Modular Equivalence for Normal Logic Programs

Emilia Oikarinen and **Tomi Janhunen** [1]

Abstract. A Gaifman-Shapiro-style architecture of program modules is introduced in the case of normal logic programs under stable model semantics. The composition of program modules is suitably limited by module conditions which ensure the compatibility of the module system with stable models. The resulting module theorem properly strengthens Lifschitz and Turner's splitting set theorem [17] for normal logic programs. Consequently, the respective notion of equivalence between modules, i.e. modular equivalence, proves to be a congruence relation. Moreover, it is shown how our translation-based verification method [15] is accommodated to the case of modular equivalence; and how the verification of weak/visible equivalence can be optimized as a sequence of module-level tests.

1 INTRODUCTION

Answer set programming (ASP) is a promising constraint programming paradigm [20, 19, 11] in which problems are solved by capturing their solutions as *answer sets* or *stable models* of logic programs. The development and optimization of logic programs in ASP gives rise to a meta-level problem of verifying whether subsequent programs are equivalent. To solve this problem, a translation-based approach has been proposed and extended further [14, 23, 21, 25]. The underlying idea is to combine two logic programs P and Q under consideration into two logic programs $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ which have no stable models iff P and Q are *weakly equivalent*, i.e. have the same stable models. This enables the use of the same ASP solver, such as SMODELS or DLV, for the equivalence verification problem as for the search of stable models in general. First experimental results [14, 21] suggest that the translation-based method can be effective and sometimes much faster than a simple cross-check.

As a potential limitation, the translation-based method as described above treats programs as integral entities and therefore no computational advantage is sought by breaking programs into smaller parts, say *modules* of some kind. Such an optimization strategy is largely preempted by the fact that weak equivalence, denoted by $\equiv$, fails to be a *congruence relation* for $\cup$, i.e. weak equivalence is not preserved under substitutions in unions of programs. More formally put, $P \equiv Q$ does not imply $P \cup R \equiv Q \cup R$ in general. The same can be stated about *uniform equivalence* [22] but not about *strong equivalence* [16] which admits substitutions by definition.

From our point of view, strong equivalence seems inappropriate for *fully modularizing* the verification task of weak equivalence because programs P and Q may be weakly equivalent even if they build on respective modules $P_i \subseteq P$ and $Q_i \subseteq Q$ that are not strongly equivalent. For the same reason, program transformations that are known to preserve strong equivalence [4] do not provide an inclusive basis for reasoning about weak equivalence. Nevertheless, there are cases where one can utilize the fact that strong equivalence implies weak equivalence. For instance, if P and Q are composed of strongly equivalent pairs of modules P_i and Q_i for all i, then P and Q can be directly inferred to be strongly/weakly equivalent. These observations about strong equivalence motivate the strive for a weaker congruence relation compatible with weak equivalence at program-level.

To address the lack of a suitable congruence relation in the context of ASP, we propose a new design in this article. The design superficially resembles that of Gaifman and Shapiro [10] but stable model semantics [12] and special module conditions are incorporated. The feasibility of the design is crystallized in a *module theorem* which shows the module system fully compatible with stable models. In fact, the module theorem established here is a proper strengthening of the splitting set theorem established by Lifschitz and Turner [17] in the case of normal logic programs. The main difference is that our result allows negative recursion between modules. Moreover, it enables the introduction of a notion of equivalence, i.e. *modular equivalence*, which turns out to be a proper congruence relation and reduces to weak equivalence for program modules which have a completely specified input and no hidden (auxiliary) atoms. Such modules correspond to normal logic programs without auxiliary atoms. If normal programs P and Q are composed of modularly equivalent modules P_i and Q_i for all i, then P and Q are modularly equivalent, or equivalently stated, weakly equivalent. The notion of modular equivalence opens immediately new prospects as regards the translation-based verification method [14, 21]. First, the method can be tuned for the task of verifying modular equivalence by attaching a *context generator* to program modules in analogy to [25]. Second, we demonstrate how the verification of weak equivalence can be reorganized as a sequence of tests, each of which concentrates on a pair of respective modules in the programs subject to the verification task.

The rest of this article is structured as follows. As a preparatory step, the syntax and semantics of normal logic programs is recalled in Section 2. A summary of equivalence relations follows in Section 3. Section 4 concentrates on specifying program modules as well as establishing the module theorem discussed above. The notion of modular equivalence is then presented in Section 5 which also includes a proof of the congruence property and a brief account of computational complexity. Moreover, connections between modular equivalence and the translation-based method for verifying weak equivalence [14] are worked out. Section 6 briefly contrasts our work with earlier approaches. Finally, a conclusion is given in Section 7.

2 NORMAL LOGIC PROGRAMS

We present *normal logic programs* in the *propositional* case.

Definition 1 *A normal logic program (NLP) is a finite set of rules of the form $h \leftarrow B^+, {\sim}B^-$, where h is an atom, B^+ and B^- are sets of atoms, and ${\sim}B = \{{\sim}b \mid b \in B\}$ for any set of atoms B.*

[1] Helsinki University of Technology (TKK), P.O. Box 5400, FI-02015 TKK, Finland. Email:{Emilia.Oikarinen,Tomi.Janhunen}@tkk.fi

The symbol "$\sim$" denotes *default negation* and we define *default literals* as atoms a or their default negations $\sim a$. A rule consists of a *head* h and a *body* $B^+ \cup \sim B^-$. A rule is a *fact* whenever its body is empty, and "$\leftarrow$" is omitted. A rule is *positive*, if $B^- = \emptyset$. A NLP consisting of positive rules only is a *positive logic program*.

