A semantics for modular general logic programs

Sandro Etalle*,2

Faculty W I N S, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Received June 1996; revised June 1997
Communicated by G. Levi

Abstract

Modular programs are built as a combination of separate modules, which may be developed and verified separately. Therefore, in order to reason over such programs, compositionality plays a crucial role: the semantics of the whole program must be obtainable as a simple function from the semantics of its individual modules. In the field of logic programming, the need for a compositional semantics has been long recognized, however, while for definite (i.e. negation-free) logic programs a few such semantics have been proposed, in the literature of normal logic programs (programs which employ the negation operator), compositionality has received scarce attention. This is mainly due to the fact that normal programs typically have a nonmonotonic behavior, which is difficult to fit in a compositional framework.

Here we propose a declarative compositional semantics for general logic programs. First, a compositional semantics for first-order modules is presented and proved correct wrt the set of logical consequences of the module in three-valued logic. In a second stage, the obtained results are applied to modular normal logic programs, obtaining a semantics which is correct with respect to the set of logical consequences of the completion of the program and – in contrast with the other approaches – which is always computable. This semantics might be regarded as a compositional counterpart of Kunen's semantics. Finally we discuss and show how these results have to be modified in order to be applied to normal constraint logic programs.

Keywords: Logic programming; Modularity; Open programs; Negation; Semantics; Constraint logic programming

Contents

1. Introduction .......................................................... 52
2. Preliminaries ......................................................... 53

* Tel.: (+31-20) 525 7541; fax: (+31-20) 525 7490; e-mail: sandro@wins.uva.nl.
2 A preliminary, shorter version of this paper has appeared in the Proceedings of the 1996 Joint International Conference on Logic Programming.

This work was partially carried out at the Department of Computer Science of the University of Genova.
1. **Introduction**

*Modularity in Logic Programming.* Modularity is a crucial feature of most modern programming languages. It allows one to construct a program out of a number of separate *modules*, which can be developed, optimized and verified separately. Indeed, the incremental and modular design is by now a well established software-engineering methodology which helps to verify and maintain large applications.

In the logic programming field, modularity has received a considerable attention (see for instance [8]), and has generated two distinct approaches to it: the first one is inspired by the work of O'Keefe [28] and is based on the consideration that module composition in basically a *metalinguistic* operation, in which the modular construct should be independent from the logic language being used; the second one originated with the work of Miller [26, 27], and is obtained by using a logical system richer than Horn clauses, thus providing a *linguistic* approach.

In this paper we follow the first approach. Viewing modularity in terms of *metalinguistic* operations on programs has several advantages. In fact it leads to the definition of a simple and powerful methodology for structuring programs which does not require to extend the underlying language’s syntax. This is essential if we want to compose modules written in different languages. Furthermore, the typical mechanisms of the object-oriented paradigm, such as encapsulation and information hiding can be easily realized within this framework (see [3]).

*The need for a compositional semantics.* In order to deal with modular programs, it is crucial that the semantics we refer to is *compositional*, i.e. that the semantics of the whole program is a (simple) function of the semantics of its modules. The need for a compositional semantics becomes even more pressing if one wants to build applications in which logic modules are combined with modules that are not logic programs themselves, such as constraint solvers, imperative programs, neural networks, etc. In fact, compositionality enables one to reason about the logic module in isolation, while the reference to knowledge provided by other modules is maintained intact.
In logic programming, this need for a compositional semantics has been long rec-
cognized. For definite (i.e. negation-free) logic programs a few semantics have been
proposed; to the best of our knowledge, the first papers to discuss various forms of
compositional semantic characterizations of definite logic programs were the ones of
Lassez and Maher [21,23], further work has been done by Mancarella and Pedreschi
[25] and Brogi et al. [6]. In [15] Gaifman and Shapiro proposed a compositional
semantics, which was further extended in [5] and – for CLP programs – in [14].

Compositionality vs. non-monotonicity. However, in the development of semantics
for normal logic programs (logic programs which employ the negation operator),
compositionality has been widely disregarded. Notable exception to this are the pa-
pers by Maher [24] and Ferrand and Lallouet [12] (comparison between these papers
and this one is deferred to the concluding section). The reason of this disattention
is that, because of the presence of the negation-as-failure mechanism, the semantics
of normal logic programs is typically non-monotonic. Now, compositionality and non-
monotonicity are (almost) irreconcilable aspects. Compositionality implies that the ‘old
knowledge’ is maintained when new knowledge is added. Non-monotonicity is de-
efined exactly as the opposite. Thus, it seems that one can enjoy either compositionality
or non-monotonicity, not both. Still, we need both aspects: on one hand, the non-
monotonicity that arises from the use of negation as failure is something we want in
our logic programming language, because it enables us to define relations in a natural
and succinct manner. On the other hand, modularity, and therefore compositionality
of the declarative semantics, is essential when one wants to use a logic programming
language in real life applications.

