Modularity Aspects of Disjunctive Stable Models*

Tomi Janhunen[1], Emilia Oikarinen[1], Hans Tompits[2], and Stefan Woltran[2]

[1] Helsinki University of Technology
Department of Computer Science and Engineering
P.O. Box 5400, FI-02015 TKK, Finland
{Tomi.Janhunen,Emilia.Oikarinen}@tkk.fi
[2] Technische Universität Wien
Institut für Informationssysteme, 184/3
Favoritenstraße 9–11, A-1040 Vienna, Austria
{tompits,stefan}@kr.tuwien.ac.at

Abstract. Practically all programming languages used in software engineering allow to split a program into several modules. For fully declarative and nonmonotonic logic programming languages, however, the modular structure of programs is hard to realise, since the output of an entire program cannot in general be composed from the output of its component programs in a direct manner. In this paper, we consider these aspects for the stable-model semantics of disjunctive logic programs (DLPs). We define the notion of a *DLP-function*, where a well-defined input/output interface is provided, and establish a novel module theorem enabling a suitable compositional semantics for modules. The module theorem extends the well-known splitting-set theorem and allows also a generalisation of a shifting technique for splitting shared disjunctive rules among components.

1 Introduction

Practically all programming languages used in software engineering allow the user to split a program into several modules, which are composed by well-defined semantics over the modules' input/output interface. This not only helps towards a good programming style, but admits also to delegate coding tasks among several programmers, which then realise the specified input/output behaviour in terms of concrete modules.

The paradigm of *answer-set programming* (ASP), and in particular the case of *disjunctive logic programs* (DLPs) under the *stable-model semantics* [1], which we deal with herein, requires a fully declarative nonmonotonic semantics which is defined only over complete programs and therefore *prima facie* not directly applicable to modular programming. Due to this obstacle, the concept of a module has not raised much attention yet in nonmonotonic logic programming, and, except for a few dedicated papers [2,3,4], modules mostly appeared as a by-product in investigations of formal properties like stratification, splitting, or, more recently, in work on equivalence between

* This work was partially supported by the Academy of Finland under project #211025 ("Advanced Constraint Programming Techniques for Large Structured Problems") and by the Austrian Science Foundation (FWF) under project P18019-N04 ("Formal Methods for Comparing and Optimizing Nonmonotonic Logic Programs").

C. Baral, G. Brewka, and J. Schlipf (Eds.): LPNMR 2007, LNAI 4483, pp. 175–187, 2007.

programs [5,6,7,8]. The approach by Oikarinen and Janhunen [8] accommodates the module architecture discussed by Gaifman and Shapiro [2] for non-disjunctive programs and establishes a *module theorem* for stable models. This result indicates that the compositionality of stable models can be achieved in practice if positively interdependent atoms are not scattered among several modules.

In this paper, we deal with the formal underpinnings for modular programming in the context of disjunctive logic programs under the stable-model semantics. To begin with, we introduce the notion of a *DLP-function* which can roughly be described as a disjunctive logic program together with a well-defined input/output interface providing *input atoms*, *output atoms*, and *hidden* (*local*) *atoms*. In that, we follow Gelfond [9] who introduced *lp-functions* for specifying (partial) definitions of new relations in terms of old, known ones. This functional view of programs is also apparent if the latter are understood as *queries* over an input (i.e., a database). Indeed, several authors (like, e.g., Eiter, Gottlob, and Mannila [6]) introduce logic programs as an extension of Datalog, which poses no major problems with respect to stable semantics as long as a single program with a specified input/output behaviour is considered.

The latter point leads us to the second main issue addressed in our framework, viz. the question of a (semantically meaningful) method for the *composition* of modules. If the underlying semantics is inherently nonmonotonic, as is the case for stable semantics, this generates several problems, which were first studied in detail by Gaifman and Shapiro [2] for logic programs (without default negation) under minimal Herbrand models. As they observed, it is necessary to put certain syntactical restrictions on the programs to be composed, in order to make the semantics act accordingly. We shall follow their approach closely but extend it to programs permitting both default negation and disjunction. Notably, the problem of compositional semantics also arises in relation to the so-called *splitting-set theorem* [5,6,7], which aims at computing the stable models of a composed program by a suitable combination of the models of two programs which result from the split of the entire program.

After having the basic syntactical issues of DLP-functions laid out, we then define their model theory in terms of a generalisation of the stable-model semantics, where particular care is taken regarding the input of a DLP-function. The adequacy of our endeavour is witnessed by the main result of our paper, viz. the *module theorem*, providing the foundation for a fully compositional semantics: It shows how stable models of entire programs can be composed by joining together compatible stable models of their respective component programs. We round off our results with some applications of the module theorem. First, the module theorem readily extends the splitting-set theorem [5,6]. Second, it leads to a general *shifting principle* that can be used to simplify programs, i.e., to split shared disjunctive rules among components. Third, it gives rise to a notion of modular equivalence for DLP-functions that turns out to be a proper congruence relation supporting program substitutions.

2 The Class $\mathcal{D}$ of DLP-Functions

A *disjunctive rule* is an expression of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, {\sim}c_1, \ldots, {\sim}c_k, \tag{1}$$

where $n, m, k \geq 0$, and $a_1, \ldots, a_n$, $b_1, \ldots, b_m$, and $c_1, \ldots, c_k$ are propositional atoms. Since the order of atoms is considered insignificant, we write $A \leftarrow B, \sim C$ as a shorthand for rules of form (1), where $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$, and $C = \{c_1, \ldots, c_k\}$. The basic intuition behind (1) is that if each atom in the positive body B can be inferred and none of the atoms in the negative body C, then some atom in the head A can be inferred. When both B and C are empty, we have a *disjunctive fact*, written $A \leftarrow$. If A is empty, then we have a *constraint*, written $\bot \leftarrow B, \sim C$.

A *disjunctive logic program* (DLP) is conventionally formed as a set of disjunctive rules. Additionally, we want a distinguished *input and output interface* for each DLP. To this end, we extend a definition originally proposed by Gaifman and Shapiro [2] to the case of disjunctive programs.[1] Given a set R of disjunctive rules, we write $\mathrm{At}(R)$ for the *signature* of R, i.e., the set of (ground) atoms appearing in the rules of R. The set $\mathrm{Head}(R)$ consists of those elements of $\mathrm{At}(R)$ having head occurrences in R. This is exactly the set of atoms which are *defined* by the rules of R.