The *Herbrand base* $\mathrm{Hb}(P)$ of a NLP P is any fixed set of atoms containing all atoms appearing in the rules of P. Moreover, the base $\mathrm{Hb}(P)$ is supposed to be finite whenever P is. Under this definition, the Herbrand base of P may contain atoms which have no occurrences in P. This aspect is useful e.g. when programs are optimized and there is a need to keep track of removed atoms. Given a NLP P, an *interpretation* M of P is a subset of $\mathrm{Hb}(P)$ defining which atoms $a \in \mathrm{Hb}(P)$ are true ($a \in M$) and which are false ($a \notin M$). An interpretation $M \subseteq \mathrm{Hb}(P)$ is a *(classical) model* of P, denoted by $M \models P$ iff $B^+ \subseteq M$ and $B^- \cap M = \emptyset$ imply $h \in M$ for each rule $h \leftarrow B^+, \sim B^- \in P$. For a positive program P, an interpretation $M \subseteq \mathrm{Hb}(P)$ is the (unique) *least model* of P, denoted by $\mathrm{LM}(P)$, iff there is no $M' \models P$ such that $M' \subset M$. *Stable models* as proposed by Gelfond and Lifschitz [12] generalize least models for normal logic programs.

Definition 2 *An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of a NLP P iff $M = \mathrm{LM}(P^M)$ where the Gelfond-Lifschitz reduct $P^M = \{h \leftarrow B^+ \mid h \leftarrow B^+, \sim B^- \in P \text{ and } M \cap B^- = \emptyset\}$.*

Stable models are not necessarily unique in general: a NLP may have several stable models or no stable models at all. The set of stable models of a NLP P is denoted by $\mathrm{SM}(P)$.

The *positive dependency relation* $\leq \subseteq \mathrm{Hb}(P) \times \mathrm{Hb}(P)$ of P is the reflexive and transitive closure of a relation $\leq_1$ defined as follows. Given $a, b \in \mathrm{Hb}(P)$, we say that b *depends directly* on a, denoted $a \leq_1 b$, iff there is a rule $b \leftarrow B^+, \sim B^- \in P$ such that $a \in B^+$. The *positive dependency graph* of P, denoted by $\mathrm{Dep}^+(P)$, is a graph with $\mathrm{Hb}(P)$ and $\{\langle b, a\rangle \mid a \leq_1 b\}$ as the sets of vertices and edges, respectively. A *strongly connected component* of $\mathrm{Dep}^+(P)$ is a maximal subset $C \subseteq \mathrm{Hb}(P)$ such that $a \leq b$ holds for all $a, b \in C$. The strongly connected components of $\mathrm{Dep}^+(P)$ partition $\mathrm{Hb}(P)$ into equivalence classes. The dependency relation $\leq$ can then be generalized for strongly connected components: $C_i \leq C_j$, i.e. C_j depends on C_i, iff $c_i \leq c_j$ for any $c_i \in C_i$ and any $c_j \in C_j$.

A *splitting set* for a NLP P [17] is any set $U \subseteq \mathrm{Hb}(P)$ such that for every $h \leftarrow B^+, \sim B^- \in P$, if $h \in U$ then $B^+ \cup B^- \subseteq U$. The *bottom* of P relative to U is $\mathrm{b}_U(P) = \{h \leftarrow B^+, \sim B^- \in P \mid \{h\} \cup B^+ \cup B^- \subseteq U\}$, and the *top* of P relative to U is $\mathrm{t}_U(P) = P \setminus \mathrm{b}_U(P)$. The top can be partially evaluated with respect to an interpretation $X \subseteq U$. The result is a program $\mathrm{e}(\mathrm{t}_U(P), X)$ that contains a rule $h \leftarrow (B^+ \setminus U), \sim (B^- \setminus U)$ for each $h \leftarrow B^+, \sim B^- \in \mathrm{t}_U(P)$ such that $B^+ \cap U \subseteq X$ and $(B^- \cap U) \cap X = \emptyset$. Given a splitting set U for a NLP P, a *solution* to P with respect to U is a pair $\langle X, Y\rangle$ such that $X \subseteq U$, $Y \subseteq \mathrm{Hb}(P) \setminus U$, $X \in \mathrm{SM}(\mathrm{b}_U(P))$, and $Y \in \mathrm{SM}(\mathrm{e}(\mathrm{t}_U(P), X))$. The *splitting set theorem* by Lifschitz and Turner [17] relates solutions with stable models. Given a NLP P, a splitting set U for P, and $M \subseteq \mathrm{Hb}(P)$, it holds that $M \in \mathrm{SM}(P)$ iff $\langle M \cap U, M \setminus U\rangle$ is a solution to P with respect to U.

3 NOTIONS OF EQUIVALENCE

Next we briefly review a number of equivalence relations for NLPs. As stated in Section 1, two NLPs P and Q are (weakly) equivalent, denoted by $P \equiv Q$, iff $\mathrm{SM}(P) = \mathrm{SM}(Q)$. Lifschitz et al. [16] introduce a much stronger notion: P and Q are *strongly equivalent*, denoted $P \equiv_\mathrm{s} Q$, iff $P \cup R \equiv Q \cup R$ for any NLP R acting as a context. Yet another relation originates from the database community [22]: P and Q are *uniformly equivalent*, denoted $P \equiv_\mathrm{u} Q$, iff $P \cup F \equiv Q \cup F$ for any set of facts F. It is clear that $P \equiv_\mathrm{s} Q$ implies $P \equiv_\mathrm{u} Q$ which implies $P \equiv Q$, but not vice versa (in both cases).

Strongly equivalent logic programs are semantics preserving substitutes of each other and thus $\equiv_\mathrm{s}$ can be understood as a *congruence relation* among NLPs, i.e. if $P \equiv_\mathrm{s} Q$, then $P \cup R \equiv_\mathrm{s} Q \cup R$ for all NLPs R. On the other hand, $\equiv_\mathrm{u}$ is not a congruence for $\cup$, as shown in Example 1 below and the same applies to $\equiv$. Thus $\equiv$ and $\equiv_\mathrm{u}$ are best suited for program-level rather than module-level comparisons.