Contribution of this paper. In this paper we propose a semantics for modular logic
programs. This semantics is compositional while remaining non-monotonic to a certain
extent. In essence, the semantics is compositional and monotonic on the level of union
of modules, while addition of clauses to modules remains a non-monotonic operation.

We carry out our task by first providing a compositional semantics for first-order
programs, which extends the semantics given by Sato [30] (which in turn can be
regarded as an extension to first-order programs of Kunen’s [20] semantics). In a
second stage we show how this can be naturally used to provide a compositional
semantics for normal logic programs and normal CLP. The semantics we propose can
also be regarded as a compositional extension of Kunen’s semantics [20]. Finally we
discuss and show how these results have to be modified in order to be applied to
normal Constraint Logic Programs, and, in the last section, to programs in which are
present some base (built-in) predicates which have a predefined meaning.

2. Preliminaries

We assume that the reader is familiar with the basic concepts of logic programming;
throughout the paper we use the standard terminology of [1,22]. Symbols with a ~
on top denote tuples of objects, for instance \( \bar{x} \) denotes a tuple of variables \( x_1, \ldots, x_n \),
and $x = y$ stands for $x_1 = y_1 \land \cdots \land x_n = y_n$. Throughout the paper we will work with three valued logic: the truth values are then true, false and undefined. We adopt the truth tables of [19], which can be summarized as follows: the usual logical connectives have value true (or false) when they have that value in ordinary two valued logic for all possible replacements of undefined by true or false, otherwise they have the value undefined. Three valued logic allows us to define connectives that do not exist in two valued logic. In particular in the sequel we use the symbol $\leftrightarrow$ corresponding to Lukasiewicz’s operator of “having the same truth value”: $a \leftrightarrow b$ is true if $a$ and $b$ are both true, both false or both undefined; in any other case $a \leftrightarrow b$ is false. As opposed to it, the usual $\leftrightarrow$ is undefined when one of its arguments is undefined.

In most cases we restrict our attention to formulae which we consider “well-behaving” in the three valued semantics. A logic connective $\Diamond$ is allowed iff the following property holds: when $a \Diamond b$ is true or false then its truth value does not change if the interpretation of one of its argument is changed from undefined to true or false. A first order formula is allowed iff it contains only allowed connectives. Notice that any formula containing the connective $\leftrightarrow$ is not allowed, while formulae built with the three-valued counterpart of the “usual” logic connectives are allowed. Allowed formulae can be seen as monotonic functions over the lattice on the set \{undefined, true, false\} which has undefined as bottom element and true and false are not comparable. Finally, in what follows we always assume the equality symbol $=$ to be part of the language of the programs and modules we deal with, so – in some cases – in order to avoid confusion we will use $\equiv$ to denote equality at meta-level. Modules are defined on a fixed base language $\mathcal{L}_B$, which contains all the constants and function symbols which may occur in the module itself, and the predicate symbols of those relations which have a predefined meaning. We assume that $\mathcal{L}_B$, always contains the equality symbol and (with a harmless overload of notation), three predicative constants $t, f, u$, corresponding to the truth values true, false, undefined. The primitive predicate symbols in $\mathcal{L}_B \setminus \{t, f, u\}$ are assumed to be defined in a fixed first-order consistent base theory $\mathcal{A}$. Typical choices for $\mathcal{A}$ are for example the set of equality axioms together with Clark’s equality theory, the domain closure axiom, or axioms defining arithmetic primitives. A relation we will always assume being part of the language is equality (=); its meaning may be either the identity over the domain of discourse or – if one prefers – it may be given by a suitable complete theory, in which case it is assumed to be incorporated in $\mathcal{A}$.

Semantics. A three-valued structure $\mathcal{S}$ for the language $\mathcal{L}_B \cup \{p_1, \ldots, p_k\}$ is a triple $(\text{Dom, Rel, Fun})$ where Dom is the domain (or universe) and it is a non-empty set, Fun is a set of functions on Dom, one for each function symbol in $\mathcal{L}_B$, and Rel is an interpretation over Dom, which is two valued for the predicates in $\mathcal{L}_B \setminus \{u\}$, and three valued for the other predicate symbols ($u \cup \{p_1, \ldots, p_k\}$). We also assume that $t, f$ and $u$ always take the value true, false and undefined. Given a sentence $\phi$, we use the notation $Val(\phi, \mathcal{S})$ to denote the truth value of $\phi$ in $\mathcal{S}$. Further, we say that $\mathcal{S}$ is a model of the set of sentences $\Gamma$ if for each sentence $\phi \in \Gamma$ we have that $Val(\phi, \mathcal{S}) = true$; consequently, the three-valued logical consequence relation $\models$ is defined as follows: $\Gamma \models \phi$ iff $Val(\phi, \mathcal{S}) = true$ for every model $\mathcal{S}$ of $\Gamma$. 
First-order programs and modules. A modular logic program consists of a number of logic modules, each of which consists of a number of predicate definitions. The definition (of a predicate \( p \)) is a formula of the form
\[
p(\bar{x}) \leftrightarrow \phi[\bar{x}]
\]
where \( \bar{x} \) is a tuple of distinct variables, and \( \phi[\bar{x}] \) is a first order formula whose free variables are exactly the variables of \( \bar{x} \) (the notation \( \phi[\bar{x}] \) is used to emphasize this fact). \( p(\bar{x}) \) and \( \phi[\bar{x}] \) are usually referred to as the head and the body of the definition. Then, a module \( M \) on a base language \( \mathcal{L}_B \) is a collection of predicate definitions such that each predicate is defined at most once, and none of the predicates in \( \mathcal{L}_B \) is defined in \( M \).

