8. Consider the following one-step modal epistemic specification

$$\begin{aligned} &\text{:- } a, \mathrm{K}b;\\ &\text{:- } b, \mathrm{K}a;\\ &\text{:- } \mathrm{M}a, \mathrm{M}b;\\ a \vee b &\text{ :- } . \end{aligned}$$

It has two world views, $\{\{a\}\}$ and $\{\{b\}\}$. It is not clear how to find a manifold encoding for this specification which lets one differentiate its output from a manifold encoding of a specification having one world view $\{\{a\}, \{b\}\}$ (for example the specification consisting only of $a \vee b$). The difficulty is that one would have to encode also an indicator in which world view(s) an interpretation occurs, which appears to be a hard, if not impossible, task.

The important observation in the example is that the upper part of the specification can trigger incoherences (in this case because of constraint violations), and for this reason not all answer sets of the lower part necessarily have a corresponding answer set in a world view of the complete specification. A similar issue has been described in [28], where specifications are called *safe* if (for the special case of one-step modal epistemic specifications) the modal reduct of the upper part with respect to the collection of answer sets of the lower part has answer sets when any answer set of the lower part is added as a set of facts, that is if

$$\forall A \in AS(\Pi_1) : AS(\Pi_2^{AS(\Pi_1)} \cup A) \neq \emptyset.$$

For any safe one-step modal epistemic specification Π and one of its world views W, any $A \in W$ extends an $A' \in AS(\Pi_1)$ and, vice versa, each $A' \in AS(\Pi_1)$ is contained in some $A \in W$. Therefore, for any epistemic literal ℓ in Π and any world view W of Π, we have that $W \models \ell$ if and only if $AS(\Pi_1) \models \ell$, and as a consequence $\Pi^W = \Pi^{AS(\Pi_1)}$ and so $W = AS(\Pi^{AS(\Pi_1)})$ is unique.

Example 9. Consider the following variant Π^g of the main motivating example of [16].

$$\begin{aligned} \mathit{eligible}(X) &\text{ :- } \mathit{highGPA}(X);\\ \mathit{eligible}(X) &\text{ :- } \mathit{minority}(X), \mathit{fairGPA}(X);\\ \mathit{notEligible}(X) &\text{ :- } \mathit{notFairGPA}(X), \mathit{notHighGPA}(X);\\ \mathit{interview}(X) &\text{ :- } \neg\mathrm{K}\mathit{eligible}(X), \neg\mathrm{K}\mathit{notEligible}(X). \end{aligned}$$

This (and any extensions by facts) is a safe one-step modal epistemic specification: The first three rules form the lower part Π_1 and the last rule forms the upper part Π_2.

Moreover, observe that due to the considerations above, for the lower part Π_1 of a one-step modal epistemic specification Π, $AS(\Pi_1) \models \mathrm{M}a$ iff $\Pi_1 \models_b a$ ($AS(\Pi_1) \models \neg\mathrm{M}a$ iff $\Pi_1 \not\models_b a$) and $AS(\Pi_1) \models \mathrm{K}a$ iff $\Pi_1 \models_c a$ ($AS(\Pi_1) \models \neg\mathrm{K}a$ iff $\Pi_1 \not\models_c a$) for epistemic literals $\mathrm{M}a, \neg\mathrm{M}a, \mathrm{K}a, \neg\mathrm{K}a$ in Π.

We can then use Proposition 5 in order to simulate the modal reduct Π^W of the unique world view W of Π. In particular,

$$W \models \mathrm{M}p(t_1, \ldots, t_n) \quad \text{iff} \quad \mathrm{d}_p^p(t_1, \ldots, t_n, \boldsymbol{X}) \in AS({\Pi_1}_p^{bc}),$$

(with $\boldsymbol{X}$ being a sequence of suitably chosen variables, cf. Section 4) and

$$W \models \mathrm{K}p(t_1,\ldots,t_n) \quad \text{iff} \quad \mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X}) \in AS(\Pi_1{}^{cc}_p).$$

Moreover, we have $W \models \neg\mathrm{M}p(t_1,\ldots,t_n)$ iff $\mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X}) \notin AS(\Pi_1{}^{bc}_p)$, and $W \models \neg\mathrm{K}p(t_1,\ldots,t_n)$ iff $\mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X}) \notin AS(\Pi_1{}^{cc}_p)$. Making sure that all $\Pi_1{}^{cc}_p$ and $\Pi_1{}^{bc}_p$ use distinct symbols, different also from those in Π, we can form the union of all of these programs.