Example 1 *[4, Example 1] Consider NLPs $P = \{a.\}$ and $Q = \{a \leftarrow \sim b.\ a \leftarrow b.\}$. It holds $P \equiv_\mathrm{u} Q$, but $P \cup R \not\equiv Q \cup R$ for $R = \{b \leftarrow a.\}$. Thus $P \not\equiv_\mathrm{s} Q$ and $\equiv_\mathrm{u}$ is not a congruence for $\cup$.*

For $P \equiv Q$ to hold, the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ have to be identical subsets of $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$, respectively, and the same can be stated about $\equiv_\mathrm{s}$ and $\equiv_\mathrm{u}$. This makes these notions of equivalence less useful if $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$ differ by some atoms which formalize some auxiliary concepts and are not trivially false. Following the ideas from [13] we partition $\mathrm{Hb}(P)$ into its *visible* and *hidden* parts $\mathrm{Hb}_\mathrm{v}(P)$ and $\mathrm{Hb}_\mathrm{h}(P)$, respectively. The idea behind *visible equivalence* is that atoms in $\mathrm{Hb}_\mathrm{h}(P)$ and $\mathrm{Hb}_\mathrm{h}(Q)$ are local to P and Q and negligible as regards the equivalence of P and Q.

Definition 3 *[13] Normal logic programs P and Q are visibly equivalent, denoted by $P \equiv_\mathrm{v} Q$, iff $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q)$ and there is a bijection $f : \mathrm{SM}(P) \to \mathrm{SM}(Q)$ such that for all interpretations $M \in \mathrm{SM}(P)$, $M \cap \mathrm{Hb}_\mathrm{v}(P) = f(M) \cap \mathrm{Hb}_\mathrm{v}(Q)$.*

Note that the number of stable models is preserved under $\equiv_\mathrm{v}$. Such a strict correspondence of models is much dictated by the ASP methodology: the stable models of a program usually correspond to the solutions of the problem being solved and thus the exact preservation of models is highly significant. In the fully visible case, i.e. $\mathrm{Hb}_\mathrm{h}(P) = \mathrm{Hb}_\mathrm{h}(Q) = \emptyset$, the relation $\equiv_\mathrm{v}$ becomes very close to $\equiv$. The only difference is the requirement $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ insisted by $\equiv_\mathrm{v}$. This is of little importance as Herbrand bases can always be extended to meet $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. Since weak equivalence is not a congruence, visible equivalence cannot be a congruence either.

The *relativized variants* of $\equiv_\mathrm{s}$ and $\equiv_\mathrm{u}$ introduced by Woltran [25] allow the context to be constrained using a set of atoms A.

Definition 4 *Two NLPs P and Q are strongly equivalent relative to A, denoted by $P \equiv_\mathrm{s}^A Q$, iff $P \cup R \equiv Q \cup R$ for all NLPs R over the set of atoms A; uniformly equivalent relative to A, denoted by $P \equiv_\mathrm{u}^A Q$, iff $P \cup F \equiv Q \cup F$ for all sets of facts $F \subseteq A$.*

Setting $A = \emptyset$ in the above reduces both $P \equiv_\mathrm{s}^A Q$ and $P \equiv_\mathrm{u}^A Q$ to $\equiv$. Thus neither of them is a congruence for $\cup$.

Eiter et al. [7] use *equivalence frames* to capture various kinds of equivalence relations such as those defined above. The relation $\equiv_\mathrm{v}$ makes an exception in this respect as it does not fit into equivalence frames based on *projected answer sets*. A projective variant of Definition 3 would require $\{M \cap \mathrm{Hb}_\mathrm{v}(P) \mid M \in \mathrm{SM}(P)\} = \{N \cap \mathrm{Hb}_\mathrm{v}(Q) \mid N \in \mathrm{SM}(Q)\}$. As a consequence, the number of answer sets might not be preserved which we find to contrast the general nature of ASP as discussed after Definition 3.

4 MODULAR LOGIC PROGRAMS

We define *logic program modules* similarly to [10] but consider the case of NLPs instead of positive (disjunctive) logic programs.

Definition 5 *A triple $\mathbb{P} = (P, I, O)$ is a logic program module, if*

1. *P is a finite set of rules of the form $h \leftarrow B^+, {\sim}B^-$;*
2. *I and O are sets of propositional atoms such that $I \cap O = \emptyset$; and*
3. *$\mathrm{Head}(P) \cap I = \emptyset$ where $\mathrm{Head}(P) = \{h \mid h \leftarrow B^+, {\sim}B^- \in P\}$.*

The Herbrand base of module $\mathbb{P}$, $\mathrm{Hb}(\mathbb{P})$, is the set of atoms appearing in P combined with $I \cup O$. Intuitively the set I defines the *input* of a module and the set O is the *output*. The atoms in $I \cup O$ are visible, i.e. the visible Herbrand base of module $\mathbb{P}$ is $\mathrm{Hb}_\mathrm{v}(\mathbb{P}) = I \cup O$. Notice that $I \cup O$ can also contain atoms not appearing in P similarly to the possibility of having additional atoms in the Herbrand bases of NLPs. All other atoms are hidden, i.e. $\mathrm{Hb}_\mathrm{h}(\mathbb{P}) = \mathrm{Hb}(\mathbb{P}) \setminus \mathrm{Hb}_\mathrm{v}(\mathbb{P})$.

For the composition of modules we take the union of the disjoint sets of rules involved in them in analogy to [10]. The conditions given in [10] are not yet sufficient for our purposes, and we impose a further restriction denying positive recursion between modules.