Example 1. The following elementary module OddEven will be used only for the preliminaries; it provides a definition for the predicates \( \text{even}(y) \) and \( \text{odd}(y) \).
\[
\begin{align*}
c_1 & : \text{odd}(x) \leftrightarrow \exists y (x = y + 1 \land \text{even}(y)) \\
c_2 & : \text{even}(x) \leftrightarrow x = 0 \lor \exists y (x = y + 1 \land \text{odd}(y)).
\end{align*}
\]

We denote by \( \text{Def}(M) \) the set of predicates that are defined in \( M \), and let \( \text{Open}(M) \) be the set of predicates which are neither in \( \text{Def}(M) \) nor in \( \mathcal{L}_B \) (recall that we assume that \( \text{Def}(M) \cap \mathcal{L}_B = \emptyset \)). Predicates in \( \text{Open}(M) \) are supposed to be imported, i.e. defined in some other – maybe unspecified – module \( M' \). Those predicates are also referred to as the open predicates of \( M \). If \( \text{Open}(M) \) is empty then the module is said to be closed. A closed module corresponds to a classical first-order program. Also, we define \( \text{Pred}(M) \) as \( \text{Def}(M) \cup \text{Open}(M) \).

The unfolding operation. The semantics we are going to give is based on the unfolding operation. We now recall its definition.

Definition 2 (Unfolding). Let \( cl : p(\bar{x}) \leftrightarrow \phi[\bar{x}] \) and \( d : q(\bar{y}) \leftrightarrow \psi[\bar{y}] \) be two predicate definitions (which we assume to be standardized apart). Let \( q(\bar{t}) \) be an atomic subformula of \( \phi[\bar{x}] \). Then, by unfolding \( q(\bar{t}) \) in \( cl \) (via \( d \)) we mean substituting \( q(\bar{t}) \) with \( \psi[\bar{t}/\bar{y}] \) in \( cl \). In this case \( cl \) is called the unfolded definition while \( d \) is the unfolding one.

For example, let us consider again module OddEven of Example 1. By unfolding \( \text{odd}(y) \) in clause \( c_2 \) (via \( c_1 \)) we obtain the rule
\[
\text{even}(x) \leftrightarrow x = 0 \lor \exists y (x = y + 1) \land \exists z (y = z + 1 \land \text{even}(z)))
\]
This can be further rewritten as\(^3\)
\[
\text{even}(x) \leftrightarrow x = 0 \lor \exists z (x = (z + 1) + 1 \land \text{even}(z)))
\]

\(^3\) We are always allowed to replace a definition \( p(\bar{x}) \leftrightarrow \phi[\bar{x}] \) with \( p(\bar{x}) \leftrightarrow \phi'[\bar{x}] \) provided that \( \phi \) and \( \phi' \) are both allowed formulae and that \( \mathcal{A} \models \phi[\bar{x}] \leftrightarrow \phi'[\bar{x}] \), where \( \mathcal{A} \) is the fixed base theory. If these conditions are satisfied such a replacement leaves unmodified all the program’s properties we are interested in.
Let $d$ be a predicate definition, and $N$ be a module, we say that $d'$ is obtained by unfolding $d$ with $N$, and we write $d' \equiv d \circ N$ iff $d'$ is obtained from $d$ by unfolding (via the definitions of $N$) all its body atoms for which there exist a definition in $N$ (then if $\text{Pred(\text{Body}(d))} \cap \text{Def}(N) = \emptyset$ we have that $d \circ N \equiv d$). Furthermore, if $M$ is the module $\{d_1, \ldots, d_k\}$ we write

$$M' \equiv M \circ N$$

and we say that $M'$ is obtained by unfolding $M$ with $N$ iff $M' \equiv \{d_1 \circ N, \ldots, d_k \circ N\}$. As usual, we associate the $\circ$ operator to the left. Thus, $M \circ N \circ O$ should be read as $(M \circ N) \circ O$. Now, for a module $M$, we adopt the following notation:

$$M^n \equiv \begin{cases} \{p(\vec{x}) \Leftrightarrow p(\vec{x}) \mid p \in \text{Def}(M)\} & \text{if } n = 0, \\ M^{n-1} \circ M & \text{otherwise}. \end{cases}$$

So, intuitively, $M^n$ is obtained from $M$ by unfolding $n$ times all its body atoms (using the definitions of $M$ itself as unfolding definitions). Notice that $M \equiv M^1 \equiv M \circ M^0 \equiv M^0 \circ M$.