That means that we can replace each occurrence of $\mathrm{K}p(t_1,\ldots,t_n)$ in Π by the manifold atom $\mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X})$ (and $\neg\mathrm{K}p(t_1,\ldots,t_n)$ by the corresponding default negated atom, i.e. $\mathrm{not}\ \mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X})$) and add $\Pi_1{}^{cc}_{\{p\}}$; symmetrically, we can replace each occurrence of $\mathrm{M}p(t_1,\ldots,t_n)$ by $\mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X})$ (and $\neg\mathrm{M}p(t_1,\ldots,t_n)$ by $\mathrm{not}\ \mathrm{d}^p_p(t_1,\ldots,t_n,\boldsymbol{X})$) and add $\Pi_1{}^{bc}_{\{p\}}$. Let us call the program obtained in this way $\overline{\Pi}$. $\overline{\Pi}$ can be split such that $\Pi_1{}^{bc}_{\{p\}}$ and $\Pi_1{}^{cc}_{\{p\}}$ form the bottom program $\overline{\Pi}_1$, and the partial evaluation $\overline{\Pi}'$ of $\overline{\Pi}$ with respect to $AS(\overline{\Pi}_1)$ coincides with Π^W for the unique world view W of Π. It follows that the restriction of each $A \in AS(\overline{\Pi})$ to the symbols of Π is in W, and for each $A \in W$, an $A' \in AS(\overline{\Pi})$ exists, such that the restriction of A' to symbols in Π is A.

Example 10. Reconsider the main motivating example Π^g of [16] as reported in Example 9. $\overline{\Pi^g}$ is:

$$\begin{aligned}
\mathit{eligible}(X) &\text{ :- } \mathit{highGPA}(X);\\
\mathit{eligible}(X) &\text{ :- } \mathit{minority}(X), \mathit{fairGPA}(X);\\
\mathit{notEligible}(X) &\text{ :- } \mathit{notFairGPA}(X), \mathit{notHighGPA}(X);\\
\mathit{interview}(X) &\text{ :- } \mathrm{not}\ \mathrm{d}^{\mathit{eligible}}_{\mathit{eligible}}(X,X), \mathrm{not}\ \mathrm{d}^{\mathit{notEligible}}_{\mathit{notEligible}}(X,X);\\
\mathrm{d}^{\mathit{eligible}}_{\mathit{eligible}}(X,X_1) &\text{ :- } c_1, \mathrm{d}^{\mathit{eligible}}_{\mathit{highGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{eligible}}_{\mathit{eligible}}(X,X_1) &\text{ :- } c_1, \mathrm{d}^{\mathit{eligible}}_{\mathit{minority}}(X,X_1), \mathrm{d}^{\mathit{eligible}}_{\mathit{fairGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{eligible}}_{\mathit{notEligible}}(X,X_1) &\text{ :- } c_1, \mathrm{d}^{\mathit{eligible}}_{\mathit{notFairGPA}}(X,X_1), \mathrm{d}^{\mathit{eligible}}_{\mathit{notHighGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{eligible}}_{\mathit{eligible}}(X_1,X_1) &\text{ :- } i_1;\\
c_1 &\text{ :- } \mathrm{not}\ i_1;\\
i_1 &\text{ :- } \mathrm{not}\ c_1;\\
&\text{ :}\sim \mathrm{d}^{\mathit{eligible}}_{\mathit{eligible}}(X_1,X_1);\\
&\text{ :}\sim i_1;\\
\mathrm{d}^{\mathit{notEligible}}_{\mathit{eligible}}(X,X_1) &\text{ :- } c_2, \mathrm{d}^{\mathit{notEligible}}_{\mathit{highGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{notEligible}}_{\mathit{eligible}}(X,X_1) &\text{ :- } c_2, \mathrm{d}^{\mathit{notEligible}}_{\mathit{minority}}(X,X_1), \mathrm{d}^{\mathit{notEligible}}_{\mathit{fairGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{notEligible}}_{\mathit{notEligible}}(X,X_1) &\text{ :- } c_2, \mathrm{d}^{\mathit{notEligible}}_{\mathit{notFairGPA}}(X,X_1), \mathrm{d}^{\mathit{notEligible}}_{\mathit{notHighGPA}}(X,X_1);\\
\mathrm{d}^{\mathit{notEligible}}_{\mathit{notEligible}}(X_1,X_1) &\text{ :- } i_2;\\
c_2 &\text{ :- } \mathrm{not}\ i_2;\\
i_2 &\text{ :- } \mathrm{not}\ c_2;\\
&\text{ :}\sim \mathrm{d}^{\mathit{notEligible}}_{\mathit{notEligible}}(X_1,X_1);\\
&\text{ :}\sim i_2.
\end{aligned}$$