Definition 6 *Consider $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ and let $C_1, \ldots, C_n$ be the strongly connected components of $\mathrm{Dep}^+(P_1 \cup P_2)$. There is a positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$, if $C_i \cap O_1 \neq \emptyset$ and $C_i \cap O_2 \neq \emptyset$ for some component C_i.*

The idea is that all inter-module dependencies go through the input/output interface of the modules, and hidden atoms are local to each module. If there is a strongly connected component C_i in $\mathrm{Dep}^+(P_1 \cup P_2)$ containing atoms from both O_1 and O_2, we know that, some output atom a of $\mathbb{P}_1$ depends on some output atom b of $\mathbb{P}_2$ which again depends on a. This yields a positive recursion.

Definition 7 *The join of modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$, denoted by $\mathbb{P}_1 \sqcup \mathbb{P}_2$, is defined if*

1. *$O_1 \cap O_2 = \emptyset$;*
2. *$\mathrm{Hb}_\mathrm{h}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) = \mathrm{Hb}_\mathrm{h}(\mathbb{P}_2) \cap \mathrm{Hb}(\mathbb{P}_1) = \emptyset$; and*
3. *there is no positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$.*

Then $\mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2)$.

Note that the first condition is implied by the third, and the second can be circumvented in practice by renaming the hidden atoms uniquely for each module. The join operation is *commutative* and *associative* whenever the respective joins are defined. The following hold for $\mathbb{P}_1 \sqcup \mathbb{P}_2$: $P_1 \cap P_2 = \emptyset$, $\mathrm{Hb}_\mathrm{h}(\mathbb{P}_1) \cap \mathrm{Hb}_\mathrm{h}(\mathbb{P}_2) = \emptyset$, and $\mathrm{Hb}_\mathrm{v}(\mathbb{P}_1) \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2) = (I_1 \cap I_2) \cup (I_1 \cap O_2) \cup (I_2 \cap O_1)$. Also, $\mathrm{Hb}(\mathbb{P}_1 \sqcup \mathbb{P}_2) = \mathrm{Hb}(\mathbb{P}_1) \cup \mathrm{Hb}(\mathbb{P}_2)$, and similarly for the hidden and visible part of $\mathrm{Hb}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$, respectively. The conditions above impose no restrictions on positive dependencies *inside* modules or on *negative* dependencies in general. The input of $\mathbb{P}_1 \sqcup \mathbb{P}_2$ can be smaller than the union of inputs of individual modules because $\mathbb{P}_1$ may provide input for $\mathbb{P}_2$, and conversely, as demonstrated below.

Example 2 *The join of $\mathbb{P} = (\{a \leftarrow {\sim}b.\}, \{b\}, \{a\})$ and $\mathbb{Q} = (\{b \leftarrow {\sim}a.\}, \{a\}, \{b\})$ is $\mathbb{P} \sqcup \mathbb{Q} = (\{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\}, \emptyset, \{a, b\})$.*

The stable semantics of a module is defined with respect to a given input, i.e. a subset of the input atoms of the module. Input is seen as a set of facts (or a database) to be added to the module.

Definition 8 *The instantiation of a module $\mathbb{P} = (P, I, O)$ with an input $A \subseteq I$ is $\mathbb{P}(A) = \mathbb{P} \sqcup \mathbb{F}_A$, where $\mathbb{F}_A = (\{a. \mid a \in A\}, \emptyset, I)$.*

Note that $\mathbb{P}(A) = (P \cup \{a. \mid a \in A\}, \emptyset, I \cup O)$ is essentially a NLP with $\mathrm{Hb}_\mathrm{v}(\mathbb{P}(A)) = I \cup O$, and we can generalize the stable model semantics for modules. In the sequel $\mathbb{P}(A)$ is identified with the respective NLP $P \cup F_A$, where $F_A = \{a. \mid a \in A\}$, and $M \cap I$ acts as a particular input with respect to which the module is instantiated.

Definition 9 *An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a stable model of $\mathbb{P} = (P, I, O)$, denoted $M \in \mathrm{SM}(\mathbb{P})$, iff $M = \mathrm{LM}(P^M \cup F_{M \cap I})$.*

A concept of *compatibility* is used to describe when $M_1 \in \mathrm{SM}(\mathbb{P}_1)$ can be combined with $M_2 \in \mathrm{SM}(\mathbb{P}_2)$: stable models M_1 and M_2 are *compatible*, iff $M_1 \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1)$. Theorem 1 relates program-level stability with module-level stability.

Theorem 1 *(Module theorem). If $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined for modules $\mathbb{P}_1$ and $\mathbb{P}_2$, then $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ iff $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2)$, and M_1 and M_2 are compatible.*

Proof sketch. Suppose $\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 = (P, I, O)$ is defined for modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$. "$\Rightarrow$" Let $M \in \mathrm{SM}(\mathbb{P})$. Then $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1)$ and $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2)$ are clearly compatible and it is straightforward to show that conditions 1 and 2 in Definition 7 imply $M_1 \in \mathrm{SM}(\mathbb{P}_1)$ and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$. "$\Leftarrow$" Let $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ be compatible and define $M = M_1 \cup M_2$. There is a strict total ordering $<$ for the strongly connected components C_i of $\mathrm{Dep}^+(P)$ such that if $C_i < C_j$, then $C_i \leq C_j$ and $C_j \not\leq C_i$; or $C_i \not\leq C_j$ and $C_j \not\leq C_i$. Let $C_1 < \cdots < C_n$ be such an ordering. Show that exactly one of the following holds for each C_i: (i) $C_i \subseteq I$, (ii) $C_i \subseteq O_1 \cup \mathrm{Hb}_\mathrm{h}(\mathbb{P}_1)$, or (iii) $C_i \subseteq O_2 \cup \mathrm{Hb}_\mathrm{h}(\mathbb{P}_2)$. Finally, show by induction that $M \cap (\cup_{i=1}^k C_i) = \mathrm{LM}(P^M \cup F_{M \cap I}) \cap (\cup_{i=1}^k C_i)$ for $0 \leq k \leq n$ by applying the splitting set theorem [17] to $P^M \cup F_{M \cap I}$. □

Example 3 shows that condition 3 in Definition 7 is necessary to guarantee that local stability implies global stability.