The unfolding operation, when applied to a closed module, is correct, in the sense that it maintains the set of allowed logical consequences. This is the content of the following Lemma, which is due to Sato [30].

**Lemma 3** (Correctness of the unfolding operation [30]). Let $M_0, M_1, \ldots, M_n$ be a sequence of closed modules on the base language $\mathcal{L}_B$ such that for each $i \in [1, n]$ there exists definitions $c_{l_i}$ and $c_{l'_i}$ such that

(i) $c_{l'_i}$ is obtained from $c_{l_i}$ via an unfolding operation, using a definition of $M_0$ as unfolding definition.

(ii) $M_i = M_{i-1} \setminus \{c_{l_i}\} \cup \{c_{l'_i}\}$

Then, for any allowed formula $\phi$, $M_0 \cup A \models \phi$ iff $M_n \cup A \models \phi$.

Kunen’s semantics for first-order programs. We now restate the results of [30] on the semantics for first-order logic programs in a form based on the unfolding operation. Originally, in [20], Kunen proposed to consider as the semantics for normal logic programs the set of logical consequences of the programs completion in three-valued logic. This approach – as opposed to virtually all others available for normal programs – has the advantage of leading to a semantics which is always computable, and thus had a great impact in the logic programming community. In [30], Sato provides an extension to first-order programs of the above-mentioned characterization given in [20].

First, we need to define the skeleton of a module. For a module $M$, we denote

$$\text{Dummy}(M) \equiv \{p(\vec{x}) \Leftrightarrow u \mid p \in \text{Def}(M)\}.$$  

Then, the skeleton of $M$ is defined as

$$[M] \equiv M \circ \text{Dummy}(M)$$
Using the skeleton and the unfolding operator, we can generate an infinite chain 
\([M^0],[M^1],[M^2],\ldots\).

**Example 4.** Consider the module \(\text{OddEven}\) of Example 1. Its skeleton \([\text{OddEven}]\) is

\[
\begin{align*}
\text{odd}(x) & \iff \exists y (x = y + 1 \land y < x ) \\
\text{even}(x) & \iff x = 0 \lor \exists y (x = y + 1 \land y < x )
\end{align*}
\]

Further, \([\text{OddEven}^2]\), the skeleton of \(\text{OddEven}\) is (after some rewriting)

\[
\begin{align*}
\text{odd}(x) & \iff x = 1 \lor \exists y (x = y + 1 \land y < x ) \\
\text{even}(x) & \iff x = 0 \lor \exists y (x = y + 1 \land y < x )
\end{align*}
\]

Intuitively, the skeleton of a module contains all the knowledge that is 'immediately accessible', i.e. that can be obtained directly applying just one definition. The following is a simple lemma we will need to use throughout the paper.

**Lemma 5.** Let \(M\) be a module on base language \(\mathcal{L}_B\), let \(\Delta\) be a base theory for \(\mathcal{L}_B\). If \(\phi\) is an allowed formula, then \([M] \cup \Delta \models \phi\) iff \(\Delta \models \phi \circ [M]\).

**Proof.** We prove the thesis by structural induction on \(\phi\). Suppose \(\phi\) is an atom of the form \(p(\vec{t})\). We have to consider two cases.

If \(p \notin \text{Def}(M)\), then \([M] \cup \Delta \models \phi\) iff \(\Delta \models \phi\). Again, because \(p \notin \text{Def}(M)\), \(p(\vec{t}) \equiv p(\vec{t}) \circ [M]\). Thus the thesis holds.

Otherwise, \(p \in \text{Def}(M)\). In this case \([M]\) must contain a definition \(p(\vec{x}) \leftrightarrow \psi\). But then, since we know that \(\text{Def}(M) \cap \text{Pred}(\psi) = \emptyset\), we have that

\[
[M] \cup \Delta \models p(\vec{t})
\]

iff \([M] \cup \Delta \models (\vec{x} = \vec{t}) \land \psi\) since \(\text{Def}(M) \cap \text{Pred}(\psi) = \emptyset\),

iff \(\Delta \models (\vec{x} = \vec{t}) \land \psi\),

iff \(\Delta \models p(\vec{t}) \circ [M]\).

The inductive steps for the logical operators are straightforward. \(\square\)

Finally, we can restate Theorem 3.3 from [30] (page 66) as follows.