This rewriting can be extended to safe modally acyclic epistemic specifications essentially by a repeated application, but special care must be taken of the involved weak constraints.

6 Conclusion

In this paper, we provided a novel method to rewrite ASP programs in such a way that reasoning over all answer sets of the original program can be formulated within the same program. Our method exploits the well-known concept of weak constraints. We illustrated the impact of our method by encoding the problems of (i) deciding whether a propositional formula in CNF has a unique minimal model, (ii) computing the ideal extension of an argumentation framework. For (i) and (ii), known complexity results witness that our encodings are adequate in the sense that efficient ASP encodings without weak constraints or similar constructs are assumed to be infeasible. As a final application we considered (iii) epistemic specifications, where we used our concepts to simulate the semantics of epistemic literals within a single world view (thus we had to restrict ourselves to a particular subclass of epistemic specifications). Our encodings provide evidence that the class of disjunctive (non-disjunctive) safe one-step modal epistemic specifications is easier to evaluate (in Θ_3^P resp. Θ_2^P) as the respective general class of disjunctive (non-disjunctive) epistemic specifications (which have been shown to be hard for Σ_3^P resp. Σ_2^P in [21]).

Concerning related work, we remark that the manifold program for cautious consequences is closely related to the concept of data disjunctions [29] (this paper also contains a detailed discussion about the complexity class Θ_2^P and related classes for functional problems). Concepts similar to manifold programs have also been studied in the area of default logic, where a method for reasoning within a single extension has been proposed [30]. That method uses set-variables which characterize the set of generating defaults of the original extensions. However, such an approach differs considerably from ours as it encodes certain aspects of the semantics (which ours does not), which puts it closer to meta-programming (cf. [31]).

As future work, we intend studying the use of alternative preferential constructs in place of weak constraints. Moreover, we are currently developing a suitable language for expressing reasoning with brave, cautious and definite consequences, allowing also for mixing different reasoning modes. This language should serve as a platform for natural encodings of problems in complexity classes Θ_2^P, Θ_3^P, $\mathrm{FP}_{||}^{\mathrm{NP}}$, and $\mathrm{FP}_{||}^{\Sigma_2^P}$. A first step towards this direction has already been undertaken in [32]; such extensions should also pave the way to simulate a broader class of epistemic specifications.