Definition 1. *A* DLP-function*, Π, is a quadruple $\langle R, I, O, H\rangle$, where I, O, and H are pairwise distinct sets of atoms, and R is a set of disjunctive rules such that*

$$\mathrm{At}(R) \subseteq I \cup O \cup H \textit{ and } \mathrm{Head}(R) \subseteq O \cup H.$$

The elements of I are called input atoms*, the elements of O* output atoms*, and the elements of H* hidden atoms*.*

Given a DLP-function $\Pi = \langle R, I, O, H\rangle$, we write, with a slight abuse of notation, $A \leftarrow B, \sim C \in \Pi$ to denote that the rule $A \leftarrow B, \sim C$ is contained in the set R. The atoms in $I \cup O$ are considered to be *visible* and hence accessible to other DLP-functions conjoined with Π; either to produce input for Π or to utilise the output of Π. On the other hand, the *hidden* atoms in H are used to formalise some auxiliary concepts of Π which may not make sense in the context of other DLP-functions but may save space substantially (see, e.g., Example 4.5 of Janhunen and Oikarinen [11]). The condition $\mathrm{Head}(R) \subseteq O \cup H$ ensures that a DLP-function may not interfere with its own input by defining input atoms of I in terms of its rules. In spite of this, the rules of Π may be conditioned by input atoms appearing in the bodies of rules. Following previous ideas [9,8], we define the signature $\mathrm{At}(\Pi)$ of a DLP-function $\Pi = \langle R, I, O, H\rangle$ as $I \cup O \cup H$.[2] For notational convenience, we distinguish the *visible* and *hidden parts* of $\mathrm{At}(\Pi)$ by setting $\mathrm{At}_{\mathrm{v}}(\Pi) = I \cup O$ and $\mathrm{At}_{\mathrm{h}}(\Pi) = H = \mathrm{At}(\Pi) \setminus \mathrm{At}_{\mathrm{v}}(\Pi)$, respectively. Additionally, $\mathrm{At}_{\mathrm{i}}(\Pi)$ and $\mathrm{At}_{\mathrm{o}}(\Pi)$ provide us a way of referring to the sets I and O of input and output atoms of Π, respectively. Lastly, for any set $S \subseteq \mathrm{At}(\Pi)$ of atoms, we denote the projections of S on $\mathrm{At}_{\mathrm{i}}(\Pi)$, $\mathrm{At}_{\mathrm{o}}(\Pi)$, $\mathrm{At}_{\mathrm{v}}(\Pi)$, and $\mathrm{At}_{\mathrm{h}}(\Pi)$ by S_{i}, S_{o}, S_{v}, and S_{h}, respectively.

In formal terms, a DLP-function $\Pi = \langle R, I, O, H\rangle$ provides a mapping from subsets of I to a set of subsets of $O \cup H$ in analogy to the method by Gelfond [9]. However, the exact definition of this mapping is deferred until Section 3 where the semantics of DLP-functions will be anchored. In the sequel, the (syntactic) class of DLP-functions is

[1] There are already similar approaches within the area of ASP [9,10,11,8].

[2] Consequently, the *length* of Π in symbols, denoted by $\|\Pi\|$, gives an upper bound for $|\mathrm{At}(\Pi)|$ which is important when one considers the computational cost of translating programs [10].

denoted by $\mathcal{D}$. It is assumed for the sake of simplicity that $\mathcal{D}$ spans over a fixed (at most denumerable) signature $\mathrm{At}(\mathcal{D})$[3] so that $\mathrm{At}(\Pi) \subseteq \mathrm{At}(\mathcal{D})$ holds for each DLP-function $\Pi \in \mathcal{D}$.

The composition of DLP-functions takes place as set out in Definition 2 below. We say that a DLP-function Π_1 *respects the hidden atoms* of another DLP-function Π_2 iff $\mathrm{At}(\Pi_1) \cap \mathrm{At_h}(\Pi_2) = \emptyset$, i.e., Π_1 does not use any atoms from $\mathrm{At_h}(\Pi_2)$.

Definition 2 (Gaifman and Shapiro [2]). *The* composition *of two DLP-functions Π_1 and Π_2 that* respect the hidden atoms of each other *is the DLP-function*

$$\Pi_1 \oplus \Pi_2 = \langle R_1 \cup R_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2, H_1 \cup H_2 \rangle. \tag{2}$$

As discussed by Gaifman and Shapiro [2], program composition can be generalised for pairs of programs not respecting each other's hidden atoms. The treatment of atom types under Definition 2 is summarised in Table 1 where the intersections of the sets of the input, output, and hidden atoms of Π_1 and Π_2 are represented by the cells in the respective intersections of rows and columns (e.g., an atom $a \in O_1 \cap I_2$ becomes an output atom in $\Pi_1 \oplus \Pi_2$). Given that hidden atoms are mutually respected, ten cases arise in all. The consequences of Definition 2 should be intuitive to readers acquainted with the principles of object-oriented programming: (i) Although Π_1 and Π_2 must not share hidden atoms, they may share input atoms, i.e., $I_1 \cap I_2 \neq \emptyset$ is allowed. For now, the same can be stated about output atoms but this will be excluded by further conditions as done by Gaifman and Shapiro [2], where $O_1 \cap O_2 = \emptyset$ is assumed directly. (ii) An input atom of Π_1 becomes an output atom in $\Pi_1 \oplus \Pi_2$ if it appears as an output atom in Π_2, i.e., Π_2 provides the input for Π_1 in this setting. The input atoms of Π_2 are treated in a symmetric fashion. (iii) The hidden atoms of Π_1 and Π_2 retain their status in $\Pi_1 \oplus \Pi_2$.

Table 1. Division of atoms under $\oplus$ into input (i), output (o), or hidden (h) atoms

$\oplus$		I_2	O_2	H_2
		i	o	h
I_1	i	i	o	-
O_1	o	o	o	-
H_1	h	-	-	-

Given DLP-functions Π_1, Π_2, and Π_3 that pairwise respect the hidden atoms of each other, it holds that $\Pi_1 \oplus \Pi_2 \in \mathcal{D}$ (closure), $\Pi_1 \oplus \varnothing = \varnothing \oplus \Pi_1 = \Pi_1$ for the empty DLP-function $\varnothing = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$ (identity), $\Pi_1 \oplus \Pi_2 = \Pi_2 \oplus \Pi_1$ (commutativity), and $\Pi_1 \oplus (\Pi_2 \oplus \Pi_3) = (\Pi_1 \oplus \Pi_2) \oplus \Pi_3$ (associativity). However, the notion of *modular equivalence* [8] is based on a more restrictive operator for program composition. The basic idea is to forbid positive dependencies between programs. Technically speaking, we define the *positive dependency graph* $\mathrm{DG}^+(\Pi) = \langle \mathrm{At}(\Pi), \leq_1 \rangle$ for each DLP-function Π in the standard way [12] using only positive dependencies: an atom $a \in A$ in the head of a rule $A \leftarrow B, {\sim}C \in \Pi$ depends positively on each $b \in B$ and each pair $\langle b, a \rangle$ belongs to the edge relation $\leq_1$ in $\mathrm{DG}^+(\Pi)$, i.e., $b \leq_1 a$ holds. The reflexive and transitive closure of $\leq_1$ gives rise to the dependency relation $\leq$ over $\mathrm{At}(\Pi)$.

