Modularity in SMODELS Programs

Emilia Oikarinen

Laboratory for Theoretical Computer Science
P.O.Box 5400, FI-02015 Helsinki University of Technology, Finland
Emilia.Oikarinen@tkk.fi

Abstract. A recently proposed module system for answer set programming is generalized for the input language of the SMODELS system. To show that the stable model semantics is compositional and modular equivalence is a congruence for composition of SMODELS program modules, a general translation-based scheme for introducing syntactic extensions of the module system is presented. A characterization of the compositionality of the semantics is used as an alternative condition for module composition, which allows compositions of modules even in certain cases with positive recursion between the modules to be composed.

1 Introduction

There is a number of approaches within answer set programming (ASP) [1] involving modularity in some sense, based on e.g. *generalized quantifiers* [2], *templates* [3], *import rules* [4], the *splitting set theorem* [5] or its variants [6, 7]. However, only few of these approaches describe a flexible module system with a clearly defined interface for module interaction, and a very typical restriction is that *no recursion between modules is allowed*. In [8] we accommodate Gaifman and Shapiro's program modules [9] to the context of ASP resulting in a simple and intuitive notion for *normal logic program modules* under the *stable model semantics* [10]. A module interacts through an *input/output interface*, and full compatibility of the module system and the stable model semantics is achieved by allowing *positive recursion inside modules only*. However, the use of negative recursion is not limited in any way, and positive recursion is allowed inside modules. One of the main results is a *module theorem* showing that module-level stability implies program-level stability, and vice versa, as long as the stable models of the submodules are *compatible*. We also introduce a *notion of modular equivalence* which is a proper *congruence relation for composition of modules*, i.e., modular equivalence is preserved if a submodule is substituted with a modularly equivalent one.

In this article we extend the module system in [8] for SMODELS *programs* [11] by proposing a general translation-based scheme for introducing syntactical extensions of the module system. Furthermore, we present a semantical reformulation of module composition and modular equivalence, which occasionally allows compositions of modules even if there is positive recursion between modules to be composed.

2 SMODELS Programs and Equivalence Relations

We consider the class of programs in the input language of the SMODELS system [11] excluding *optimization statements*. An SMODELS program P is a finite set of *basic con-*

straint rules and *compute statements*, combined with a *Herbrand base* $\mathrm{Hb}(P)$, which is a fixed finite set of atoms containing all atoms appearing in P. A basic constraint rule is either a *weight rule* of the form $h \leftarrow w \leq \{B = W_B, {\sim}C = W_C\}$ or a *choice rule* of the form $\{H\} \leftarrow B, {\sim}C$ and compute statements are of the form $\mathsf{compute}\{B, {\sim}C\}$, where h is an atom, B, C, and H are sets of atoms, $H \neq \emptyset$, $W_B, W_C \subseteq \mathbb{N}$, and $w \in \mathbb{N}$. In a weight rule each $b \in B$ ($c \in C$) is associated with a weight $w_b \in W_B$ ($w_c \in W_C$). A *basic rule*, denoted by $h \leftarrow B, {\sim}C$, is a special case of a weight rule with all weights equal to 1 and $w = |B| + |C|$. An SMODELS program consisting only of basic rules is called a *normal logic program* (NLP). Basic constraint rules consist of two parts: h or H is the *head* of the rule, and the rest is called the *body*. $\mathrm{Head}(P)$ denotes the set of atoms appearing in the heads of basic constraint rules in an SMODELS program P. An *interpretation* M of a program 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* ($a \notin M$). A choice rule in P is satisfied in all interpretations $M \subseteq \mathrm{Hb}(P)$; a weight rule in P is satisfied in M iff $w \leq \sum_{b \in B \cap M} w_b + \sum_{c \in C \setminus M} w_c$ implies $h \in M$; and a compute statement in P is satisfied in M iff $B \subseteq M$ and $M \cap C = \emptyset$. An interpretation M is a *model* of a program P, denoted by $M \models P$, iff all the rules in P are satisfied in M.

Definition 1. *The reduct P^M of an* SMODELS *program P w.r.t. $M \subseteq \mathrm{Hb}(P)$ contains*

1. *rule $h \leftarrow B$ iff there is a choice rule $\{H\} \leftarrow B, {\sim}C$ in P such that $h \in H \cap M$, and $M \cap C = \emptyset$;*
2. *rule $h \leftarrow w' \leq \{B{=}\, W_B\}$ iff there is a weight rule $h \leftarrow w \leq \{B = W_B, {\sim}C = W_C\}$ in P and $w' = \max(0, \sum_{c \in C \setminus M} w_c)$.*

An SMODELS program P is *positive* if each rule in P is a weight rule with $C = \emptyset$. Given the *least model semantics* for positive programs the stable model semantics [10] straightforwardly generalizes for SMODELS programs [11, 16]. In analogy to the case of NLPs the reduct from Definition 1 is used, but the effect of compute statements must also be taken into account. Let $\mathrm{CompS}(P)$ denote the union of literals appearing in the compute statements of P.