**Theorem 6.** Let \(M\) be a module in a base language \(\mathcal{L}_B\). Then, for any allowed formula \(\phi\), \(M \cup \Delta \models \phi\) iff, for some \(n\), \([M^n] \cup \Delta \models \phi\).

**Proof.** We have that

\[
M \cup \Delta \models \phi
\]

by [30, Theorem 3.3]

iff \(\exists n\) such that \(\Delta \models \phi \circ [M^n]\) by Lemma 5

iff \(\exists n\) such that \([M^n] \cup \Delta \models \phi\).
It is important to notice that the fact that in [30] equality is always assumed to be the identity over the domain of discourse, while here we allow it to be defined by any complete theory, is not a source of conflicts. In fact – since the manipulations we employ never introduce the symbol = – all we have to do is to use a different relation symbol to denote the identity relation. □

3. A compositional semantics

Following the original paper of O’Keefe [28], the approach to modular programming we consider here is based on a meta-linguistic programs composition mechanism. In this framework, logic programs are seen as elements of an algebra and the composition operation is modeled by an operator on the algebra. Viewing modularity in terms of meta-linguistic operations on programs has several advantages. In fact it leads to the definition of a simple and powerful methodology for structuring programs which does not require to extend the underlying language’s syntax. This is not the case if one tries to extend programs by linguistic mechanisms, an approach which originated with the work of Miller [26, 27]. Moreover, meta-linguistic operations are quite powerful. For instance, the compositional systems of Mancarella and Pedreschi [25], Gaifman and Shapiro [15], Bossi et al. and Brogi et al. [6, 7] can be seen as different instances of this idea. Furthermore, the typical mechanisms of the object-oriented paradigm, such as encapsulation and information hiding, as well as more complex form of composition mechanisms – in which we may distinguish between imported, exported, and local (hidden) predicates – can be easily realized within this framework. These mechanisms are implemented – for instance – in the language Gödel [16], in Quintus Prolog [29] and in SICStus Prolog [9]. For a more detailed analysis we refer to the survey of Bugliesi et al. [8].

3.1. Module composition

To compose first-order modules we follow the same approach of [5] and use a simple program union operator.

**Definition 7 (Module Composition).** Let $M_1$ and $M_2$ be modules on the base language $\mathcal{L}_B$. We define

$$M_1 \oplus M_2 = M_1 \cup M_2$$

provided that $\text{Def}(M_1) \cap \text{Def}(M_2) = \emptyset$. Otherwise $M_1 \oplus M_2$ is undefined.

This definition extends in a straightforward way to the case of several modules: $M_1 \oplus \cdots \oplus M_k$ is defined naturally as $(M_1 \oplus \cdots \oplus M_{k-1}) \oplus M_k$. Note that, in the definition we use, we require $\text{Def}(M_i) \cap \text{Def}(M_j) = \emptyset$, for all distinct $i$ and $j$. At first, this seems to be rather restrictive, in that it prevents one from refining the definition of a predicate $p$ in a module $M_1$, by composing it with some module $M_2$ also containing
a definition for \( p \). However, the problem can be easily solved by the use of some renaming and an additional ‘interface’ module. Assume that \( p \) is defined both in \( M_1 \) and in \( M_2 \); in this case, we rename \( p \) to \( p_1 \) (resp. \( p_2 \)) in the head of the definition of \( p \) in \( M_1 \) (resp. \( M_2 \)), resulting in a module \( N'_1 \) (resp. \( N'_2 \)). We assume that \( p_1 \) and \( p_2 \) are “new” predicate symbols. Then, we define an interface module as follows:

\[
I = \{ p(\bar{x}) \Leftrightarrow p_1(\bar{x}) \lor p_2(\bar{x}) \}
\]

Now observe that \( I \oplus N'_1 \oplus N'_2 \) is well-defined (provided there are no other name clashes) and behaves exactly the way we would expect \( M_1 \oplus M_2 \) to. Thus, the disjointness condition is not a real restriction. On the other side, this condition allows us to circumvent a number of unnecessary technicalities, and, in particular, to keep module composition a monotonic operation. Further, the use of such an interface allows one to specify explicitly which kind of composition we demand (in this case it was an or-composition, but other forms of composition are possible as well). Finally, it is worth noticing that mutual recursion among modules is allowed.

### 3.2. Expressiveness of modules

Now, we have to give a formal definition to the abstract concept of (semantical) expressiveness of modules, for this we have to take into account the fact that modules are meant to be composed together.

In the rest of this section, we always assume that all the modules are given on the same fixed base language \( \mathcal{L}_B \), and that the meaning of the predicates and functions in \( \mathcal{L}_B \) is provided by a fixed base theory \( \Delta \).