Example 3 *Consider $\mathbb{P}_1 = (\{a \leftarrow b.\}, \{b\}, \{a\})$ and $\mathbb{P}_2 = (\{b \leftarrow a.\}, \{a\}, \{b\})$ with $\mathrm{SM}(\mathbb{P}_1) = \mathrm{SM}(\mathbb{P}_2) = \{\emptyset, \{a, b\}\}$. The join of $\mathbb{P}_1$ and $\mathbb{P}_2$ is not defined because of positive recursion (conditions 1 and 2 in Definition 7 are satisfied, however). For a NLP $P = \{a \leftarrow b.\ b \leftarrow a.\}$, we get $\mathrm{SM}(P) = \{\emptyset\}$. Thus, the positive dependency between a and b excludes $\{a, b\}$ from $\mathrm{SM}(P)$.*

Theorem 1 is strictly stronger than the splitting set theorem [17] for NLPs. If U is a splitting set for a NLP P, then $P = \mathbb{B} \sqcup \mathbb{T} = (\mathrm{b}_U(P), \emptyset, U) \sqcup (\mathrm{t}_U(P), U, \mathrm{Hb}(P) \setminus U)$, and furthermore $M_1 \in \mathrm{SM}(\mathbb{B})$ and $M_2 \in \mathrm{SM}(\mathbb{T})$ iff $\langle M_1, M_2 \setminus U \rangle$ is a solution for P with respect to U. On the other hand the splitting set theorem cannot be applied to e.g. $\mathbb{P} \sqcup \mathbb{Q}$ from Example 2, since neither $\{a\}$ nor $\{b\}$ is a splitting set. Our theorem also strengthens a module theorem given in [13, Theorem 6.22] to cover NLPs involving positive body literals. Theorem 1 can easily be generalized for modules consisting of several submodules. Although Theorem 1 enables the computation of stable models on a module-by-module basis, it leaves us the task of excluding mutually incompatible combinations of stable models.

5 MODULAR EQUIVALENCE

The notion of *modular equivalence* combines features from relativized uniform equivalence and visible equivalence.

Definition 10 *Logic program modules $\mathbb{P} = (P, I_P, O_P)$ and $\mathbb{Q} = (Q, I_Q, O_Q)$ are modularly equivalent, denoted by $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$, iff*

1. *$I_P = I_Q = I$ and $O_P = O_Q = O$, and*
2. *$\mathbb{P}(A) \equiv_\mathrm{v} \mathbb{Q}(A)$ for all $A \subseteq I$.*

Modular equivalence is very close to $\equiv_\mathrm{v}$ defined for modules. As a matter a fact, if Definition 3 is generalized for modules, the second condition in Definition 10 can be revised to $\mathbb{P} \equiv_\mathrm{v} \mathbb{Q}$. That, however,

is not enough to cover the first condition in Definition 10, as $\equiv_{\mathrm{v}}$ only enforces $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$. If $I = \emptyset$, modular equivalence coincides with visible equivalence. If $O = \emptyset$, then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ means that $\mathbb{P}$ and $\mathbb{Q}$ have the same number of stable models on each input.

Furthermore, in the *fully visible case*, i.e. if $\mathrm{Hb}_{\mathrm{h}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q}) = \emptyset$, modular equivalence can be seen as a special case of A-uniform equivalence for $A = I$. Recall, however, the restrictions $\mathrm{Head}(P) \cap I = \mathrm{Head}(Q) \cap I = \emptyset$ imposed by module structure. With a further restriction $I = \emptyset$, modular equivalence coincides with weak equivalence. Setting $I = \mathrm{Hb}(\mathbb{P})$ would give uniform equivalence, but the restriction $\mathrm{Head}(P) \cap I = \emptyset$ leaves room for the empty module only.

Since $\equiv_{\mathrm{v}}$ is not a congruence relation for $\cup$, neither is modular equivalence. The situation changes, however, if one considers the join operation $\sqcup$ which suitably restricts possible contexts.

Theorem 2 *Let $\mathbb{P}, \mathbb{Q}$ and $\mathbb{R}$ be logic program modules. If $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ and both $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined, then $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$.*

Proof sketch. Consider $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ such that $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, and $\mathbb{R} = (R, I_R, O_R)$ such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. It follows from Theorem 1 that $M_P = M \cap \mathrm{Hb}(\mathbb{P}) \in \mathrm{SM}(\mathbb{P})$ and $M_R = M \cap \mathrm{Hb}(\mathbb{R}) \in \mathrm{SM}(\mathbb{R})$ for any $M \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$. Since $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, there is a bijection $f : \mathrm{SM}(\mathbb{P}) \to \mathrm{SM}(\mathbb{Q})$ such that $M_P \cap (O \cup I) = f(M_P) \cap (O \cup I)$. Define $M_Q = f(M_P)$ and apply Theorem 1 to show that $M_Q \cup M_R \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$. Finally show that $g : \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R}) \to \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$ defined as $g(M) = f(M \cap \mathrm{Hb}(\mathbb{P})) \cup (M \cap \mathrm{Hb}(\mathbb{R}))$ is a bijection satisfying conditions in Definition 3. Thus $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$. □

