A Framework for Programming with Module Consequences*

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. We present a framework which allows to combine answer-set programs in a way that consequences (rather than answer sets themselves) of programs can be used as input to other programs. Situations in which such a composition of programs is required appear in many practical application problems. So far, to deal with such problems, multiple calls to answer-set solvers were usually indispensable, as a direct ASP encoding is often much less obvious. In addition, we provide a technique for compiling such frameworks into a single ASP program which consequently can be evaluated by a single call to an answer-set solver. Our approach relies on the recently introduced concept of manifold programs which make use of weak constraints to identify consequences of programs.

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, data integration, configuration, and many others. This development was fueled by the efficiency of the latest tools for processing ASP programs (so-called ASP solvers) which reached a state that makes ASP applicable for problems of practical importance [5]. The most frequent use of ASP is to compute answer sets (usually stable models) of a logic program from which the solutions of the problem encoded by the program can be obtained.

A somewhat neglected aspect of ASP are its capabilities in terms of consequence relations (or, more general, using queries over answer sets), which are firmly rooted in the tradition of nonmonotonic reasoning. Different to classical settings, in nonmonotonic reasoning there is no canonical consequence relation—the two most studied ones are brave and cautious consequence (sometimes also termed as brave and cautious reasoning); the former is also known as credulous or possible reasoning, the latter is also referred to as skeptical or certain reasoning. In the context of ASP, one is usually interested in a subset of the atomic brave or cautious consequences, which corresponds to a

* 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".

generalization of query answering for databases. In this sense, ASP can also be seen as an evolution of Datalog, a logical database query language.

As an example scenario, let us consider a problem stemming from database systems. A database is inconsistent, if a given database instance does not satisfy some of the constraints imposed. One could argue that the creation of inconsistent databases should be inhibited, but it is also obvious that this is not always possible: For instance, when integrating data, that is, whenever only partitions of the data are maintained in a locally consistent state (for example due to permissions or physical distribution), the merged data is not guaranteed to be consistent. Still, one would like to work with such data.

An attractive approach to dealing with inconsistent data is to consider minimal repairs, that is, considering minimal modifications of the data that establish a consistent state. ASP has been successfully employed for specifying and computing minimal repairs (see, e.g., [6]). In general, there is no unique minimal repair, and the usual workaround is to take a conservative approach and consider those parts of the database which hold in each minimal repair. In the ASP setting, this neatly corresponds to considering the cautious consequences of the program encoding the database repairs.

However, as mentioned earlier, support for consequence relations is somewhat limited in current answer-set programming tools: Not all ASP systems support computing atomic consequences directly, and even if they do, it is usually done as a final processing step, in the sense that it is not possible to use the atomic consequences in the same run in order to do further reasoning. One could try to simulate this kind of reasoning by adding additional rules to the program over which the consequences are computed. However, the following simple example demonstrates the problems of this approach: The program $\{a :\!- \text{not } b \;\; ; \;\; b :\!- \text{not } a\}$ has two answer sets, $\{a\}$ and $\{b\}$, and so its brave atomic consequences are a and b, while there is no cautious atomic consequence. In order to represent the question whether at least one of a or b is a consequence, one could try to add $\{q :\!- a \;\; ; \;\; q :\!- b\}$ to the program and check whether q is an atomic consequence. While this works correctly for brave consequences (a positive answer), it does not for cautious consequences. The reason is that q is indeed a cautious consequence of the modified program thus yielding a positive answer to the query, while neither a nor b is a cautious atomic consequence.

Actually, one would hope to be able to use as many language features that ASP provides in order to reason with atomic consequences of a program, but as seen in the simple example above, existing query interfaces are insufficient for this task. Indeed, if one wants to employ recursion, a hypothetical method of endowing the original program by additional rules is quite obviously inadequate in most cases.

In this work, we introduce a framework that overcomes the limitations outlined above. In particular, we propose a language that encapsulates computing brave, cautious or definite[3] consequences of a program, which can then be utilized in a larger ASP program.

We discuss properties and limitations of the language and describe techniques for implementing a system supporting the language. In particular, we propose an extension of *manifold programs*, that we have recently proposed as a method for compiling query answering into ASP in [8]. In particular, a manifold program M_P of an ASP program

[3] An alternative notion proposed in [7].

P allows for identifying all consequences of a certain type (variants exist for brave, cautious, and definite consequences) within a single answer set. The framework we present here goes beyond the concept of a single manifold program which facilitates query answering wrt. a single program. Our framework permits that the results (i.e., consequences of a certain type) of modules can serve as input for further modules which compile different queries of their own, and so forth. A so-called base program finally collects the result necessary for the overall task and computes its own answer sets. These sets are identified as the answer sets of the entire framework.