A *strongly connected component* (SCC) S of $\mathrm{DG}^+(\Pi)$ is a maximal set $S \subseteq \mathrm{At}(\Pi)$ such that $b \leq a$ holds for every $a, b \in S$. Given that $\Pi_1 \oplus \Pi_2$ is defined, we say that Π_1 and Π_2 are *mutually dependent* iff $\mathrm{DG}^+(\Pi_1 \oplus \Pi_2)$ has a SCC S shared by Π_1 and Π_2 such that $S \cap \mathrm{At_o}(\Pi_1) \neq \emptyset$ and $S \cap \mathrm{At_o}(\Pi_2) \neq \emptyset$ [8].

[3] In practice, this set could be the set of all identifiers (names for propositions or similar objects).

Definition 3. *The* join*, $\Pi_1 \sqcup \Pi_2$, of two DLP-functions Π_1 and Π_2 is $\Pi_1 \oplus \Pi_2$, providing $\Pi_1 \oplus \Pi_2$ is defined and Π_1 and Π_2 are not mutually dependent.*

It is worth pointing out that $\mathrm{At_o}(\Pi_1) \cap \mathrm{At_o}(\Pi_2) = \emptyset$ follows in analogy to Gaifman and Shapiro [2] when $\Pi_1 \sqcup \Pi_2$ is defined. At first glance, this may appear rather restrictive, e.g., the head of a disjunctive rule $A \leftarrow B, {\sim}C$ cannot be shared by modules: either $A \subseteq \mathrm{At_o}(\Pi_1)$ or $A \subseteq \mathrm{At_o}(\Pi_2)$ must hold but not both. The general shifting technique to be presented in Section 5 allows us to circumvent this problem by viewing shared rules as syntactic sugar. Moreover, since Π_1 and Π_2 are not mutually dependent in $\Pi_1 \sqcup \Pi_2$, we have (i) $S \subseteq \mathrm{At_i}(\Pi_1 \sqcup \Pi_2)$, (ii) $S \subseteq \mathrm{At_o}(\Pi_1) \cup \mathrm{At_h}(\Pi_1)$, or (iii) $S \subseteq \mathrm{At_o}(\Pi_2) \cup \mathrm{At_h}(\Pi_2)$, for each SCC S of $\mathrm{DG}^+(\Pi_1 \oplus \Pi_2)$. The first covers joint input atoms $a \in \mathrm{At_i}(\Pi_1 \sqcup \Pi_2)$ which do not depend on other atoms by definition and which end up in singleton SCCs $\{a\}$.

The dependency relation $\leq$ lifts to the level of SCCs as follows: $S_1 \leq S_2$ iff there are $a_1 \in S_1$ and $a_2 \in S_2$ such that $a_1 \leq a_2$. In the sequel, a total order $S_1 < \cdots < S_k$ of the strongly connected components in $\mathrm{DG}^+(\Pi_1 \oplus \Pi_2)$ is also employed. Such an order $<$ is guaranteed to exist but it is not necessarily unique. E.g., the relative order of S_2 and S_3 can be freely chosen given that $S_1 \leq S_2 \leq S_4$ and $S_1 \leq S_3 \leq S_4$ hold for four components under $\leq$. Nevertheless $<$ is consistent with $\leq$, i.e., $S_i < S_j$ implies $S_j \not\leq S_i$ but either $S_i \leq S_j$ or $S_i \not\leq S_j$ may hold depending on $\leq$. Given that $\Pi_1 \sqcup \Pi_2$ is defined, we may project $S_1 < \cdots < S_k$ for Π_1 and Π_2 as follows. In case of Π_1, for instance, $S_{1,i} = S_i$, if $S_i \subseteq \mathrm{At_o}(\Pi_1) \cup \mathrm{At_h}(\Pi_1)$ or $S_i \subseteq \mathrm{At_i}(\Pi_1 \sqcup \Pi_2)$, and $S_{1,i} = S_i \cap \mathrm{At_i}(\Pi_1)$, if $S_i \subseteq \mathrm{At_o}(\Pi_2) \cup \mathrm{At_h}(\Pi_2)$. In the latter case, it is possible that $S_{1,i} = \emptyset$ or $S_{1,i}$ contains several input atoms of Π_1 which are independent of each other. This violates the definition of a SCC but we do not remove or split such exceptional components—also called SCCs in the sequel—to retain a uniform indexing scheme for the components of Π, Π_1, and Π_2. Thus, we have established the respective component structures $S_{1,1} < \cdots < S_{1,k}$ and $S_{2,1} < \cdots < S_{2,k}$ for Π_1 and Π_2.

3 Model Theory and Stable Semantics

Given any DLP-function Π, by an *interpretation*, M, for Π we understand a subset of $\mathrm{At}(\Pi)$. An atom $a \in \mathrm{At}(\Pi)$ is *true* under M (symbolically $M \models a$) iff $a \in M$, otherwise *false* under M. For a negative literal ${\sim}a$, we define $M \models {\sim}a$ iff $M \not\models a$. A set L of literals is satisfied by M (denoted by $M \models L$) iff $M \models l$, for every $l \in L$. We also define $M \models \bigvee L$, providing $M \models l$ for some $l \in L$.

To begin with, we cover DLP-functions with a pure classical semantics, which treats disjunctive rules as classical implications.

Definition 4. *An interpretation $M \subseteq \mathrm{At}(\Pi)$ is a* (classical) model *of a DLP-function $\Pi = \langle R, I, O, H \rangle$, denoted $M \models \Pi$, iff $M \models R$, i.e., for every rule $A \leftarrow B, {\sim}C \in R$,*

$$M \models B \cup {\sim}C \text{ implies } M \models \bigvee A.$$

The set of all classical models of Π is denoted by $\mathrm{CM}(\Pi)$.

Classical models provide a suitable level of abstraction to address the role of input atoms in DLP-functions. Given a DLP-function Π and an interpretation $M \subseteq \mathrm{At}(\Pi)$,

the projection M_{i} can be viewed as the actual input for Π which may (or may not) produce the respective output M_{o}, depending on the semantics assigned to Π. The treatment of input atoms in the sequel will be based on *partial evaluation*: the idea is to pre-interpret input atoms appearing in Π with respect to M_{i}.

Definition 5. *For a DLP-function $\Pi = \langle R, I, O, H\rangle$ and an actual input $M_{\mathrm{i}} \subseteq I$ for Π, the* instantiation of Π with respect to M_{i}*, denoted by Π/M_{i}, is the quadruple $\langle R', \emptyset, I \cup O, H\rangle$ where R' consists of the following rules:*

1. *the rule $A \leftarrow (B \setminus I), {\sim}(C \setminus I)$, for each rule $A \leftarrow B, {\sim}C \in \Pi$ such that $M_{\mathrm{i}} \models B_{\mathrm{i}} \cup {\sim}C_{\mathrm{i}}$,*
2. *the fact $a \leftarrow$, for each atom $a \in M_{\mathrm{i}}$, and*
3. *the constraint $\bot \leftarrow a$, for each atom $a \in I \setminus M_{\mathrm{i}}$.*

The rules in the first item are free of input atoms since $A \cap I = \emptyset$ holds for each rule $A \leftarrow B, {\sim}C$ in R by Definition 1. The latter two items list rules that record the truth values of input atoms of Π in the resulting program Π/M_{i}. The reduct Π/M_{i} is a DLP-function without input whereas the visibility of atoms is not affected by instantiation.