It is instructive to consider a potentially stronger variant of $\equiv_{\mathrm{m}}$ defined in analogy to $\equiv_{\mathrm{s}}$ [16]: $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$ holds for all $\mathbb{R}$ such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. However, Theorem 2 implies that $\equiv^{\mathrm{s}}_{\mathrm{m}}$ adds nothing to $\equiv_{\mathrm{m}}$ since $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

As regards the computational complexity, deciding $\equiv_{\mathrm{m}}$ is **coNP**-hard in general, since deciding $P \equiv Q$ reduces to deciding $(P, \emptyset, \mathrm{Hb}(P)) \equiv_{\mathrm{m}} (Q, \emptyset, \mathrm{Hb}(Q))$. If $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}_{\mathrm{h}}(Q) = \emptyset$, deciding $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ reduces to deciding $P \equiv^{I}_{\mathrm{u}} Q$ [25]. Thus deciding $\equiv_{\mathrm{m}}$ is **coNP**-complete in the fully visible case. In the other extreme, if $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = \emptyset$, then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $|\mathrm{SM}(\mathbb{P})| = |\mathrm{SM}(\mathbb{Q})|$. This suggests a much higher computational complexity of deciding $\equiv_{\mathrm{m}}$ in general because classical models can be captured with stable models [20] and the counting problem #SAT is #**P**-complete [24].

A way to govern the computational complexity is to limit the use of hidden atoms as in the case of $\equiv_{\mathrm{v}}$ [15]. Therefrom we adopt the property of having *enough visible atoms* (EVA). For an interpretation $M_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$ for the visible part of a NLP P, the *hidden part* $P_{\mathrm{h}}/M_{\mathrm{v}}$ of P relative M_{v} contains $h \leftarrow B^{+}_{\mathrm{h}}, {\sim}B^{-}_{\mathrm{h}}$ for each rule $h \leftarrow B^{+}, {\sim}B^{-} \in P$ such that $h \in \mathrm{Hb}_{\mathrm{h}}(P)$ and $M_{\mathrm{v}} \models B^{+}_{\mathrm{v}} \cup {\sim}B^{-}_{\mathrm{v}}$.

Definition 11 *A NLP P has enough visible atoms iff $P_{\mathrm{h}}/M_{\mathrm{v}}$ has a unique stable model for every interpretation $M_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$.*

The idea behind the EVA property is that the interpretation of $\mathrm{Hb}_{\mathrm{h}}(P)$ is uniquely determined for each interpretation of $\mathrm{Hb}_{\mathrm{v}}(P)$. Consequently, the stable models of P can be distinguished from each other on the basis of their visible parts. By the EVA assumption [15], the verification of $\equiv_{\mathrm{v}}$ becomes **coNP**-complete for SMODELS programs[2] involving hidden atoms—enabling the translation-based method [14] for $\equiv_{\mathrm{v}}$. Although verifying the EVA property can be hard in general, there are syntactic subclasses of NLPs (e.g. those for which $P_{\mathrm{h}}/M_{\mathrm{v}}$ is always stratified) with the EVA property. It should be stressed that the use of visible atoms is not limited in this setting.

[2] This class of programs includes normal logic programs.

As for $\equiv_{\mathrm{v}}$, the EVA assumption is equally important in conjunction with $\equiv_{\mathrm{m}}$ which becomes evident once we work out the interconnections of $\equiv_{\mathrm{v}}$ and $\equiv_{\mathrm{m}}$. We begin by describing ways in which modular equivalence can be exploited in the verification of visible/weak equivalence. One concrete step in this respect is to reduce the problem of verifying $\equiv_{\mathrm{m}}$ to that of $\equiv_{\mathrm{v}}$ by introducing a special module $\mathbb{G}_I$ that acts as a context generator in analogy to [25].

Theorem 3 *If $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = O \cup I$ holds for $\mathbb{P}$ and $\mathbb{Q}$, then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ where $\mathbb{G}_I = (\{a \leftarrow {\sim}\overline{a}.\ \overline{a} \leftarrow {\sim}a \mid a \in I\}, \emptyset, I)$ generates all possible inputs for $\mathbb{P}$ and $\mathbb{Q}$.*

Proof sketch. Note that $\mathbb{G}_I$ has $2^{|I|}$ stable models of the form $A \cup \{\overline{a} \mid a \in I \setminus A\}$ where $A \subseteq I$. Thus $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{P} \sqcup \mathbb{G}_I$ and $\mathbb{Q} \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ follow by Definitions 2 and 3 and Theorem 1. It follows that $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P}(A) \equiv_{\mathrm{v}} \mathbb{Q}(A)$ for all $A \subseteq I$ iff $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$. □

Consequently, the translation-based method from [15] can be used to decide $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ given that $\mathbb{P}$ and $\mathbb{Q}$ have enough visible atoms ($\mathbb{G}_I$ has the EVA property trivially). The task is to show that $\mathrm{EQT}(\mathbb{P} \sqcup \mathbb{G}_I, \mathbb{Q} \sqcup \mathbb{G}_I)$ and $\mathrm{EQT}(\mathbb{Q} \sqcup \mathbb{G}_I, \mathbb{P} \sqcup \mathbb{G}_I)$ have no stable models.