However, there is a price to be paid for identifying program consequences by manifolding: While M_P contains optimization constructs (in [8] weak constraints were used, cf. [9]), P should not contain any optimization constructs. There is also a reason for this: While deciding whether one ground atom is a brave (respectively cautious) consequence is NP-complete for normal ground programs and Σ_2^P-complete for disjunctive ground programs (respectively co-NP- and Π_2^P-complete), enumerating brave (or cautious) consequences is complete for the complexity class $\mathrm{FP}_{||}^{\mathrm{NP}}$ for normal ground programs and for $\mathrm{FP}_{||}^{\Sigma_2^P}$ for disjunctive ground programs. It follows that unless the polynomial hierarchy collapses, a program enumerating brave or cautious consequences without optimization constructs does not exist. Moreover, in [9] the relevant decision problems for programs with weak constraints (without different levels) have been shown to be complete for the complexity class Θ_2^P (Θ_3^P in the presence of disjunction), from which the function complexity $\mathrm{FP}_{||}^{\mathrm{NP}}$ ($\mathrm{FP}_{||}^{\Sigma_2^P}$) can be obtained. It follows that the presence of weak constraints is necessary given our current knowledge on $NP \stackrel{?}{=} P$ and also not excessive.

2 Preliminaries

In this section, we review the basic syntax and semantics of ASP with weak constraints, following [10], 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* is either an atom a or its negation $\mathrm{not}\ a$. A *(disjunctive) rule* r is of the form

$$a_1 \vee \cdots \vee a_n \coloneq b_1, \ldots, b_k, \mathrm{not}\ b_{k+1}, \ldots, \mathrm{not}\ b_m$$

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 $B(r) = \{b_1, \ldots, b_k, \mathrm{not}\ b_{k+1}, \ldots, \mathrm{not}\ b_m\}$. 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 $H(r) \coloneq B(r)$.

A *weak constraint* [9] is an expression wc of the form

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

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 $\coloneq\sim B(wc)$.

A *program* P is a finite set of rules and weak constraints. $Rules(P)$ denotes the set of rules and $WC(P)$ 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 HB_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 HB_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 [11], a set $I \subseteq HB_P$ of atoms is an *answer set* for a strong program P iff it is a subset-minimal set that satisfies the *reduct*

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

A set $I \subseteq HB_P$ of atoms 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:

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

where $N_i^P(I)$ 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 $AS(P)$. Note that for programs having weak constraints only of weight and level 1, $H^{Ground(P)}(I)$ 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 [7] of a program P, denoted $P \models_d a$, if $AS(P) \neq \emptyset$ and $a \in A$ holds for all $A \in AS(P)$. The sets of all brave, cautious, definite consequences of a program P are denoted as $BC(P)$, $CC(P)$, $DC(P)$, respectively.

3 Consequence Modules

A module essentially consists of a program, a collection of partially instantiated atoms, and a reasoning mode. It can also receive some predicates as input. The idea is that this

module represents those consequences of the program under the specified reasoning mode which match one of the atoms.

Definition 1. *A* consequence module *(or module, for short) is a quadruple $\langle P, I, O, m\rangle$, where P (the module program) is a strong program, I (the input predicates) is a set of predicates, O (the output atoms) is a set of atoms (possibly containing variables), and m (the reasoning mode) is one of* brave, cautious, definite.

A consequence module framework *(or consequence module program) $F = \langle B, \mathcal{M}\rangle$ consists of a strong program B (called the base program) and a set $\mathcal{M}$ of consequence modules.*

Although the realization of a module looks very similar to known concepts (e.g. modules as defined in [12] or the signature of module atoms in [13]), we remark that the concept of a reasoning mode clearly separates our approach from previous ones. In particular, the output of a module in our approach is just a set of facts (depending on the chosen reasoning mode, this set is obtained from the answer sets of the modules) which serves as input to further modules, while in other approaches the output is usually a collection of answer sets, which have to combined with answer sets of other modules.

We define the universe U_F of a consequence module framework F as the set of all constants appearing in F (if no constant appears in F, an arbitrary constant is added), and the base HB_F of F as the set of all ground literals constructible from the predicate symbols appearing in F and the constants of U_F.

A consequence module framework is stratified on modules if there are no circular dependencies through consequence modules. In the following, let $Pred(\Sigma)$ denote the set of predicates in a syntactic element Σ.