Proposition 1. *Let Π be a DLP-function and $M \subseteq \mathrm{At}(\Pi)$ an interpretation that defines an actual input $M_{\mathrm{i}} \subseteq \mathrm{At}_{\mathrm{i}}(\Pi)$ for Π. Then, for all interpretations $N \subseteq \mathrm{At}(\Pi)$,*

$$N \models \Pi \text{ and } N_{\mathrm{i}} = M_{\mathrm{i}} \iff N \models \Pi/M_{\mathrm{i}}.$$

Thus, the input reduction, as given in Definition 5, is fully compatible with classical semantics and we may characterise the semantic operator CM by pointing out the fact that $\mathrm{CM}(\Pi) = \bigcup_{M_{\mathrm{i}} \subseteq I} \mathrm{CM}(\Pi/M_{\mathrm{i}})$. Handling input is slightly more complicated in the case of minimal models but Lifschitz's *parallel circumscription* [13] provides us a standard approach to deal with it. The rough idea is to keep the interpretation of input atoms fixed while minimising, i.e., falsifying others as far as possible.

Definition 6. *Let Π be a DLP-function and $F \subseteq \mathrm{At}(\Pi)$ a set of atoms assumed to have fixed truth values. A model $M \subseteq \mathrm{At}(\Pi)$ of Π is* F-minimal *iff there is no model N of Π such that $N \cap F = M \cap F$ and $N \subset M$.*

The set of F-minimal models of Π is denoted by $\mathrm{MM}_F(\Pi)$. In the sequel, we treat input atoms by stipulating $\mathrm{At}_{\mathrm{i}}(\Pi)$-minimality of models of Π. Then, the condition $N \cap F = M \cap F$ in Definition 6 becomes equivalent to $N_{\mathrm{i}} = M_{\mathrm{i}}$. Using this idea, Proposition 1 lifts for minimal models as follows. Recall that $\mathrm{At}_{\mathrm{i}}(\Pi/M_{\mathrm{i}}) = \emptyset$.

Proposition 2. *Let Π be a DLP-function and $M \subseteq \mathrm{At}(\Pi)$ an interpretation that defines an actual input $M_{\mathrm{i}} \subseteq \mathrm{At}_{\mathrm{i}}(\Pi)$ for Π. Then, for all interpretations $N \subseteq \mathrm{At}(\Pi)$,*

$$N \in \mathrm{MM}_{\mathrm{At}_{\mathrm{i}}(\Pi)}(\Pi) \text{ and } N_{\mathrm{i}} = M_{\mathrm{i}} \iff N \in \mathrm{MM}_{\emptyset}(\Pi/M_{\mathrm{i}}).$$

The set $\mathrm{MM}_{\mathrm{At}_{\mathrm{i}}(\Pi)}(\Pi)$ of models is sufficient to determine the semantics of a *positive* DLP-function, i.e., whose rules are of the form $A \leftarrow B$ where $A \neq \emptyset$ and only B may involve atoms from $\mathrm{At}_{\mathrm{i}}(\Pi)$. Therefore, due to non-empty heads of rules, a positive DLP Π is guaranteed to possess classical models since, e.g., $\mathrm{At}(\Pi) \models \Pi$, and thus also $\mathrm{At}_{\mathrm{i}}(\Pi)$-minimal models. To cover arbitrary DLP-functions, we interpret negative body literals in the way proposed by Gelfond and Lifschitz [1].

Definition 7. *Given a DLP-function* $\Pi = \langle R, I, O, H\rangle$ *and an interpretation* $M \subseteq \mathrm{At}(\Pi)$*, the* Gelfond-Lifschitz reduct of Π with respect to M *is the positive DLP-function*

$$\Pi^M = \langle \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in \Pi,\ A \neq \emptyset, \text{ and } M \models {\sim}C\},\ I,\ O,\ H\rangle. \quad (3)$$

Definition 8. *An interpretation* $M \subseteq \mathrm{At}(\Pi)$ *is a* stable model *of a DLP-function* Π *iff* $M \in \mathrm{MM}_{\mathrm{At_i}(\Pi)}(\Pi^M)$ *and* $M \models \mathrm{CR}(\Pi)$*, where* $\mathrm{CR}(\Pi)$ *is the set of constraints* $\bot \leftarrow B, {\sim}C \in \Pi$.

Hidden atoms play no special role in Definition 8 and their status will be clarified in Section 5 when the notion of *modular equivalence* is introduced. Definition 8 gives rise to the respective semantic operator $\mathrm{SM} : \mathcal{D} \rightarrow \mathbf{2}^{\mathbf{2}^{\mathrm{At}(\mathcal{D})}}$ for DLP-functions:

$$\mathrm{SM}(\Pi) = \{M \subseteq \mathrm{At}(\Pi) \mid M \in \mathrm{MM}_{\mathrm{At_i}(\Pi)}(\Pi^M) \text{ and } M \models \mathrm{CR}(\Pi)\}. \quad (4)$$

As a consequence of Proposition 2, a stable model M of Π is a minimal model of $\Pi^M/M_\mathrm{i} = (\Pi/M_\mathrm{i})^M$ which enables one to dismiss $\mathrm{At_i}(\Pi)$-minimality if desirable.

Example 1. Consider a DLP-function

$$\Pi = \langle \{a \vee b \leftarrow {\sim}c;\ a \leftarrow c, {\sim}b;\ b \leftarrow c, {\sim}a\}, \{c\}, \{a, b\}, \emptyset\rangle,$$

which has four stable models, $M_1 = \{a\}$, $M_2 = \{b\}$, $M_3 = \{a, c\}$, and $M_4 = \{b, c\}$, which are minimal models of the respective reducts of Π:

$$\begin{aligned}
&\Pi^{M_1}/(M_1)_\mathrm{i} = \Pi^{M_2}/(M_2)_\mathrm{i} = \langle \{a \vee b \leftarrow;\ \bot \leftarrow c\}, \emptyset, \{a, b, c\}, \emptyset\rangle,\\
&\Pi^{M_3}/(M_3)_\mathrm{i} = \langle \{a \leftarrow;\ c \leftarrow\}, \emptyset, \{a, b, c\}, \emptyset\rangle, \text{ and}\\
&\Pi^{M_4}/(M_4)_\mathrm{i} = \langle \{b \leftarrow;\ c \leftarrow\}, \emptyset, \{a, b, c\}, \emptyset\rangle.
\end{aligned}$$

□

An immediate observation is that we loose the general antichain property of stable models when input signatures are introduced. For instance, we have $M_1 \subset M_3$ and $M_2 \subset M_4$ in Example 1. However, since the interpretation of input atoms is fixed by the semantics, we perceive antichains *locally*, i.e., the set of stable models $\{N \in \mathrm{SM}(\Pi) \mid N_\mathrm{i} = M_\mathrm{i}\}$ forms an antichain, for each input $M_\mathrm{i} \subseteq \mathrm{At_i}(\Pi)$. In Example 1, the sets associated with actual inputs $\emptyset$ and $\{c\}$ are $\{M_1, M_2\}$ and $\{M_3, M_4\}$, respectively.