Definition 2. *An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of an* SMODELS *program P, denoted by $M \in \mathrm{SM}(P)$, iff $M = \mathrm{LM}(P^M)$ and $M \models \mathrm{CompS}(P)$.*

Given $a, b \in \mathrm{Hb}(P)$, we say that b *depends directly* on a, denoted by $a \leq_1 b$, iff there is a basic constraint rule in P such that b is in the head of the rule and a appears in the positive body B of the rule. 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* (SCC) of a graph is a maximal subset D of vertices such that there is a path between a and b for all $a, b \in D$.

There are several notions of equivalence proposed for logic programs. Given SMODELS programs P and Q, they are *weakly equivalent* [12], denoted by $P \equiv Q$, iff $\mathrm{SM}(P) = \mathrm{SM}(Q)$, and *strongly equivalent* [12], denoted by $P \equiv_\mathrm{s} Q$, iff $P \cup R \equiv Q \cup R$ for all SMODELS programs R. *Visible equivalence relation* [13] takes the interfaces of programs into account; the Herbrand base of P is partitioned into two parts, $\mathrm{Hb}_\mathrm{v}(P)$ and $\mathrm{Hb}_\mathrm{h}(P)$ determining the *visible* and the *hidden* parts of $\mathrm{Hb}(P)$, respectively. 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 $M \in \mathrm{SM}(P)$, it holds $M \cap \mathrm{Hb_v}(P) = f(M) \cap \mathrm{Hb_v}(Q)$. The verification of $\equiv/\equiv_\mathrm{s}$ is a **coNP**-complete decision problem for SMODELS programs [14, 15]. Deciding $\equiv_\mathrm{v}$ can be hard in general, but the computational complexity can be governed by limiting the use of hidden atoms by the property of having *enough visible atoms*, i.e. the EVA property. Intuitively, if a program has the EVA property, then its stable models can be distinguished on the basis of their visible parts. For SMODELS programs with the EVA property, the verification of visible equivalence is a **coNP**-complete decision problem [16].

3 Modular SMODELS Programs

We define SMODELS *program modules* in analogy to *normal logic program modules* (NLP modules) in [8] and *program modules* by Gaifman and Shapiro [9].

Definition 3. *A triple $\mathbb{P} = (P, I, O)$ is a* SMODELS *program module, if (i) P is a finite set of basic constraint rules and compute statements, and I and O are sets of atoms; (ii) $I \cap O = \emptyset$; and (iii) $\mathrm{Head}(P) \cap I = \emptyset$.*

The Herbrand base of an SMODELS program module $\mathbb{P}$ is the set of atoms appearing in P combined with $I \cup O$, $\mathrm{Hb_v}(\mathbb{P}) = I \cup O$, and $\mathrm{Hb_h}(\mathbb{P}) = \mathrm{Hb}(\mathbb{P}) \setminus \mathrm{Hb_v}(\mathbb{P})$. As noted in [9, 8] *module composition* needs to be restricted in order to achieve *compositionality* for the semantics. In [9] module composition is restricted to cases in which the output sets of the modules are disjoint and the hidden part of each module remains local.

Definition 4. *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be* SMODELS *program modules such that (i) $O_1 \cap O_2 = \emptyset$, and (ii) $\mathrm{Hb_h}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) = \mathrm{Hb_h}(\mathbb{P}_2) \cap \mathrm{Hb}(\mathbb{P}_1) = \emptyset$. Then the* GS-composition *of $\mathbb{P}_1$ and $\mathbb{P}_2$ is defined as*

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

As shown in [8, Example 3] the conditions for $\oplus$ are not enough to guarantee compositionality under the stable model semantics. We say that there is a *positive recursion* between $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$, if there is a SCC in $\mathrm{Dep}^+(P_1 \cup P_2)$ containing atoms from both O_1 and O_2. We deny positive recursion between modules as a further restriction for module composition.