Definition 2 (Stratification on Modules). *A consequence module framework $F = \langle B, \mathcal{M}\rangle$ is* stratified on modules *if there exists a level mapping λ (a stratification) from the set of predicates in F to $\mathbb{N}$, such that for each rule r in B, $\lambda(b) \leq \lambda(h)$ holds for each $b \in Pred(B(r))$ and $h \in Pred(H(r))$, and for each module $\langle P, I, O, m\rangle \in \mathcal{M}$, $\lambda(i) < \lambda(o)$ holds for each $i \in I$ and $o \in Pred(O)$.*

In the following, we will consider only consequence module programs, which are stratified on modules.

The semantics of a stratified consequence module program F is given by an evaluation along one of its level mappings. In other words, the answer sets of F are obtained by simply running the modules in an order of stratification and applying the modules query on the result of each.

Definition 3. *Given a stratified consequence module framework $F = \langle B, \mathcal{M}\rangle$ and λ one of its stratifications, let, for each $i \in \mathbb{N}$, $B_i = \{r \in B \mid i = max\{\lambda(h) \mid h \in H(r)\}\}$ and $\mathcal{M}_i = \{\langle P, I, O, m\rangle \in \mathcal{M} \mid i = max\{\lambda(o) \mid o \in Pred(O)\}\}$.*

Definition 4. *For a module $M = \langle P, I, O, m\rangle$ and a set A of ground atoms, $AS(A \triangleright M) = \{o\sigma \mid o \in O, o\sigma \in X\}$, where σ is a substitution, $X = BC(P \cup A)$ if $m =$ brave, $X = CC(P \cup A)$ if $m =$ cautious, $X = DC(P \cup A)$ if $m =$ definite. For a set $\mathcal{M}$ of modules, let $AS(A \triangleright \mathcal{M}) = \bigcup_{M \in \mathcal{M}} AS(A \triangleright M)$.*

Given a stratified consequence module framework $F = \langle B, \mathcal{M} \rangle$, we then define the following sequence

$$AS_0(F) = AS(B_0) \cup AS(\emptyset \triangleright \mathcal{M}_0)$$
$$AS_i(F) = AS(AS_{i-1}(F) \cup B_i) \cup AS(AS_{i-1}(F) \triangleright \mathcal{M}_i), \textit{ for } i > 0$$

in order to obtain $AS(F) = AS_n(F)$ where $n = max\{\lambda(p) \mid p \in Pred(F)\}$.

It is not hard to see that any stratification will lead to the same answer sets.

Note that the semantics of unstratified consequence module frameworks cannot be defined in this way because of circular dependencies. In this paper we refrain from studying unstratified settings, as their intended semantics is not obvious and possibly gives rise to complexity issues. In a similar way, we do not consider nested modules (that is, occurrences of modules inside module programs). While the intended semantics for these would be more obvious, they would hamper the considerations in Section 5 and possibly also give rise to complexity issues. We believe that the framework in this paper is sufficiently rich to describe many problems, which incur reasoning subtasks, in a natural way. We conjecture that considering an unrestricted language not requiring stratification and allowing for nested consequence modules would result in a relatively high complexity, while the language considered here essentially stays inside ASP complexity bounds.

4 Applications

In this section, we study some example encodings using consequence modules. The first one is a well-known problem from propositional logic, which we will describe in detail, the second one is from planning. Another example would be computing the ideal extension for abstract argumentation frameworks, which was studied in [8], and which we omit here for space reasons.

4.1 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). Our encodings will rely on the following observation which is obvious if one considers models as sets of those atoms which are assigned to true: 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 will use a simple consequence module framework for this task consisting of a single module which will take care of computing the intersection of all models of a propositional CNF formula φ, and a simple base program which, on the one hand, contains a suitable representation of the formula φ (and passes this to the module), and, on the other hand, checks whether the result of the module yields a model of φ.

Let us make this idea more precise. 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,

$$D_\varphi = \{\mathrm{at}(a) \mid a \in A\} \cup \{\mathrm{cl}(i) \mid 1 \leq i \leq n\} \cup$$

$$\{\mathrm{pos}(a,i) \mid \text{atom } a \text{ occurs positively in } c_i\} \cup$$
$$\{\mathrm{neg}(a,i) \mid \text{atom } a \text{ occurs negatively in } c_i\}.$$

For the module, we require a program whose answer sets are in a one-to-one correspondence to models of formulas. For this purpose, consider the program SAT as the set of the following rules.

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

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 $I = \{a \mid \mathit{true}(a) \in M\}$.

Our consequence module will now be given by

$$\mathrm{SAT}_{\mathrm{cautious}} = \langle \mathrm{SAT}, \{\mathrm{at}, \mathrm{cl}, \mathrm{pos}, \mathrm{neg}\}, \{\mathit{true}(X)\}, \mathsf{cautious} \rangle.$$

In fact, using D_φ as input to $\mathrm{SAT}_{\mathrm{cautious}}$, we obtain a result which characterizes those atoms in φ which are true in all models of φ.