4 Module Theorem for DLP-Functions

Our next objective is to show that stable semantics allows substitutions under joins of programs as defined in Section 2. Given two DLP-functions Π_1 and Π_2, we say that interpretations $M_1 \subseteq \mathrm{At}(\Pi_1)$ and $M_2 \subseteq \mathrm{At}(\Pi_2)$ are *mutually compatible* (*with respect to* Π_1 *and* Π_2), or just *compatible* for short, iff $M_1 \cap \mathrm{At_v}(\Pi_1) = M_2 \cap \mathrm{At_v}(\Pi_2)$, i.e., M_1 and M_2 agree about the truth values of their joint visible atoms. A quick inspection of Table 1 reveals the three cases that may arise if the join $\Pi = \Pi_1 \sqcup \Pi_2$ is defined and joint *output atoms* for Π_1 and Π_2 are disallowed: There are shared input atoms in $\mathrm{At_i}(\Pi) = \mathrm{At_i}(\Pi_1) \cap \mathrm{At_i}(\Pi_2)$ and atoms in $\mathrm{At_o}(\Pi_1) \cap \mathrm{At_i}(\Pi_2)$ and $\mathrm{At_i}(\Pi_1) \cap \mathrm{At_o}(\Pi_2)$ that are output atoms in one program and input atoms in the other program. Recall that according to Definition 3 such atoms end up in $\mathrm{At_o}(\Pi)$ when $\Pi_1 \sqcup \Pi_2$ is formed. Our first modularity result deals with the classical semantics of DLP-functions.

Proposition 3. *Let Π_1 and Π_2 be two positive DLP-functions with the respective input signatures $\mathrm{At_i}(\Pi_1)$ and $\mathrm{At_i}(\Pi_2)$ so that $\Pi_1 \sqcup \Pi_2$ is defined. Then, for any mutually compatible interpretations $M_1 \subseteq \mathrm{At}(\Pi_1)$ and $M_2 \subseteq \mathrm{At}(\Pi_2)$,*

$$M_1 \cup M_2 \models \Pi_1 \sqcup \Pi_2 \iff M_1 \models \Pi_1 \text{ and } M_2 \models \Pi_2. \tag{5}$$

The case of minimal or stable models, respectively, is much more elaborate. The proof of Theorem 1 (see below) is based on *cumulative projections* defined for a join $\Pi_1 \sqcup \Pi_2$ of DLP-functions Π_1 and Π_2 and a pair of compatible interpretations $M_1 \subseteq \mathrm{At}(\Pi_1)$ and $M_2 \subseteq \mathrm{At}(\Pi_2)$. It is clear that $M_1 = M \cap \mathrm{At}(\Pi_1)$ and $M_2 = M \cap \mathrm{At}(\Pi_2)$ hold for $M = M_1 \cup M_2$ in this setting. Next, we use a total order $S_1 < \cdots < S_k$ of the SCCs in $\mathrm{DG}^+(\Pi_1 \oplus \Pi_2)$ to define an increasing sequence of interpretations

$$N^j = N_{\mathrm{i}} \cup (N \cap (\textstyle\bigcup_{i=1}^{j} S_i)), \tag{6}$$

for each interpretation $N \in \{M, M_1, M_2\}$ and $0 \leq j \leq k$. Furthermore, let $\Pi = \langle R, I, O, H\rangle = \Pi_1 \sqcup \Pi_2$. The relative complement $\overline{M} = \mathrm{At}(\Pi) \setminus M$ contains atoms *false* under M and we may associate a set $R[S_i]$ of rules with each SCC S_i using $\overline{M}$:

$$R[S_i] = \{(A \cap S_i) \leftarrow B \mid A \leftarrow B \in R,\, A \cap S_i \neq \emptyset, \text{ and } A \setminus S_i \subseteq \overline{M}\}. \tag{7}$$

For each rule $A \leftarrow B \in R$, the reduced rule $(A \cap S_i) \leftarrow B$ is the contribution of $A \leftarrow B$ for the component S_i in case $\bigvee(A \setminus S_i)$ is false under M, i.e., $\bigvee A$ is not eventually satisfied by some other component of Π; note that $M \models \Pi$ will be assumed in the sequel. Although $R[S_i]$ depends on M, we omit M in the notation for the sake of conciseness. For each $0 \leq j \leq k$, we may now collect rules associated with the first j components and form a DLP-function with the same signature as Π:

$$\Pi^j = \langle \textstyle\bigcup_{i=1}^{j} R[S_i], I, O, H\rangle. \tag{8}$$

This implies that non-input atoms in $\bigcup_{i=j+1}^{k} S_i$ are false under interpretations defined by (6). Since each rule of Π is either contained in Π_1 or Π_2, we may use $\overline{M_1} = \mathrm{At}(\Pi_1) \setminus M_1$ and $\overline{M_2} = \mathrm{At}(\Pi_2) \setminus M_2$ to define Π_1^j and Π_2^j analogously, using (7) and (8) for Π_1 and Π_2, respectively. It follows that $\Pi_1^j \sqcup \Pi_2^j$ is defined and $\Pi^j = \Pi_1^j \sqcup \Pi_2^j$ holds for every $0 \leq j \leq k$ due to the compatibility of M_1^j and M_2^j and the fact that $\Pi = \Pi_1 \sqcup \Pi_2$. Moreover, it is easy to inspect from the equations above that, by definition, $M^{j-1} \subseteq M^j$ and the rules of Π^{j-1} are contained in Π^j, for every $0 < j \leq k$.

Finally, we may accommodate the definitions from above to the case of a single DLP-function by substituting Π for Π_1 and $\varnothing$ for Π_2. Then, $\mathrm{DG}^+(\Pi)$ is partitioned into strongly connected components $S_1 < \cdots < S_k$ of Π and the construction of cumulative projections is applicable to an interpretation $M \subseteq \mathrm{At}(\Pi)$, giving rise to interpretations M^j and DLP-functions Π^j for each $0 \leq j \leq k$. Lemmas 1 and 2 deal with a structure of this kind associated with Π and describe how the satisfaction of rules and $\mathrm{At_i}(\Pi)$-minimality are conveyed under cumulative projections.