The introduction of $\equiv_{\mathrm{m}}$ was much motivated by the need of modularizing the verification of $\equiv$.[3] To make this idea clear, we propose a strategy to utilize $\equiv_{\mathrm{m}}$ in the task of verifying the visible/weak equivalence of P and Q. It is required that the module structure for P and Q is either specified explicitly or detected automatically by computing the strongly connected components of $\mathrm{Dep}^{+}(P)$ and $\mathrm{Dep}^{+}(Q)$. Assuming that Q is obtained from P through local modifications, it is likely that these components are pairwise compatible and we can partition P and Q so that $P = \mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ and $Q = \mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_n$ where $\mathbb{P}_i$ and $\mathbb{Q}_i$ have the same input and output sets for all i and $\mathbb{P}_i = \mathbb{Q}_i$ might hold for a number of i's. Now, verifying $\mathbb{P}_i \equiv_{\mathrm{m}} \mathbb{Q}_i$ for every i is not of interest as $\mathbb{P}_i \not\equiv_{\mathrm{m}} \mathbb{Q}_i$ does not necessarily imply $P \not\equiv_{\mathrm{v}} Q$. However, the verification of $P \equiv_{\mathrm{v}} Q$ can still be organized as a sequence of n module-level tests

$$(\bigsqcup_{j=1}^{i-1} \mathbb{Q}_j) \sqcup \mathbb{P}_i \sqcup (\bigsqcup_{j=i+i}^{n} \mathbb{P}_j) \equiv_{\mathrm{m}} (\bigsqcup_{j=1}^{i-1} \mathbb{Q}_j) \sqcup \mathbb{Q}_i \sqcup (\bigsqcup_{j=i+1}^{n} \mathbb{P}_j) \quad (1)$$

where $1 \leq i \leq n$. The resulting chain of equalities conveys $P \equiv_{\mathrm{v}} Q$ under the assumption that P and Q have a completely specified input. If not, then $\equiv_{\mathrm{m}}$ can be addressed in a similar fashion using (1).

Example 4 *Consider $P = \mathbb{P}_1 \sqcup \mathbb{P}_2$ and $Q = \mathbb{Q}_1 \sqcup \mathbb{Q}_2$ where $\mathbb{P}_1 = (\{c \leftarrow {\sim}a.\}, \{a, b\}, \{c\})$, $\mathbb{P}_2 = (\{a \leftarrow b.\}, \emptyset, \{a, b\})$, $\mathbb{Q}_1 = (\{c \leftarrow {\sim}b.\}, \{a, b\}, \{c\})$ and $\mathbb{Q}_2 = (\{b \leftarrow a.\}, \emptyset, \{a, b\})$. Now $\mathbb{P}_1 \not\equiv_{\mathrm{m}} \mathbb{Q}_1$ but $\mathbb{P}_1 \equiv_{\mathrm{v}} \mathbb{Q}_1$ for all inputs generated by $\mathbb{P}_2$ and $\mathbb{Q}_2$. Thus $\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{Q}_2$, $P \equiv_{\mathrm{v}} Q$ and $P \equiv Q$.*

The programs involved in each test (1) differ in $\mathbb{P}_i$ and $\mathbb{Q}_i$ for which the other modules form a common context, say $\mathbb{C}_i$. A way to optimize the verification of $\mathbb{P}_i \sqcup \mathbb{C}_i \equiv_{\mathrm{m}} \mathbb{Q}_i \sqcup \mathbb{C}_i$ is to view $\mathbb{C}_i$ as a module generating input for $\mathbb{P}_i$ and $\mathbb{Q}_i$ and to adjust the method from [15] to use $\mathrm{EQT}(\mathbb{P}_i, \mathbb{Q}_i) \sqcup \mathbb{C}_i$ rather than $\mathrm{EQT}(\mathbb{P}_i \sqcup \mathbb{C}_i, \mathbb{Q}_i \sqcup \mathbb{C}_i)$. We expect computational advantage from this strategy especially when the context $\mathbb{C}_i$ is clearly larger than the modules $\mathbb{P}_i$ and $\mathbb{Q}_i$.

6 RELATED WORK

The notion of modular equivalence is already contrasted with other equivalence relations in Sections 3 and 5.

[3] Recall that $\equiv_{\mathrm{v}}$ coincides with $\equiv$ for programs P and Q having equal and fully visible Herbrand bases.

Bugliesi et al. [2] present an extensive survey of modularity in conventional logic programming. Two mainstream programming disciplines can be identified: *programming-in-the-large* where programs are composed with algebraic operators and *programming-in-the-small* with abstraction mechanisms. Our approach belongs to the former discipline due to resemblance to [10] but stable model semantics and the denial of positive recursion between modules can be pointed out as obvious differences. Eiter et al. [6] present a programming-in-the-small approach to ASP and view program modules as *generalized quantifiers* the definitions of which are allowed to nest, i.e. P can refer to another module Q by using it as a generalized quantifier.

A variety of module conditions (cf. Definition 7) have been introduced. Maher [18] forbids all recursion between modules and considers Przymusinski's *perfect models* rather than stable models. Brogi et al. [1] employ operators for program composition and visibility conditions that correspond to the second item in Definition 7. However, their approach covers only positive programs and the least model semantics. Etalle and Gabbrielli [8] also seek congruence relations but for *constraint logic programs*. Their module architecture is based on a condition similar to ours, i.e. $\mathrm{Hb}(P) \cap \mathrm{Hb}(Q) \subseteq \mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$, but no distinction of input and output is made.

Eiter, Gottlob, and Mannila [5] consider *disjunctive* logic programs as a query language for relational databases. Thus normal programs with variables are covered as a special case. A query program π is instantiated with respect to an input database D confined by an input schema $\mathbf{R}$. The semantics of π determines e.g. the stable models of $\pi[D]$ which are projected with respect to an output schema $\mathbf{S}$. In view of our work, π can be understood as a program module $\mathbb{P}$ with input I and output O based on $\mathbf{R}$ and $\mathbf{S}$, respectively, so that $\pi[D]$ corresponds to $\mathbb{P}(D)$. Their module architecture enables a generalization of splitting sets as both *positive and negative dependencies* are taken into account and no inter-module recursion is tolerated.