For the base program, let us now define the program MODELCHECK as the set of the following rules

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

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 the framework $\langle D_\varphi \cup \mathrm{MODELCHECK}, \{\mathrm{SAT}_{\mathrm{cautious}}\}\rangle$ has an answer set.*

A slight adaption of this encoding allows us to formalize reasoning under the closed-world assumption (CWA), cf. [14], over a propositional knowledge base φ, since the atoms a in φ, for which the corresponding atoms $\mathit{true}(a)$ are *not* contained in any answer set of the program SAT on input D_φ, i.e. those atoms $\mathit{true}(a)$ *not* contained in the output of the module $\mathrm{SAT}_{\mathrm{cautious}}$ on input D_φ, are exactly those which are added negated to φ for CWA-reasoning. In other words, the framework,

$$\langle D_\varphi \cup \{\mathit{false}(X) \text{ :- not } \mathit{true}(X), \mathrm{at}(X)\}, \{\mathrm{SAT}_{\mathrm{cautious}}\}\rangle$$

represents the closed-world closure of φ. Further extensions of the base program can now be used to formulate CWA-reasoning problems.

4.2 Planning

Let us consider secure planning, which is also known as conformant or certain planning [15–18]. Given a description of a nondeterministic transition system involving states (composed of fluents) and actions (occurring between states), a secure plan is a sequence of actions (a plan), which allows for reaching a goal state in any possible outcome of action execution. This means that, starting from a set of initial states, executing a secure plan must not get "stuck" during execution (the subsequent action must always be executable), and must eventually reach the goal state.

Let us consider the problem of deciding whether a given plan is secure. For the language $\mathcal{K}$ of [17], some ASP encodings have been defined in [19]. Let us assume the availability of a program TRAJ (for a given transition system described in $\mathcal{K}$) that has one answer set for each trajectory for a sequence of actions given in the input, where a trajectory is a sequence of states along a path in the given transition system that is labeled by the sequence of actions in the input. For so-called proper transition systems (cf. [17]) it is sufficient to check whether all trajectories end in a goal state.

We can then define a consequence module

$$\mathrm{TRAJ}_{\mathrm{cautious}} = \langle \mathrm{TRAJ}, \{a_1, \ldots, a_n\}, \{f_1(\overline{t_1}), \ldots, f_n(\overline{t_n})\}, \mathsf{cautious} \rangle.$$

where $a_1, \ldots, a_n$ are predicates representing actions (and thus plans), and $f_1(\overline{t_1}), \ldots, f_n(\overline{t_n})$ are atoms representing a goal state. Secure plan checking can then be captured by a framework

$$\langle A_P \cup G = \{ \coloneq \mathrm{not}\ g\ \ ;\ \ g \coloneq f_1(\overline{t_1}), \ldots, f_n(\overline{t_n}) \}, \{\mathrm{TRAJ}_{\mathrm{cautious}}\} \rangle$$

where A_P is an encoding of the plan to be checked. If there is an answer set, the plan is secure.

Now assume that INITEX is a program that computes initial states of a given transition system and which moreover derives an atom i with each initial state (cf. [19]). We define the consequence module

$$\mathrm{INITEX}_{\mathrm{brave}} = \langle \mathrm{INITEX}, \emptyset, \{i\}, \mathsf{brave} \rangle.$$

Moreover, assume the existence of a program ENUMPLANS, which whenever i holds, generates as answer sets all possible plans of a specified length (cf. [19]). Finding secure plans for proper planning domains can then be accomplished by the following framework.

$$\langle \mathrm{ENUMPLANS} \cup G, \{\mathrm{INITEX}_{\mathrm{brave}}, \mathrm{TRAJ}_{\mathrm{cautious}}\} \rangle$$

5 Transforming Consequence Modules to ASP

While one could implement the suggested language using oracles formed of ASP systems (see [20] for a recent realization of such an approach), and so lifting the framework on a metalevel, we propose an alternative method which allows for an implementation using ASP itself. We make use of manifold programs that we have recently proposed in [8], and elaborate on them.

5.1 Manifold Programs

The main idea of manifold programs is to obtain a translation which creates a copy of a given program for each element of a subset of its Herbrand base. Let us first consider the simpler case of propositional programs.

We create a copy of 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 5. *For a strong propositional program P and $S \subseteq HB_P$, define its* manifold *as*

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

where $I^a = \{p^a \mid \text{atom } p \in I\} \cup \{\text{not } p^a \mid \text{not } p \in I\}$ for a set I of atoms and an atom a. We assume $HB_P \cap HB_{P_S^{tr}} = \emptyset$, that is, all symbols in P_S^{tr} are assumed to be fresh.