Lemma 1. *Let Π be a positive DLP-function with an input signature $\mathrm{At_i}(\Pi)$ and strongly connected components $S_1 < \cdots < S_k$. Given a model $M \subseteq \mathrm{At}(\Pi)$ for Π, the following hold for the cumulative projections M^j and Π^j, with $0 \leq j \leq k$:*

1. *For every* $0 \leq j \leq k$, $M^j \models \Pi^j$.
2. *If* $N^j \models \Pi^j$, *for some interpretation* $N^j \subseteq M^j$ *of* Π^j, *where* $j > 0$, *then* $N^{j-1} \models \Pi^{j-1}$, *for the interpretation* $N^{j-1} = N^j \setminus S_j$ *of* Π^{j-1}.
3. *If* M^j *is an* $\mathrm{At_i}(\Pi)$*-minimal model of* Π^j, *for* $j > 0$, *then* M^{j-1} *is an* $\mathrm{At_i}(\Pi)$*-minimal model of* Π^{j-1}.

Example 2. To demonstrate cumulative projections in a practical setting, let us analyse a DLP-function $\Pi = \langle R, \emptyset, \{a, b, c, d, e\}, \emptyset \rangle$, where R contains the following rules:

$$\begin{array}{ll} a \vee b \leftarrow; & d \leftarrow c; \\ a \leftarrow b; & e \leftarrow d; \\ b \leftarrow a; & d \leftarrow e; \\ a \leftarrow c; & c \vee d \vee e \leftarrow a, b. \end{array}$$

The SCCs of Π are $S_1 = \{a, b, c\}$ and $S_2 = \{d, e\}$ with $S_1 < S_2$. The classical models of Π are $M = \{a, b, d, e\}$ and $N = \{a, b, c, d, e\}$. Given M, Π^1 and Π^2 have the respective sets of rules $R[S_1] = \{a \vee b \leftarrow;\ a \leftarrow b;\ b \leftarrow a;\ a \leftarrow c\}$ and $R[S_1] \cup R[S_2]$ where $R[S_2] = \{d \leftarrow c;\ e \leftarrow d;\ d \leftarrow e;\ d \vee e \leftarrow a, b\}$. According to (6), we have $M^0 = \emptyset$, $M^1 = \{a, b\}$, and $M^2 = M$. Then, e.g., $M^1 \models \Pi^1$ and $M^2 \models \Pi^2$ by the first item of Lemma 1. Since M^2 is an $\emptyset$-minimal model of Π^2, the last item of Lemma 1 implies that M^1 is an $\emptyset$-minimal model of Π^1. □

Lemma 2. *Let* Π *be a positive DLP-function with an input signature* $\mathrm{At_i}(\Pi)$ *and strongly connected components* $S_1 < \cdots < S_k$. *Then, an interpretation* $M \subseteq \mathrm{At}(\Pi)$ *of* Π *is an* $\mathrm{At_i}(\Pi)$*-minimal model of* Π *iff* M *is an* $\mathrm{At_i}(\Pi)$*-minimal model of* Π^k.

Example 3. For Π from Example 2, the rule $d \vee e \leftarrow a, b$ forms the only difference between Π^2 and Π but this is insignificant: M is also an $\emptyset$-minimal model of Π. □

Proposition 4. *Let* Π_1 *and* Π_2 *be two* positive *DLP-functions with the respective input signatures* $\mathrm{At_i}(\Pi_1)$ *and* $\mathrm{At_i}(\Pi_2)$ *so that* $\Pi_1 \sqcup \Pi_2$ *is defined. Then, for any mutually compatible models* $M_1 \subseteq \mathrm{At}(\Pi_1)$ *and* $M_2 \subseteq \mathrm{At}(\Pi_2)$ *of* Π_1 *and* Π_2, *respectively,*

$$M_1 \cup M_2 \text{ is } \mathrm{At_i}(\Pi_1 \sqcup \Pi_2)\text{-minimal} \iff M_1 \text{ is } \mathrm{At_i}(\Pi_1)\text{-minimal and } M_2 \text{ is } \mathrm{At_i}(\Pi_2)\text{-minimal.}$$

Proof sketch. The proof of this result proceeds by induction on the cumulative projections M^j, M_1^j, and M_2^j induced by the SCCs $S_1 < \cdots < S_k$ of $\mathrm{DG}^+(\Pi_1 \sqcup \Pi_2)$, i.e., M^j is shown to be an $\mathrm{At_i}(\Pi)$-minimal model of Π^j iff M_1^j is an $\mathrm{At_i}(\Pi_1)$-minimal model of Π_1 and M_2^j is an $\mathrm{At_i}(\Pi_2)$-minimal model of Π_2, where Π^j, Π_1^j, and Π_2^j, for $0 \leq j \leq k$, are determined by (7) and (8) when applied to Π, Π_1, and Π_2. Lemma 2 closes the gap between the $\mathrm{At_i}(\Pi)$-minimality of $M^k = M$ as a model of Π^k from (8) with $j = k$ and as that of Π. The same can be stated about $M_1^k = M_1$ and $M_2^k = M_2$ but in terms of the respective projections $S_{i,1} < \cdots < S_{i,k}$ obtained for $i \in \{1, 2\}$. □

Lemma 3. *Let* Π_1 *and* Π_2 *be two DLP-functions with the respective input signatures* $\mathrm{At_i}(\Pi_1)$ *and* $\mathrm{At_i}(\Pi_2)$ *so that* $\Pi = \Pi_1 \sqcup \Pi_2$ *is defined. Then, also* $\Pi_1^{M_1} \sqcup \Pi_2^{M_2}$ *is defined for any mutually compatible interpretations* $M_1 \subseteq \mathrm{At}(\Pi_1)$ *and* $M_2 \subseteq \mathrm{At}(\Pi_2)$, *and* $\Pi^M = \Pi_1^{M_1} \sqcup \Pi_2^{M_2}$ *holds for their union* $M = M_1 \cup M_2$.

Theorem 1 (Module Theorem). *Let Π_1 and Π_2 be two DLP-functions with the respective input signatures $\mathrm{At_i}(\Pi_1)$ and $\mathrm{At_i}(\Pi_2)$ so that $\Pi_1 \sqcup \Pi_2$ is defined. Then, for any mutually compatible interpretations $M_1 \subseteq \mathrm{At}(\Pi_1)$ and $M_2 \subseteq \mathrm{At}(\Pi_2)$,*

$$M_1 \cup M_2 \in \mathrm{SM}(\Pi_1 \sqcup \Pi_2) \iff M_1 \in \mathrm{SM}(\Pi_1) \text{ and } M_2 \in \mathrm{SM}(\Pi_2).$$

Proof. Let $M_1 \subseteq \mathrm{At}(\Pi_1)$ and $M_2 \subseteq \mathrm{At}(\Pi_2)$ be compatible interpretations and $M = M_1 \cup M_2$. Due to compatibility, we can recover $M_1 = M \cap \mathrm{At}(\Pi_1)$ and $M_2 = M \cap \mathrm{At}(\Pi_2)$ from M. Additionally, we have $\mathrm{CR}(\Pi) = \mathrm{CR}(\Pi_1) \cup \mathrm{CR}(\Pi_2)$ and $\Pi^M = \Pi_1^{M_1} \sqcup \Pi_2^{M_2}$ is defined by Lemma 3. Now, $M \in \mathrm{SM}(\Pi)$ iff

$$M \text{ is an } \mathrm{At_i}(\Pi)\text{-minimal model of } \Pi^M \text{ and } M \models \mathrm{CR}(\Pi). \tag{9}$$