**Definition 8.** Let \( M \) and \( N \) be two modules on the base language \( \mathcal{L}_B \), such that \( \text{Def}(M) = \text{Def}(N) \). Let \( \Delta \) be a base theory for \( \mathcal{L}_B \). We say that

\( M \) is **compositionally more expressive** than \( N \) (w.r.t. \( \Delta \)),

\( M \succsim \Delta N \),

iff for any other module \( Q \) (on \( \mathcal{L}_B \)) such that \( M \oplus Q \) and \( N \oplus Q \) are defined, we have that for any allowed formula \( \phi \), if \( N \oplus Q \cup \Delta \models \phi \) then \( M \oplus Q \cup \Delta \models \phi \). We also say that

\( M \) and \( N \) are **compositionally equivalent** (w.r.t. \( \Delta \)),

\( M \sim \Delta N \)

iff \( M \succsim \Delta N \succsim \Delta M \).

In other words, we say that two first-order modules are compositionally equivalent if they have the same set of logical consequences in every possible context. Therefore \( \sim \) is actually a congruence relation. The following lemma states an obvious yet important property of \( \succsim \).

**Lemma 9.** Let \( M, N \) and \( Q \) be modules such that \( M \oplus Q \) is defined. If \( M \succsim \Delta N \) then \( M \oplus Q \succsim \Delta N \oplus Q \).
3.3. A compositional semantics for first-order modules

In this section, we are going to prove our main result, which will provide a computable, compositional semantics for first-order modules.

First we need some technical tools. The main one is the following operator. Let \( \varphi \) and \( \varphi' \) be allowed formulae. We write

\[
\varphi \leftrightarrow \varphi'
\]

if \( \varphi' \) can be obtained from \( \varphi \) by substituting some (or none) of its subformulas with the constant \( u \). For example, we have that

\[
a \land ((b \lor c) \rightarrow d) \leftrightarrow a \land ((b \lor c) \rightarrow u) \leftrightarrow a \land u \leftrightarrow u.
\]

Clearly, \( \leftrightarrow \) determines an order relation; further, it enjoys the following simple yet important property. The proof of the following Remark is straightforward.

**Remark 10.** Let \( \varphi \) and \( \varphi' \) be allowed formulae such that \( \varphi \leftrightarrow \varphi' \). Then for any structure \( S \) we have that

- if \( \text{Val}(\varphi', S) = \text{true} \) (resp. \( \text{false} \)) then \( \text{Val}(\varphi, S) = \text{true} \) (resp. \( \text{false} \)) as well.

Consequently, for any theory \( I \)

- if \( I \models \varphi' \) then \( I \models \varphi \).

Now, we extend the domain of \( \leftrightarrow \) to modules as follows (this causes no ambiguity). We write

\[
M \leftrightarrow N
\]

if \( \text{Def}(M) \equiv \text{Def}(N) \) and for each definition \( p(\bar{x}) \leftrightarrow \varphi[\bar{x}] \) of \( M \) there exists in \( N \) a (renaming of) a definition \( p(\bar{x}) \leftrightarrow \varphi'[\bar{x}] \) such that \( \varphi[\bar{x}] \leftrightarrow \varphi'[\bar{x}] \). Of course, if \( M \leftrightarrow N \leftrightarrow Q \) then \( M \leftrightarrow Q \). Therefore \( \leftrightarrow \) induces an order relation on the modules, and it will be used in that sense. Now, it is important to relate \( \leftrightarrow \) and \( \models_A \).

**Lemma 11.** Let \( M \) and \( N \) be modules on \( L_B \). If \( M \leftrightarrow N \) then \( M \models_A N \).

**Proof.** Take any module \( Q \) such that \( M \oplus Q \) is defined. Then \( N \oplus Q \) is defined as well and \( M \oplus Q \leftrightarrow N \oplus Q \). In order to prove the thesis we have to show that, for any allowed formula \( \varphi \) if \( N \oplus Q \models \varphi \) then \( M \oplus Q \models \varphi \). We now show that for each \( n \),

\[
\text{if } [(N \oplus Q)^n] \models \varphi \text{ then } [(M \oplus Q)^n] \models \varphi
\]

By Theorem 6 this will imply the thesis. Assume that \( [(N \oplus Q)^n] \cup A \models \varphi \). By Lemma 5 we have that \( A \models (N \oplus Q)^n \). Now, by Remark 12 we have that \( [(M \oplus Q)^n] \models [(N \oplus Q)^n] \), so \( \varphi \circ [(N \oplus Q)^n] \) can be obtained from \( \varphi \circ [(M \oplus Q)^n] \) by replacing some subformulas with the predicative constant \( u \). Therefore, being both \( \varphi \circ [(N \oplus Q)^n] \) and \( \varphi \circ [(M \oplus Q)^n] \) allowed formulae, by Remark 10 we have that \( A \models \varphi \circ [(M \oplus Q)^n] \). Again, by Lemma 5 we have that \( [(M \oplus Q)^n] \cup A \models \varphi \). This proves (1), and thus the thesis. \( \square \)
It is easy to check that the converse of this lemma does not hold. Thus, \(\rightsquigarrow\) is a stronger order relation than \(\Rightarrow_A\). Further, the relation \(\rightsquigarrow\) has the advantage of being independent from the base theory \(A\). Other simple properties of the operators \(\rightsquigarrow\) and \([\ ]\) that are going to be needed in the sequel are the following.