Example 1. Consider $\Phi = \{p \vee q \coloneq \ ; \ r \coloneq p \ ; \ r \coloneq q\}$ for which $AS(\Phi) = \{\{p, r\}, \{q, r\}\}$, $BC(\Phi) = \{p, q, r\}$ and $CC(\Phi) = DC(\Phi) = \{r\}$. When forming the manifold for $HB_\Phi = \{p, q, r\}$, we obtain

$$\Phi_{HB_\Phi}^{tr} = \left\{ \begin{array}{l} p^p \vee q^p \coloneq c \ ; \ r^p \coloneq c, p^p \ ; \ r^p \coloneq c, q^p \ ; \ c \coloneq \text{not } i \ ; \\ p^q \vee q^q \coloneq c \ ; \ r^q \coloneq c, p^q \ ; \ r^q \coloneq c, q^q \ ; \ i \coloneq \text{not } c \ ; \\ p^r \vee q^r \coloneq c \ ; \ r^r \coloneq c, p^r \ ; \ r^r \coloneq c, q^r \end{array} \right\}$$

Note that given a strong program P and $S \subseteq HB_P$, the construction of P_S^{tr} can be done in polynomial time (w.r.t. the size of P). The answer sets of the transformed program consist of (and extend) 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.

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

$$\{c, p^p, r^p, p^q, r^q, p^r, r^r\}, \{c, \underline{q^p, r^p}, p^q, r^q, p^r, r^r\}, \{c, p^p, r^p, \underline{q^q, r^q}, p^r, r^r\},$$
$$\{c, p^p, r^p, p^q, r^q, \underline{q^r, r^r}\}, \{c, \underline{q^p, r^p}, \underline{q^q, r^q}, p^r, r^r\}, \{c, \underline{q^p, r^p}, p^q, r^q, \underline{q^r, r^r}\},$$
$$\{c, p^p, r^p, \underline{q^q, r^q}, \underline{q^r, r^r}\}, \{c, \underline{q^p, r^p}, \underline{q^q, r^q}, \underline{q^r, r^r}\}.$$

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 6. *Given a strong propositional program P and $S \subseteq HB_P$, 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.

Example 3. For the program Φ as given Example 1, $\Phi_{HB_\Phi}^{bc}$ is given by $\Phi_{HB_\Phi}^{tr} \cup \{:\sim \text{not } p^p \ ; \ :\sim \text{not } q^q \ ; \ :\sim \text{not } r^r \ ; \ :\sim i\}$. We obtain that $AS(\Phi_{HB_\Phi}^{bc}) = \{A_1, A_2\}$, where $A_1 = \{c, p^p, r^p, q^q, r^q, p^r, r^r\}$ and $A_2 = \{c, p^p, r^p, q^q, r^q, q^r, r^r\}$, as these two answer sets are the only ones that violate no weak constraint. We can observe that $\{a \mid a^a \in A_1\} = \{a \mid a^a \in A_2\} = \{p, q, r\} = BC(\Phi)$.

For cautious consequences, we use a similar idea, taking into account that if a program is inconsistent (in the sense that it does not have any answer set), each atom is a cautious consequence.

Definition 7. *Given a strong propositional program P and $S \subseteq HB_P$, let*

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

Example 4. Recall program Φ from Example 1. We have $\Phi_{HB_\Phi}^{cc} = \Phi_{HB_\Phi}^{tr} \cup \{:\sim p^p \ ; \ :\sim q^q \ ; \ :\sim r^r \ ; \ p^p :\!- i \ ; \ q^q :\!- i \ ; \ r^r :\!- i \ ; \ :\sim i\}$. We obtain that $AS(\Phi_{HB_\Phi}^{cc}) = \{A_3, A_4\}$, where $A_3 = \{c, q^p, r^p, p^q, r^q, p^r, r^r\}$ and $A_4 = \{c, q^p, r^p, p^q, r^q, q^r, r^r\}$, as these two answer sets are the only ones that violate only one weak constraint, namely $:\sim r^r$. We observe that $\{a \mid a^a \in A_3\} = \{a \mid a^a \in A_4\} = \{r\} = CC(\Phi)$.

Finally we slightly adapt the construction for definite consequences.

Definition 8. *Given a strong propositional program P and $S \subseteq HB_P$, let*

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

Example 5. Recall program Φ from Example 1. We have $\Phi_{HB_\Phi}^{dc} = \Phi_{HB_\Phi}^{tr} \cup \{:\sim p^p \ ; \ :\sim q^q \ ; \ :\sim r^r \ ; \ i^p :\!- i \ ; \ i^q :\!- i \ ; \ i^r :\!- i :\sim i^p \ ; \ :\sim i^q \ ; \ :\sim i^r \ ; \ :\sim i\}$. 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 $\Phi_{HB_\Phi}^{dc}$.