By Proposition 4, we get that (9) holds iff (i) M_1 is an $\mathrm{At_i}(\Pi_1)$-minimal model of $\Pi_1^{M_1}$ and $M_1 \models \mathrm{CR}(\Pi_1)$, and (ii) M_2 is an $\mathrm{At_i}(\Pi_2)$-minimal model of $\Pi_2^{M_2}$ and $M_2 \models \mathrm{CR}(\Pi_2)$. Thus, $M \in \mathrm{SM}(\Pi)$ iff $M_1 \in \mathrm{SM}(\Pi)$ and $M_2 \in \mathrm{SM}(\Pi)$. □

The moral of Theorem 1 and Definition 3 is that stable semantics supports modularisation as long as positive dependencies remain within program modules. The proof of the theorem reveals the fact that such modules may involve several strongly connected components. Splitting them into further modules is basically pre-empted by hidden atoms which cannot be placed in separate modules. Theorem 1 can be easily extended for DLP-functions consisting of more than two modules. In view of this, we say that a sequence $M_1, \ldots, M_n$ of stable models for modules $\Pi_1, \ldots, \Pi_n$, respectively, is *compatible*, iff M_i and M_j are pairwise compatible, for all $1 \leq i, j \leq n$.

Corollary 1. *Let $\Pi_1, \ldots, \Pi_n$ be a sequence of DLP-functions such that $\Pi_1 \sqcup \cdots \sqcup \Pi_n$ is defined. Then, for all compatible sequences $M_1, \ldots, M_n$ of interpretations,*

$$\textstyle\bigcup_{i=1}^{n} M_i \in \mathrm{SM}(\Pi_1 \sqcup \cdots \sqcup \Pi_n) \iff M_i \in \mathrm{SM}(\Pi_i), \text{ for all } 1 \leq i \leq n.$$

5 Applications

In this section, we demonstrate the applicability of Theorem 1 on three issues, viz. splitting DLP-functions, shifting disjunctions, and checking equivalence.

Splitting DLP-Functions. Theorem 1 is strictly stronger than the splitting-set theorem [5]. Given a DLP-function of form $\Pi = \langle R, \emptyset, O, \emptyset \rangle$ (which is essentially an "ordinary" DLP), a *splitting set* $U \subseteq O$ for Π satisfies, for each $A \leftarrow B, {\sim}C \in R$, $A \cup B \cup C \subseteq U$, whenever $A \cap U \neq \emptyset$. Given a splitting set U for Π, the *bottom*, $\mathrm{b}_U(R)$, of R with respect to U contains all rules $A \leftarrow B, {\sim}C \in R$ such that $A \cup B \cup C \subseteq U$, whereas the *top*, $\mathrm{t}_U(R)$, of R is $R \setminus \mathrm{b}_U(R)$. Thus, we may define $\Pi = \Pi_B \sqcup \Pi_T$, where $\Pi_B = \langle \mathrm{b}_U(R), \emptyset, U, \emptyset \rangle$ and $\langle \mathrm{t}_U(R), U, O \setminus U, \emptyset \rangle$. Then, Theorem 1 implies for any interpretation $M \subseteq \mathrm{At}(\Pi) = O$ that $M \cap U \in \mathrm{SM}(\Pi_B)$ and $M \in \mathrm{SM}(\Pi_T)$ iff $\langle M \cap U, M \setminus U \rangle$ is a *solution* for Π with respect to U, i.e., M is a stable model of Π. On the other hand, as demonstrated in previous work [8], the splitting-set theorem can be applied to DLP-functions like $\langle \{a \leftarrow {\sim}b;\ b \leftarrow {\sim}a\}, \emptyset, \{a, b\}, \emptyset \rangle$ only in a trivial way, i.e., for $U = \emptyset$ or $U = \{a, b\}$. In contrast, Theorem 1 applies to the preceding DLP-function, i.e., $\langle \{a \leftarrow {\sim}b\}, \{b\}, \{a\}, \emptyset \rangle \sqcup \langle \{b \leftarrow {\sim}a\}, \{a\}, \{b\}, \emptyset \rangle$ is defined.

Shifting Disjunctions. A further application of our module theorem results in a general shifting principle, defined as follows.

Definition 9. *Let $\Pi = \langle R, I, O, H\rangle$ be a DLP-function with strongly connected components $S_1 < \cdots < S_k$. The* general shifting *of Π is the DLP-function* $\mathrm{GSH}(\Pi) = \langle R', I, O, H\rangle$*, where R' is*

$$\{(A \cap S_i) \leftarrow B, \sim C, \sim(A \setminus S_i) \mid A \leftarrow B, \sim C \in \Pi,\ 1 \leq i \leq k, \text{ and } A \cap S_i \neq \emptyset\}.$$

This is a proper generalisation of the *local shifting* transformation [14] which is not applicable to the program Π given below because of head cycles involved.

Example 4. Consider Π with the following rules:

$$\begin{array}{ll} a \vee b \vee c \vee d \leftarrow; & \\ a \leftarrow b; & \quad c \leftarrow d; \\ b \leftarrow a; & \quad d \leftarrow c. \end{array}$$

For $\mathrm{GSH}(\Pi)$, the first rule is replaced by $a \vee b \leftarrow \sim c, \sim d$ and $c \vee d \leftarrow \sim a, \sim b$. It is easy to verify that both Π and $\mathrm{GSH}(\Pi)$ have $\{a, b\}$ and $\{c, d\}$ as their stable models. □

Theorem 2. *For any DLP-function $\Pi = \langle R, I, O, H\rangle$,* $\mathrm{SM}(\Pi) = \mathrm{SM}(\mathrm{GSH}(\Pi))$.

Proof. Let $S_1 < \cdots < S_k$ be the strongly connected components of Π and $M \subseteq \mathrm{At}(\Pi) = I \cup O \cup H$ an interpretation. By applying the construction of cumulative projections for both Π^M and $\mathrm{GSH}(\Pi)^M$, we obtain

$$(\Pi^M)^k = \{(A \cap S_i) \leftarrow B \mid A \leftarrow B, \sim C \in \Pi, \\ M \cap C = \emptyset, 1 \leq i \leq k, A \cap S_i \neq \emptyset, \text{ and } A \setminus S_i \subseteq \overline{M}\},$$

which coincides with $(\mathrm{GSH}(\Pi)^M)^k$. It follows by Lemma 2 that M is an $\mathrm{At_i}(\Pi)$-minimal model of Π^M iff M is an $\mathrm{At_i}(\Pi)$-minimal model of $\mathrm{GSH}(\Pi)^M$. □

Theorem 2 provides us a technique to split disjunctive rules among components that share them in order to get joins of components defined.