**Remark 12.** For any module \(M\), we have that
- \(M \rightsquigarrow [M]\)
- \([M^{n+1}] \rightsquigarrow [M^n]\).

Also, let \(M\), \(N\) and \(Q\) be modules on a common base language \(\mathcal{L}_B\). If \(M \rightsquigarrow N\) then
- \([M] \rightsquigarrow [N]\).
- \(M \oplus Q \rightsquigarrow N \oplus Q\),
- \(M \circ Q \rightsquigarrow N \circ Q\) and
- \(Q \circ M \rightsquigarrow Q \circ N\).

The proofs of these properties are straightforward and hence omitted. Next, we need our main lemma. The proof is long, tedious and technical, and can be found in the appendix.

**Lemma 13.** Let \(M\) and \(N\) be modules on the base language \(\mathcal{L}_B\) such that \(M \oplus N\) is defined. Then \([(M^n) \oplus [N^n]] \rightsquigarrow [(M \oplus N)^n]\).

Now, we are finally able to prove our main theorem.

**Theorem 14 (Main).** Let \(M_1, \ldots, M_k\) be first-order modules such that \(M_1 \oplus \cdots \oplus M_k\) is defined. Let also \(A\) be a base theory for \(\mathcal{L}_B\). Then, for each allowed \(\phi\) there exists an integer \(n\) such that the following statements are equivalent:

\[\begin{align*}
&\cdot M_1 \oplus \cdots \oplus M_k \cup A \models \phi; \\
&\cdot [M^n_1] \oplus \cdots \oplus [M^n_k] \cup A \models \phi.
\end{align*}\]

**Proof.** The proof is given by induction on the number of modules \(k\). First we consider the base case: \(k = 2\).

\((\Leftarrow)\) From Remark 12 we know that \(M^n_1 \to [M^n_1]\) and therefore (via the same Remark) that \(M^n_1 \oplus M^n_2 \rightsquigarrow [M^n_1] \oplus [M^n_2]\). So, by Lemma 11, if \([M^n_1] \oplus [M^n_2] \cup A \models \phi\) then \(M^n_1 \oplus M^n_2 \cup A \models \phi\). Therefore, by the correctness of the unfolding operation, Lemma 3 and Lemma 9, it follows that \(M_1 \oplus M_2 \cup A \models \phi\).

\((\Rightarrow)\) Assume that \(M_1 \oplus M_2 \models \phi\). By Theorem 6 we have that there exists an integer \(n\) such that

\[\left((M_1 \oplus M_2)^n\right) \cup A \models \phi.\]

Now,

\[\left((M_1 \oplus M_2)^n\right) \quad \text{by Lemmata 13 and 11}\]

\[\leq_A \left([M^n_1] \oplus [M^n_2]\right)^n \quad \text{by Remark 12}\]
This, together with (2) proves the thesis for the case \( k=2 \).

Now we consider the case \( k>2 \); first, we need an observation: let \( \phi \) be an allowed formula, and \( P \) and \( Q \) be modules such that \( P \oplus Q \) exists, then for any integer \( n \), the following syntactic equality holds

\[
\phi \circ ([P^n] \oplus [Q^n]) \equiv \phi \circ [P^n \circ [Q^0]] \circ [Q^n].
\]

The proof of (3) is immediate from the definitions. We now proceed with the inductive step: we assume that the thesis holds for \( k \) or less modules, and we prove it for \( k+1 \) modules. Let \( N \equiv M_1 \oplus \cdots \oplus M_k \). Then,