Faber et al. [9] apply the *magic set method* in the evaluation of Datalog programs with negation, i.e. effectively normal programs. This involves the concept of an *independent set* S of a program P which is a specialization of a splitting set. The rough idea is that S gives rise to a *module* $T = \{h \leftarrow B^+, {\sim}B^- \in P \mid h \in S\}$ of P so that $T \subseteq P$ and $\mathrm{Head}(T) = S$ and the semantics of S is not affected by $P \setminus T$. Like splitting sets, independent sets are not that flexible in parceling NLPs. E.g., the splitting achieved in Example 2 is impossible. In view of our results, the account of *dangerous rules* unnecessarily pushes negative recursion inside modules. Moreover, the module theorem of Faber et al. [9] is weaker than Theorem 1.

7 CONCLUSION

In this article, we propose a module architecture for answer set programming. The compatibility of the module system and stable models is achieved by denying positive recursion between modules. A number of interesting results is obtained. First, the splitting set theorem [17] is generalized to a case where negative recursion is allowed between modules. Second, the respective equivalence relation $\equiv_{\mathrm{m}}$ is a proper congruence relation for the join operation $\sqcup$ between modules. Third, the verification of modular equivalence can be accomplished with existing methods so that specialized solvers need not be developed. Last but not least, we have a preliminary plan how the task of verifying weak equivalence can be modularized using $\equiv_{\mathrm{m}}$.

Yet the potential gain from the modular verification strategy has to be evaluated by conducting experiments. A further theoretical question is how the existing model theory based on *SE-models* and *UE-models* [3] is tailored to the case of modular equivalence. There is also a need to expand the module architecture and module theorem proposed here to cover other classes of logic programs such as e.g. weight constraint programs, and disjunctive/nested programs.

ACKNOWLEDGEMENTS

This research has been partially funded by the Academy of Finland (project #211025). The first author acknowledges the financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, and Finnish Cultural Foundation.

REFERENCES

[1] A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini, 'Modular logic programming', *ACM Trans. Program. Lang. Syst.*, **16**(4), 1361–1398, (1994).
[2] M. Bugliesi, E. Lamma, and P. Mello, 'Modularity in logic programming', *J. Log. Program.*, **19/20**, 443–502, (1994).
[3] T. Eiter and M. Fink, 'Uniform equivalence of logic programs under the stable model semantics', in *Proc. ICLP'03*, pp. 224–238, (2003).
[4] T. Eiter, M. Fink, H. Tompits, and S. Woltran, 'Simplifying logic programs under uniform and strong equivalence', in *Proc. LPNMR'04*, pp. 87–99, (2004). LNAI 2923.
[5] T. Eiter, G. Gottlob, and H. Mannila, 'Disjunctive datalog.', *ACM Trans. Database Syst.*, **22**(3), 364–418, (1997).
[6] T. Eiter, G. Gottlob, and H. Veith, 'Modular logic programming and generalized quantifiers', in *Proc. LPNMR'97*, pp. 290–309, (1997).
[7] T. Eiter, H. Tompits, and S. Woltran, 'On solution correspondences in answer-set programming.', in *Proc. IJCAI'05*, pp. 97–102, (2005).
[8] S. Etalle and M. Gabbrielli, 'Transformations of CLP modules', *Theor. Comput. Sci.*, **166**(1–2), 101–146, (1996).
[9] W. Faber, G. Greco, and N. Leone, 'Magic sets and their application to data integration.', in *Proc. ICDT'05*, pp. 306–320, (2005). LNCS 3363.
[10] H. Gaifman and E. Shapiro, 'Fully abstract compositional semantics for logic programs', in *Proc. POPL'89*, pp. 134–142, (1989).
[11] M. Gelfond and N. Leone, 'Logic programming and knowledge representation — the A-Prolog perspective', *Artif. Intell.*, **138**, 3–38, (2002).
[12] M. Gelfond and V. Lifschitz, 'The stable model semantics for logic programming', in *Proc. ICLP'88*, pp. 1070–1080, (1988).
[13] T. Janhunen, 'Translatability and intranslatability results for certain classes of logic programs', Research report A82, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, (2003).
[14] T. Janhunen and E. Oikarinen, 'Testing the equivalence of logic programs under stable model semantics', in *Proc. JELIA'02*, pp. 493–504, (2002). LNAI 2424.
[15] T. Janhunen and E. Oikarinen, 'Automated verification of weak equivalence within the SMODELS system', (2005). Submitted to *TPLP*.
[16] V. Lifschitz, D. Pearce, and A. Valverde, 'Strongly equivalent logic programs', *ACM Trans. Comput. Log.*, **2**(4), 526–541, (2001).
[17] V. Lifschitz and H. Turner, 'Splitting a logic program', in *Proc. ICLP'94*, pp. 23–37, (1994).
[18] M. J. Maher, 'A transformation system for deductive database modules with perfect model semantics', *Theor. Comput. Sci.*, **110**(2), 377–403, (1993).
[19] V. W. Marek and M. Truszczyński, 'Stable models and an alternative logic programming paradigm', in *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398, Springer, (1999).
[20] I. Niemelä, 'Logic programming with stable model semantics as a constraint programming paradigm', *Ann. Math. Artif. Intell.*, **25**(3-4), 241–273, (1999).
[21] E. Oikarinen and T. Janhunen, 'Verifying the equivalence of logic programs in the disjunctive case', in *Proc. LPNMR'04*, pp. 180–193, (2004). LNAI 2923.
[22] Y. Sagiv, 'Optimizing datalog programs', in *Proc. PODS'87*, pp. 349–362, (1987).
[23] H. Turner, 'Strong equivalence made easy: nested expressions and weight constraints.', *TPLP*, **3**(4–5), 609–622, (2003).
[24] L. G. Valiant, 'The complexity of enumeration and reliability problems', *SIAM J. Comput.*, **8**(3), 410–421, (1979).
[25] S. Woltran, 'Characterizations for relativized notions of equivalence in answer set programming', in *Proc. JELIA'04*, pp. 161–173, (2004).