Example 5. For the DLP-function Π from Example 4, we obtain $R_1 = \{a \vee b \leftarrow \sim c, \sim d;\ a \leftarrow b;\ b \leftarrow a\}$ and $R_2 = \{c \vee d \leftarrow \sim a, \sim b;\ c \leftarrow d;\ d \leftarrow c\}$ as the sets of rules associated with $\Pi_1 = \langle R_1, \{c, d\}, \{a, b\}, \emptyset\rangle$ and $\Pi_2 = \langle R_2, \{a, b\}, \{c, d\}, \emptyset\rangle$, for which $\Pi_1 \sqcup \Pi_2 = \langle R_1 \cup R_2, \emptyset, \{a, b, c, d\}, \emptyset\rangle$ is defined. □

Checking Equivalence. Finally, we briefly mention how DLP-functions can be compared with each other at the level of modules as well as entire programs.

Definition 10. *Two DLP-functions Π_1 and Π_2 are* modularly equivalent*, denoted by $\Pi_1 \equiv_\mathrm{m} \Pi_2$, iff*

1. $\mathrm{At_i}(\Pi_1) = \mathrm{At_i}(\Pi_2)$ *and* $\mathrm{At_o}(\Pi_1) = \mathrm{At_o}(\Pi_2)$*, and*
2. *there is a bijection $f : \mathrm{SM}(\Pi_1) \to \mathrm{SM}(\Pi_2)$ such that for all interpretations $M \in \mathrm{SM}(\Pi_1)$, $M \cap \mathrm{At_v}(\Pi_1) = f(M) \cap \mathrm{At_v}(\Pi_2)$.*

Using $\equiv_{\mathrm{m}}$, we may reformulate the content of Theorem 2 as $\Pi \equiv_{\mathrm{m}} \mathrm{GSH}(\Pi)$. The proof for a congruence property lifts from the case of normal programs using the module theorem strengthened to the disjunctive case (i.e., Theorem 1).

Corollary 2. *Let Π_1, Π_2, and Π be DLP-functions. If $\Pi_1 \equiv_{\mathrm{m}} \Pi_2$ and both $\Pi_1 \sqcup \Pi$ and $\Pi_2 \sqcup \Pi$ are defined, then $\Pi_1 \sqcup \Pi \equiv_{\mathrm{m}} \Pi_2 \sqcup \Pi$.*

Applying Corollary 2 in the context of Theorem 2 indicates that shifting can be localised to a particular component $\Pi_2 = \mathrm{GSH}(\Pi_1)$ in a larger DLP-function $\Pi_1 \sqcup \Pi$.

A broader discussion which relates modular equivalence with similar notions proposed in the literature [15] is subject of future work but some preliminary comparisons in the case of disjunction-free programs are given by Oikarinen and Janhunen [8].

6 Conclusion and Discussion

In this paper, we discussed a formal framework for modular programming in the context of disjunctive logic programs under the stable-model semantics. We introduced syntax and semantics of DLP-functions, where input/output interfacing is realised, and proved a novel module theorem, establishing a suitable compositional semantics for program modules. Although our approach is not unique in the sense that there are different possibilities for defining the composition of modules, it nevertheless shows the limits of modularity in the context of a nonmonotonic declarative programming language. In any case, we believe that research in this direction not only yields results of theoretical interest but also could serve as a basis for future developments addressing practicably useful methods for software engineering in ASP.

Concerning previous work on modularity in ASP, Eiter, Gottlob, and Mannila [6] consider the class of disjunctive Datalog as query programs over relational databases. In contrast to our results, their module architecture is based on both *positive and negative dependencies* and no recursion between modules is tolerated. These constraints enable a straightforward generalisation of the splitting-set theorem for that architecture. Eiter, Gottlob, and Veith [3] address modularity within ASP by viewing program modules as *generalised quantifiers* allowing nested calls. This is an abstraction mechanism typical to programming-in-the-small approaches. Finally, Faber *et al*. [16] apply the *magic set method* in the evaluation of Datalog programs with negation, introducing the concept of an *independent set*, which is a specialisation of a splitting set. The module theorem put forward by Faber *et al*. [16] is, however, weaker than Theorem 1 presented in Section 4.

References

1. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
2. Gaifman, H., Shapiro, E.: Fully Abstract Compositional Semantics for Logic Programs. In: Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL'89). (1989) 134–142
3. Eiter, T., Gottlob, G., Veith, H.: Modular Logic Programming and Generalized Quantifiers. In: Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97). Volume 1265 of LNCS, Springer (1997) 290–309

4. Baral, C., Dzifcak, J., Takahashi, H.: Macros, Macro Calls and Use of Ensembles in Modular Answer Set Programming. In: Proceedings of the 22nd International Conference on Logic Programming (ICLP'06). Volume 4079 of LNCS, Springer (2006) 376–390
5. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: Proceedings of the 11th International Conference on Logic Programming (ICLP'94), MIT Press (1994) 23–37
6. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems **22**(3) (1997) 364–418
7. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective Integration of Declarative Rules with External Evaluations for Semantic Web Reasoning. In: Proceedings of the 3rd European Semantic Web Conference (ESWC'06). Volume 4011 of LNCS, Springer (2006) 273–287
8. Oikarinen, E., Janhunen, T.: Modular Equivalence for Normal Logic Programs. In: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06). (2006) 412–416.
9. Gelfond, M.: Representing Knowledge in A-Prolog. In Kakas, A., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II. Volume 2408 of LNCS, Springer (2002) 413–451
10. Janhunen, T.: Some (In)translatability Results for Normal Logic Programs and Propositional Theories. Journal of Applied Non-Classical Logics **16**(1–2) (2006) 35–86
11. Janhunen, T., Oikarinen, T.: Automated Verification of Weak Equivalence within the SMODELS System. Theory and Practice of Logic Programming (2006). To appear
12. Ben-Eliyahu, R., Dechter, R.: Propositional Semantics for Disjunctive Logic Programs. Annals of Mathematics and Artificial Intelligence **12**(1–2) (1994) 53–87
13. Lifschitz, V.: Computing Circumscription. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85), Morgan Kaufmann (1985) 121–127
14. Eiter, T., Fink, M., Tompits, H., Woltran, T.: Simplifying Logic Programs under Uniform and Strong Equivalence. In: Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'03). Volume 2923 of LNAI, Springer (2004) 87–99
15. Eiter, T., Tompits, H., Woltran, S.: On Solution Correspondences in Answer-Set Programming. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05), Professional Book Center (2005) 97–102
16. Faber, W., Greco, G., Leone, N.: Magic Sets and Their Application to Data Integration. In: Proceedings of the 10th International Conference on Database Theory (ICDT'05). Volume 3363 of LNCS, Springer (2005) 306–320