Proposition 1. *Given a strong propositional program P and $S \subseteq HB_P$, for any $B \in AS(P_S^{bc})$, $\{b \mid b^b \in B\} = BC(P) \cap S$; for any $C \in AS(P_S^{cc})$, $\{c \mid c^c \in C\} = CC(P) \cap S$; for any $D \in AS(P_S^{dc})$, $\{d \mid d^d \in D\} = DC(P) \cap S$.*

Obviously, one can compute all brave, cautious, or definite consequences of a program by choosing $S = HB_P$. We also note that the programs from Definitions 6, 7 and 8 yield multiple answer sets. However each of these yields the same atoms a^a, so it is sufficient to compute one of these. This issue will be addressed in Section 5.2.

We now generalize these techniques to non-ground strong programs. In principle, one could annotate each predicate (rather than atom as before) with ground atoms of

a subset of the Herbrand Base. However, one can also move the annotations to the non-ground level: For example, instead of annotating a rule $p(X,Y) \text{:-} q(X,Y)$ by the set $\{r(a), r(b)\}$ yielding $p^{r(a)}(X,Y) \text{:-} q^{r(a)}(X,Y)$ and $p^{r(b)}(X,Y) \text{:-} q^{r(b)}(X,Y)$ we will annotate using only the predicate r and extend the arguments of p, yielding the compact rule $\mathrm{d}_p^r(X,Y,Z) \text{:-} \mathrm{d}_q^r(X,Y,Z)$ (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). In this particular example we have assumed that the program is to be annotated by all ground instances of $r(Z)$; we will use this assumption also in the following for simplifying the presentation. In practice, one can clearly add atoms to the rule body for restricting the instances of the predicate by which we annotate, in the example this would yield $p^r(X,Y,Z) \text{:-} q^r(X,Y,Z), dom(Z)$ where the predicate dom should be defined appropriately. In the following, recall that $\alpha(p)$ denotes the arity of a predicate p.

Definition 9. *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 $\{a_q^{tr} \mid \text{atom } a \in \mathcal{L}\} \cup \{\text{not } a_q^{tr} \mid \text{not } a \in \mathcal{L}\}$.*

Note that we assume that even though the variables $X_1, \ldots, X_{\alpha(q)}$ are fresh, they will be the same for each a_q^{tr}. One could define similar notions also for partially ground atoms or for sets of atoms characterized by a collection of defining rules, from which we refrain here for the ease of presentation. We define the manifold program in analogy to Definition 5, the only difference being the different way of annotating.

Definition 10. *Given a strong program P and a set S of predicates, define its* manifold *as*

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

Example 6. Consider program $\Psi = \{p(X) \vee q(X) \text{:-} r(X); \ ; \ r(a) \text{:-} \ ; \ r(b) \text{:-} \}$ for which $AS(\Psi) = \{\{p(a), p(b), r(a), r(b)\}, \{p(a), q(b), r(a), r(b)\}, \{q(a), p(b), r(a), r(b)\}, \{q(a), q(b), r(a), r(b)\}\}$. Hence, $BC(\Psi) = \{p(a), p(b), q(a), q(b), r(a), r(b)\}$ and $CC(\Psi) = DC(\Psi) = \{r(a), r(b)\}$. Forming the manifold for $S = \{p\}$, we obtain

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

$AS(\Psi_S^{tr})$ consists of $\{i\}$ plus 16 answer sets, corresponding to all combinations of the four 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 6, 7, and 8, 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 $\text{:}\sim \ \mathrm{d}_p^p(X_1, \ldots, X_{\alpha(p)}, X_1, \ldots, X_{\alpha(p)})$ gives rise to $\{\text{:}\sim \ \mathrm{d}_p^p(c_1, \ldots, c_{\alpha(p)}, c_1, \ldots, c_{\alpha(p)}) \mid c_1, \ldots, c_{\alpha(p)} \in U\}$ where U is the Herbrand base of the program in question, that is one weak constraint for each ground instance annotated by itself.

Definition 11. *Given a strong program P and a set S of predicate symbols, let*

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

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 2. *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, $\Psi_S^{cc} = \Psi_S^{tr} \cup \{:\sim \mathrm{d}_p^p(X_1, X_1)\ ;\ \mathrm{d}_p^p(X_1, X_1) :\!- i\ ;\ :\sim i\}$ 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 $R = \{\mathrm{d}_r^p(a,a), \mathrm{d}_r^p(a,b), \mathrm{d}_r^p(b,a), \mathrm{d}_r^p(b,b)\}$. 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, $\Psi_S^{dc} = \Psi_S^{tr} \cup \{:\sim \mathrm{d}_p^p(X_1, X_1)\ ;\ i_p(X_1) :\!- i\ ;\ :\sim i_p(X_1)\ ;\ :\sim i\}$. 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.