Definition 5. *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be* SMODELS *program modules. If $\mathbb{P}_1 \oplus \mathbb{P}_2$ is defined and there is no positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$, the* join *of $\mathbb{P}_1$ and $\mathbb{P}_2$, denoted by $\mathbb{P}_1 \sqcup \mathbb{P}_2$, is defined as $\mathbb{P}_1 \oplus \mathbb{P}_2$.*

The stable model semantics of an SMODELS program 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 combined with the module. The *instantiation of a module* $\mathbb{P} = (P, I, O)$ with respect to an input $A \subseteq I$ is $\mathbb{P}(A) = \mathbb{P} \sqcup \mathbb{F}_A = (P \cup \mathrm{F}_A, \emptyset, I \cup O)$, where $\mathrm{F}_A = \{a. \mid a \in A\}$. In the sequel $\mathbb{P}(A)$ is identified with the program $P \cup \mathrm{F}_A$.

Definition 6. *An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a stable model of an* SMODELS *program module $\mathbb{P} = (P, I, O)$, denoted by $M \in \mathrm{SM}(\mathbb{P})$, iff $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$ and $M \models \mathrm{CompS}(P)$.*

We generalize the notions of *visible and modular equivalence*, denoted by $\equiv_{\mathrm{v}}$ and $\equiv_{\mathrm{m}}$, respectively, for SMODELS program modules as follows.

Definition 7. *For* SMODELS *program modules* $\mathbb{P}=(P, I_P, O_P)$ *and* $\mathbb{Q}=(Q, I_Q, O_Q)$,

- $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$ *iff* $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$ *and there is a bijection* $f : \mathrm{SM}(\mathbb{P}) \to \mathrm{SM}(\mathbb{Q})$ *such that for all* $M \in \mathrm{SM}(\mathbb{P})$, $M \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = f(M) \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$*; and*
- $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ *iff* $I_P = I_Q$ *and* $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$.

The definition of $\equiv_{\mathrm{m}}$ above is a reformulation of the one given in [8] and results in exactly the same relation. Note that the condition $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$ insisted by $\equiv_{\mathrm{v}}$ together with $I_P = I_Q$ implies $O_P = O_Q$ for $\equiv_{\mathrm{m}}$. Visible equivalence relates more modules than modular equivalence, e.g., consider $\mathbb{P} = (\{a \leftarrow b.\}, \{b\}, \{a\})$ and $\mathbb{Q} = (\{b \leftarrow a.\}, \{a\}, \{b\})$. Now, $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$ as $\mathrm{SM}(\mathbb{P}) = \mathrm{SM}(\mathbb{Q}) = \{\emptyset, \{a, b\}\}$, but $\mathbb{P} \not\equiv_{\mathrm{m}} \mathbb{Q}$ because input interfaces of $\mathbb{P}$ and $\mathbb{Q}$ differ. This indicates that visibly equivalent modules cannot necessarily act as substitutes for each other.

A concept of *compatibility* is used to describe when interpretations of modules can be combined together. Given modules $\mathbb{P}_1$ and $\mathbb{P}_2$ we say that $M_1 \subseteq \mathrm{Hb}(\mathbb{P}_1)$ and $M_2 \subseteq \mathrm{Hb}(\mathbb{P}_2)$ are *compatible* iff $M_1 \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_1)$. Furthermore, given sets of interpretations $A_1 \subseteq 2^{\mathrm{Hb}(\mathbb{P}_1)}$ and $A_2 \subseteq 2^{\mathrm{Hb}(\mathbb{P}_2)}$ for modules $\mathbb{P}_1$ and $\mathbb{P}_2$, the *natural join* of A_1 and A_2, denoted by $A_1 \bowtie A_2$, is defined as

$$\{M_1 \cup M_2 \mid M_1 \in A_1, M_2 \in A_2 \text{ and } M_1 \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_1)\}.$$

If a program (module) consists of several submodules, its stable models are locally stable for the respective submodules; and on the other hand, local stability implies global stability for compatible stable models of the submodules.

Theorem 1. *(Module theorem) Let* $\mathbb{P}_1$ *and* $\mathbb{P}_2$ *be* SMODELS *program modules such that* $\mathbb{P}_1 \sqcup \mathbb{P}_2$ *is defined. Then* $\mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2) = \mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2)$.

Theorem 1 is a generalization of [8, Theorem 1] for SMODELS program modules. Instead of proving Theorem 1 directly from scratch we propose a general translation-based scheme for introducing syntactical extensions for the module theorem.