References

1. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V.W., Truszczyński, M., Warren, D.S., eds.: The Logic Programming Paradigm – A 25-Year Perspective. Springer (1999) 375–398
2. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4) (1999) 241–273

3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2002)
4. Gelfond, M.: Representing knowledge in A-Prolog. In Kakas, A., Sadri, F., eds.: Computational Logic: From Logic Programming into the Future. Number 2408 in LNCS/LNAI, Springer (2002) 413–451
5. Balduccini, M., Gelfond, M., Watson, R., Nogeira, M.: The USA-Advisor: A case study in answer set planning. In Eiter, T., Faber, W., Truszczyński, M., eds.: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001). Volume 2173 of LNCS., Springer (2001) 439–442
6. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for advanced integration of incomplete and inconsistent data. In: Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), ACM Press (2005) 915–917
7. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing configuration knowledge with weight constraint rules. In Provetti, A., Son, T.C., eds.: Proceedings of the 1st International Workshop on Answer Set Programming. (2001)
8. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In Baral, C., Brewka, G., Schlipf, J., eds.: Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007). Volume 4483 of LNCS., Springer (2007) 3–17
9. Bravo, L., Bertossi, L.E.: Logic programs for consistently querying data integration systems. In Gottlob, G., Walsh, T., eds.: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann (2003) 10–15
10. Saccà, D.: Multiple total stable models are definitely needed to solve unique solution problems. Inf. Process. Lett. **58**(5) (1996) 249–254
11. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5) (2000) 845–860
12. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3) (2006) 499–562
13. Reiter, R.: On closed world data bases. In Gallaire, H., Minker, J., eds.: Logic and Databases. Plenum Press (1978) 55–76
14. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artif. Intell. **171**(10-15) (2007) 619–641
15. Dung, P.M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artif. Intell. **171**(10-15) (2007) 642–674
16. Gelfond, M.: Logic programming and reasoning with incomplete information. Annals of Mathematics and Artificial Intelligence **12**(1–2) (1994) 89–116
17. Gelfond, M.: Strong introspection. In: Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991), AAAI Press / The MIT Press (1991) 386–391
18. Wang, K., Zhang, Y.: Nested epistemic logic programs. In Baral, C., Greco, G., Leone, N., Terracina, G., eds.: Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005). Volume 3662 of LNCS. (2005) 279–290
19. Zhang, Y.: Computational properties of epistemic logic programs. In Doherty, P., Mylopoulos, J., Welty, C.A., eds.: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), AAAI Press (2006) 308–317
20. Zhang, Y.: Updating epistemic logic programs. J. Log. Comput. **19**(2) (2009) 405–423
21. Truszczyński, M.: Revisiting epistemic specifications. In: (this volume). Springer (2010)
22. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. **9**(3/4) (1991) 365–386

23. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. Argument and Computation **1**(2) (2010) 144–177
24. Dunne, P.E.: The computational complexity of ideal semantics. Artif. Intell. **173**(18) (2009) 1559–1591
25. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2) (1995) 321–358
26. Dunne, P.E.: Computational properties of argument systems satisfying graph-theoretic constraints. Artif. Intell. **171**(10-15) (2007) 701–729
27. Osorio, M., Zepeda, C., Nieves, J.C., Cortés, U.: Inferring acceptable arguments with answer set programming. In: Proceedings of the 6th Mexican International Conference on Computer Science (ENC 2005), IEEE (2005) 198–205
28. Watson, R.: A splitting set theorem for epistemic specifications. In Baral, C., Truszczyński, M., eds.: Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (NMR 2000). (2000) http://arxiv.org/abs/cs/0003038.
29. Eiter, T., Veith, H.: On the complexity of data disjunctions. Theor. Comput. Sci. **288**(1) (2002) 101–128
30. Delgrande, J.P., Schaub, T.: Reasoning credulously and skeptically within a single extension. Journal of Applied Non-Classical Logics **12**(2) (2002) 259–285
31. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing preferred answer sets by meta-interpretation in answer set programming. TPLP **3**(4-5) (2003) 463–498
32. Faber, W., Woltran, S.: A framework for programming with module consequences. In de Vos, M., Schaub, T., eds.: Proceedings of the LPNMR 2009 Workshop on Software Engineering for Answer Set Programming (SEA 2009). (2009) 34–48 http://sea09.cs.bath.ac.uk/downloads/sea09proceedings.pdf.