5.2 Transforming Consequence Module Frameworks by Manifolding

The main intuition is to replace each module by a suitable manifold program. In particular, given a module $M = \langle P, I, O, m\rangle$ in a framework F, we intend to create its manifold transform as $P^{bc}_{Pred(O)}$ if $m = \mathsf{brave}$, $P^{cc}_{Pred(O)}$ if $m = \mathsf{cautious}$, $P^{dc}_{Pred(O)}$ if $m = \mathsf{definite}$. Together with suitable *adaptor rules*, which map the transformed predicates back to predicates of the original program, these will be joined to the base program of the framework.

However, there are two main issues to resolve: As remarked earlier, the various manifold programs may admit more than one answer set, which are equivalent with respect to the consequences represented in them. Still, in the context of modules we would like to have a single answer set. The second issue deals with the fact that the manifold transforms of different modules should not interfere with each other.

The first issue can be dealt with by adding penalties in a way that only one answer set remains. In order to avoid interference with other weak constraints, these should be put into a separate level of lower importance. To this end one should fix an arbitrary order of the ground atoms in $X = \{\mathrm{d}^q_q(c_1, \ldots, c_{\alpha(q)}, c'_1, \ldots, c'_{\alpha(q)}) \mid c_i, c'_j \in U_F, c_k \neq c'_k\} \cup \{\mathrm{d}^p_q(c_1, \ldots, c_{\alpha(q)}, c'_1, \ldots, c'_{\alpha(p)}) \mid p \neq q, c_i, c'_j \in U_F\}$ and assigning weights of the exponential sequence $1, 2, 4, 8, \ldots$ to them. This is because each atom should incur a penalty which is greater than the sum of penalties of all preceding atoms. In particular, if $a_0, a_1, \ldots$ is an enumeration of X respecting the chosen order, add a weak constraint $:\sim a_i.[2^i : 1]$. The weak constraints of the original manifold programs should be put in the more important level 2 (higher levels are more important in the semantics of weak constraints), so all weak constraints introduced in Section 5.1 should be extended by $[1 : 2]$ (weight 1 is the default for weights, which was implicitly used in Section 5.1).

The weak constraints introduced in this way can be thought of reducing the set of answer-set candidates in the following way: If answer sets without a_0 exist, further consider only those, otherwise there is no reduction. So the resulting candidates either all do not contain a_0, or all do. Then, among the result, if answer sets without a_1 exist, further consider only those, otherwise there is no reduction. The remaining candidates do not differ on the presence of a_0 and a_1. Continuing like this, in the end the remaining candidates will not differ on the presence of any element in X. If the original set of answer-set candidates differs only on elements in X, then only one answer set remains.

The second modification regards combinability of manifold programs. We would like to be able to simply form the union of all manifold programs replacing the modules. The way in which manifold programs have been defined in Section 5.1, they could in principle share predicate names, which would lead to unwanted interferences. We therefore ensure that each manifold program introduces a unique set of predicates by extending the predicates d^q_p, i_q (p^a, i^a in the propositional case) and i, c by a string uniquely identifying the module, which the manifold program represents.

Definition 12. *For a module $M = \langle P, I, O, m\rangle$, let its* manifold transform *be defined as $T(M) = P^m_{Pred(O)}$, where $P^{\mathsf{brave}}_{Pred(O)}$, $P^{\mathsf{cautious}}_{Pred(O)}$, $P^{\mathsf{definite}}_{Pred(O)}$ correspond to the manifold programs of Section 5.1 with the modifications described above.*

The adaptor rules *for the module M are defined as*

$$A_M = \{p(t_1, \ldots, t_{\alpha(p)}) :\!- \mathrm{d}^p_p(t_1, \ldots, t_{\alpha(p)}, t_1, \ldots, t_{\alpha(p)}) \mid p(t_1, \ldots, t_{\alpha(p)}) \in O\}$$

The manifold program *for a consequence module framework $F = \langle B, \mathcal{M} \rangle$ is then* $T(F) = B \cup \bigcup_{M \in \mathcal{M}} (T(M) \cup A_M)$.

Now we can state the correspondence result.