Proposition 1. *Let* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *be two classes of logic program modules such that* $\mathcal{C}_2 \subseteq \mathcal{C}_1$*, and consider a translation function* $\mathrm{Tr} : \mathcal{C}_1 \to \mathcal{C}_2$ *such that for any program modules* $\mathbb{P} = (P, I, O), \mathbb{Q} \in \mathcal{C}_1$,

1. *if* $\mathbb{P} \sqcup \mathbb{Q}$ *is defined, then* $\mathrm{Tr}(\mathbb{P}) \sqcup \mathrm{Tr}(\mathbb{Q})$ *is defined,*
2. $\mathrm{Tr}(\mathbb{P}) \sqcup \mathrm{Tr}(\mathbb{Q}) \equiv_{\mathrm{m}} \mathrm{Tr}(\mathbb{P} \sqcup \mathbb{Q})$*, and*
3. $(P, I, O \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{P})) \equiv_{\mathrm{m}} (\mathrm{Tr}(P), I, O \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{P}))$.

Now, if the module theorem holds for modules in $\mathcal{C}_2$*, then it holds for modules in* $\mathcal{C}_1$.

The proof is omitted due to space limitations. Intuitively, the conditions for the translation function serve the following purposes: first, possible compositions of modules are not limited by the translation; second, the translation is *modular*; and third, it has to be *faithful* in the sense that it preserves the roles to the atoms in the original module.

Now, to prove Theorem 1 it suffices to provide a translation from SMODELS program modules to NLP modules satisfying the conditions of Proposition 1. For instance, it suffices to take a (possibly exponential) translation similarly to [11]. Furthermore, the *congruence property* of $\equiv_{\mathrm{m}}$ directly generalizes for SMODELS program modules, as Theorem 1 can be used instead of [8, Theorem 1] in the proof of [8, Theorem 2].

Corollary 1. *Let $\mathbb{P}, \mathbb{Q}$ and $\mathbb{R}$ be* SMODELS *program modules such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. If $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, then $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$.*

To analyze the computational complexity of verifying $\equiv_{\mathrm{m}}$ for SMODELS program modules, note first that $P \equiv_{\mathrm{v}} Q$ iff $(P, \emptyset, \mathrm{Hb}_{\mathrm{v}}(P)) \equiv_{\mathrm{m}} (Q, \emptyset, \mathrm{Hb}_{\mathrm{v}}(Q))$ for any SMODELS programs P and Q. Furthermore, given SMODELS program modules $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$, $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$, where $\mathbb{G}_I = (\{\{I\}.\}, \emptyset, I)$ a context generator module for I. Thus the problem of verifying $\equiv_{\mathrm{m}}$ has the same computational complexity as verification of $\equiv_{\mathrm{v}}$. We say that an SMODELS program module $\mathbb{P} = (P, I, O)$ has the EVA property iff P has the EVA property for $\mathrm{Hb}_{\mathrm{v}}(P) = I \cup O$ [16]. Verification of $\equiv_{\mathrm{m}}$ is a **coNP**-complete decision problem for SMODELS program modules with the EVA property, since $\mathbb{G}_I$ has the EVA property trivially.

4 Semantical Reformulation of Modular Equivalence

Even though [8, Example 3] shows that conditions for $\oplus$ are not enough to guarantee that the module theorem holds, there are cases where $\mathbb{P} \sqcup \mathbb{Q}$ is not defined and $\mathrm{SM}(\mathbb{P} \oplus \mathbb{Q}) = \mathrm{SM}(\mathbb{P}) \bowtie \mathrm{SM}(\mathbb{Q})$. Consider, e.g., $\mathbb{P} = (\{a \leftarrow b.\ a \leftarrow {\sim}c.\}, \{b\}, \{a, c\})$ and $\mathbb{Q} = (\{b \leftarrow a.\}, \{a\}, \{b\})$. Now, $\mathbb{P} \oplus \mathbb{Q} = (\{a \leftarrow b.\ a \leftarrow {\sim}c.\ b \leftarrow a.\}, \emptyset, \{a, b, c\})$ is defined as outputs and hidden atoms differ. Since $\mathrm{SM}(\mathbb{P}) = \{\{a\}, \{a, b\}\}$ and $\mathrm{SM}(\mathbb{Q}) = \{\emptyset, \{a, b\}\}$, we get $\mathrm{SM}(\mathbb{P} \oplus \mathbb{Q}) = \mathrm{SM}(\mathbb{P}) \bowtie \mathrm{SM}(\mathbb{Q}) = \{\{a, b\}\}$. This suggests that the denial of positive recursion between modules can be relaxed in certain cases.

We define a semantical characterization for module composition that maintains the compositionality of the semantics.

Definition 8. *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be* SMODELS *program modules such that $\mathbb{P}_1 \oplus \mathbb{P}_2$ is defined and $\mathrm{SM}(\mathbb{P}_1 \oplus \mathbb{P}_2) = \mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2)$. Then the semantical join of $\mathbb{P}_1$ and $\mathbb{P}_2$, denoted by $\mathbb{P}_1 \underline{\sqcup} \mathbb{P}_2$, is defined as $\mathbb{P}_1 \oplus \mathbb{P}_2$.*

The module theorem holds by definition for SMODELS program modules composed with $\underline{\sqcup}$. We present an alternative formulation for modular equivalence taking features from strong equivalence [12]: $\mathbb{P} = (P, I_P, O_P)$ and $\mathbb{Q} = (Q, I_Q, O_Q)$ are *semantically modularly equivalent*, denoted by $\mathbb{P} \equiv_{\mathrm{sem}} \mathbb{Q}$, iff $I_P = I_Q$ and $\mathbb{P} \underline{\sqcup} \mathbb{R} \equiv_{\mathrm{v}} \mathbb{Q} \underline{\sqcup} \mathbb{R}$ for all modules $\mathbb{R}$ such that $\mathbb{P} \underline{\sqcup} \mathbb{R}$ and $\mathbb{Q} \underline{\sqcup} \mathbb{R}$ are defined. It is straightforward to see that $\equiv_{\mathrm{sem}}$ is a congruence for $\underline{\sqcup}$ and reduces to $\equiv_{\mathrm{v}}$ for modules with a completely specified input.

Theorem 2. *$\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \equiv_{\mathrm{sem}} \mathbb{Q}$ for any* SMODELS *program modules $\mathbb{P}$ and $\mathbb{Q}$.*

Theorem 2 implies that $\equiv_{\mathrm{m}}$ is a congruence for $\underline{\sqcup}$, too, and it is possible to replace $\mathbb{P}$ with modularly equivalent $\mathbb{Q}$ in contexts allowed by $\underline{\sqcup}$. The syntactical restriction denying positive recursion between modules is easy to check, since SCCs can be found in a linear time with respect to the size of the dependency graph [17]. Checking whether $\mathrm{SM}(\mathbb{P}_1 \oplus \mathbb{P}_2) = \mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2)$ holds can be a computationally much harder.

Theorem 3. *For* SMODELS *program modules* $\mathbb{P}_1$ *and* $\mathbb{P}_2$ *such that* $\mathbb{P}_1 \oplus \mathbb{P}_2$ *is defined, deciding whether* $\mathrm{SM}(\mathbb{P}_1 \oplus \mathbb{P}_2) = \mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2)$ *holds is a* $\mathbf{coNP}$*-complete decision problem.*

Theorem 3 shows that there is a tradeoff for allowing positive recursion between modules, as more effort is needed to check that composition of such modules does not compromise the compositionality of the semantics.

Acknowledgements The author is grateful to Tomi Janhunen for his helpful comments and suggestions. The research is partially funded by Academy of Finland (project #211025). The financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, Finnish Foundation of Technology, and Finnish Cultural Foundation is gratefully acknowledged.

References

1. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3-4) (1999) 241–273
2. Eiter, T., Gottlob, G., Veith, H.: Modular logic programming and generalized quantifiers. In: LPNMR, Volume 1265 of LNCS, Springer (1997) 290–309
3. Ianni, G., Ielpa, G., Pietramala, A., Santoro, M.C., Calimeri, F.: Enhancing answer set programming with templates. In: NMR (2004) 233–239
4. Tari, L., Baral, C., Anwar, S.: A language for modular answer set programming: Application to ACC tournament scheduling. In: ASP, CEUR-WS.org (2005) 277–292
5. Lifschitz, V., Turner, H.: Splitting a logic program. In: ICLP, MIT Press (1994) 23–37
6. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM TODS **22**(3) (1997) 364–418
7. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. In: ICDT, Volume 3363 of LNCS, Springer (2005) 306–320
8. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: ECAI, IOS Press (2006) 412–416
9. Gaifman, H., Shapiro, E.Y.: Fully abstract compositional semantics for logic programs. In: POPL, ACM Press (1989) 134–142
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP, MIT Press (1988) 1070–1080
11. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
12. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM TOCL **2**(4) (2001) 526–541
13. Janhunen, T.: Some (In)translatability Results for Normal Logic Programs and Propositional Theories. JANCL **16**(1-2) (2006) 35–86
14. Marek, V.W., Truszczyński, M.: Autoepistemic logic. J. ACM **38**(3) (1991) 588–619
15. Pearce, D., Tompits, H., Woltran, S.: Encodings for equilibrium logic and logic programs with nested expressions. In: EPIA, Volume 2258 of LNCS, Springer (2001) 306–320
16. Janhunen, T., Oikarinen, E.: Automated verification of weak equivalence within the SMODELS system. TPLP **7**(4) (2007) 1–48
17. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comp. **1**(2) (1972) 146–160