Proposition 3. *For a consequence module framework F, $AS(F) = AS(T(F)) \cap HB_F$. In fact, there is a one-to-one correspondence between $AS(F)$ and $AS(T(F)) \cap HB_F$.*

Some of the key observations for this correspondence result are that the dependencies of predicates of the base program remain unaltered in $T(F)$, and that module dependencies between predicates in F become standard dependencies in $T(F)$ via the predicates introduced by manifolding. This allows for applying the splitting set theorem of [21] to the program without weak constraints, mimicking the sequence $AS_i(\cdot)$ of Definition 4. However, the manifold parts of $T(F)$ give rise to many answer-set candidates, among which there are also answer sets that contain exactly the consequences under the respective reasoning mode. The combined program $T(F)$ thus will contain many answer-set candidates, but among these there are also precisely the answer sets of the framework, because the latter are defined by replacing the modules by the respective consequences.

The combination of all weak constraints then eliminates all but these candidates. Combining the weak constraints, considering first only those weak constraints described in Section 5.1, has the desired effect because the symbols introduced by $T(M)$ are not contained in any other $T(M')$. Because of this and since these weak constraints all have weight 1, any global optimum must also be an optimum locally for any $T(M)$. Therefore, without adding the additional weak constraints for enforcing uniqueness, the first part of Proposition 3 already holds. One can then show that the differences between multiple answer sets of $T(F)$ representing a single answer set of F is only due to atoms in the sets X described above, which are reduced to precisely one using the method described earlier, thus obtaining a one-to-one correspondence.

6 Conclusion

In this paper, we provided a novel framework for specifying ASP programs, which involve the consequences of subprograms, defining syntax and semantics of the proposed language. We gave examples for problems that possess a comparably natural representation in this language, while a traditional ASP specification is not obvious. Moreover, we proposed a transformation of consequence module frameworks to ASP with weak constraints, based on an adaption of the recently proposed manifold program technique, which allows for using a standard ASP solver supporting weak constraints for computing answer sets of consequence module frameworks.

For future work, we are interested in studying the effects of lifting the restriction of module stratification and of module nesting. We would also like to explore the possibility to use alternative optimization constructs offered by ASP solvers, such as `minimize` supported by `lparse` and `gringo`, in order not to be restricted by the

availability of weak constraints. Also on our agenda is analyzing the relationship between our framework and other proposals for modular ASP (see, e.g. [22, 13]). Finally, we would also like to implement a system that supports consequence module programming.

References

1. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm – A 25-Year Perspective. (1999) 375–398
2. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. AMAI **25**(3–4) (1999) 241–273
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. CUP (2002)
4. Gelfond, M.: Representing knowledge in A-Prolog. In: Computational Logic: From Logic Programming into the Future. LNCS 2408, (2002) 413–451
5. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: LPNMR'07. LNCS 4483, (2007) 3–17
6. Bravo, L., Bertossi, L.E.: Logic programs for consistently querying data integration systems. In: IJCAI 2003, (2003) 10–15
7. Saccà, D.: Multiple total stable models are definitely needed to solve unique solution problems. Inf. Process. Lett. **58**(5) (1996) 249–254
8. Faber, W., Woltran, S.: Manifold answer-set programs for meta-reasoning. In: LPNMR'09. LNCS 5753, (2009) 115–128
9. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5) (2000) 845–860
10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. TOCL **7**(3) (2006) 499–562
11. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. **9**(3/4) (1991) 365–386
12. Oikarinen, E., Janhunen, T.: Achieving compositionality of the stable model semantics for smodels programs. TPLP **8**(5-6) (2008) 717–761
13. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular nonmonotonic logic programming revisited. In: Proceedings of the ICLP'09. LNCS 5649, (2009) 145–159
14. Reiter, R.: On closed world data bases. In: Logic and Databases. Plenum Press (1978) 55–76
15. Goldman, R.P., Boddy, M.S.: Expressive planning and explicit knowledge. In: AIPS'96, AAAI Press (1996) 110–117
16. Smith, D.E., Weld, D.S.: Conformant Graphplan. In: AAAI'98, AAAI Press (1998) 889–896
17. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning: Semantics and complexity. TOCL **5**(2) (2004) 206–263
18. Son, T.C., Tu, P.H., Gelfond, M., Morales, A.R.: An approximation of action theories of and its application to conformant planning. In: LPNMR'05. LNCS 3662, (2005) 172–184
19. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning, II: the DLV$^{\mathcal{K}}$ system. Artif. Intell. **144**(1–2) (2003) 157–211
20. Balduccini, M.: A general method to solve complex problems by combining multiple answer set programs. In: Proceedings ASPOCP'09. (2009)
21. Lifschitz, V., Turner, H.: Splitting a logic program. In: ICLP'94, MIT Press (1994) 23–37
22. Oikarinen, E.: Modularity in Answer Set Programs. PhD thesis, Helsinki University of Technology, Finland (2008)