\[
\begin{align*}
M_1 \oplus \cdots \oplus M_{k+1} \cup \Delta \models \phi & \quad \text{Let } N \equiv M_1 \oplus \cdots \oplus M_k \\
\text{iff } N \oplus M_{k+1} \cup \Delta \models \phi & \quad \text{inductive hypothesis} \\
\text{iff } \exists n : [N^n] \oplus [M_{k+1}^n] \cup \Delta \models \phi & \quad \text{Lemma 5} \\
\text{iff } \exists n : \Delta \models \phi \circ ([N^n] \oplus [M_{k+1}^n]) & \quad \text{by (3)} \\
\text{iff } \exists n : \Delta \models \psi \circ [N^n] & \quad \text{Lemma 5} \\
\text{iff } \exists n : [N^n] \cup \Delta \models \psi & \quad \text{Theorem 6} \\
\text{iff } \exists n : N \cup \Delta \models \psi & \quad \text{inductive hypothesis} \\
\text{iff } \exists n,m : [M_1^m] \oplus \cdots \oplus [M_k^m] \cup \Delta \models \psi & \quad \text{Let } N' \equiv ([M_1^m] \oplus \cdots \oplus [M_k^m]) \\
\text{iff } \exists n,m : N' \cup \Delta \models \psi & \quad \text{Lemma 5} \\
\text{iff } \exists n,m : \Delta \models \psi \circ N' & \quad \text{Lemma 5} \\
\text{iff } \exists n,m : \Delta \models \phi \circ [M_{k+1}^n \circ [N^0]] \circ N' & \quad \text{since } [N^0] \equiv [N^0] \text{ and by (3)} \\
\text{iff } \exists n,m : \Delta \models \phi \circ (M_{k+1}^n \oplus N') & \quad \text{Lemma 5} \\
\text{iff } \exists n,m : M_{k+1}^n \oplus N' \cup \Delta \models \phi & \quad \text{Lemma 5} \\
\text{iff } \exists n,m : [M_1^m] \oplus \cdots \oplus [M_k^m] \oplus [M_{k+1}^n] \cup \Delta \models \phi \\
\text{take } n \equiv \sup(m,n), \text{ by Remark 12 and Lemma 11} \\
\text{iff } \exists n : [M_1^n] \oplus \cdots \oplus [M_{k+1}^n] \cup \Delta \models \phi. \quad \square
\end{align*}
\]

Notice that, if \( M \) is a module, then \( [M^n] \) is a collection of formulae of the form \( p(\bar{x}) \Leftrightarrow \phi[\bar{x}] \), where \( \phi[\bar{x}] \) contains only external predicates, i.e. open or base predicates (for instance, in \([M^n]\), recursion is impossible). In a way, we could say that each \([M^n]\) is an elementary module; using this notation the above theorem states that the semantics of a module \( M \) is given by the \( \preceq_\Delta \)-increasing sequence of elementary mod-
ules \([M^0],[M^1],[M^2],\ldots\). This construction can be seen as the modular counterpart of the usual construction of the Least Herbrand Model obtained via the operator \(T_P\) (see [1]). Indeed, in order to give an intuition (a more formal treatment of this matter will be done in Section 4), we can anticipate that if \(P\) is a closed definite logic program, and \(\text{Comp}(P)\) is its completion (which includes an appropriate base theory and domain closure axioms), we have that for any ground atomic formula \(A\), and for each \(n, A \in T_P \uparrow n\) if and only if \([\text{Comp}(P)^n] \models A\).

**Example 15.** The following program, given a directed graph, verifies whether a certain node is critical, i.e. whether by removing that node from the graph, some other nodes in the network become disconnected. We assume that the graph is represented in a module \(M_g\). This module defines only the predicate \(arc/2\) in such a way that \(arc(x, y)\) is true in \(M_g\) iff there is a (direct) link from \(x\) to \(y\) in the graph. Further, we have a module \(M_p\) which, referring to \(arc/2\) as an open predicate, defines the predicate \(path/3\) as follows:

\[
\text{path}(x, z, a) \iff arc(x, z) \lor \\
\exists y \ arc(x, y) \land \neg \text{member}(y, a) \land \text{path}(y, z, [y|a])
\]

Thus, \(\text{path}(x, y, a)\) is true iff there exists an acyclic path from \(x\) to \(y\) that avoids all the nodes in \(a\). The predicate \(\text{member}/2\) is assumed to be defined in the usual way in a separate module \(M_m\). Finally, we have a module \(M_c\) that defines the predicate \(\text{critical}/1\); it contains the single definition

\[
\text{critical}(x) \iff \exists y, z \ x \neq y \land x \neq z \land \text{path}(y, z, [\ ] \lor \neg \text{path}(y, z, [x])
\]

which states that \(x\) is critical if we can find a path from some node \(y\) to some node \(z\), both different from \(x\), but we cannot find a path from \(y\) to \(z\) that avoids \(x\). If we want to compute critical nodes of different graphs, we compose this module with different graph modules. Now, let us see how these modules behave under unfolding. We begin with module \(M_p\). The following table shows the definition of \(\text{path}/3\) in \(M_p^0\), in \(M_p^1(\equiv M_p)\) and in \(M_p^2\). In \(M_p^0\) the definition of \(\text{path}/3\) is

\[
\text{path}(x, z, a) \iff \text{path}(x, z, a)
\]

In \(M_p^1\) the definition of \(\text{path}/3\) is

\[
\text{path}(x, z, a) \iff \text{arc}(x, z) \lor \\
\exists y \ \text{arc}(x, y) \land \neg \text{member}(y, a) \land \text{path}(y, z, [y|a])
\]

While in \(M_p^2\) the definition of \(\text{path}/3\) becomes

\[
\text{path}(x, z, a) \iff \text{arc}(x, z) \lor \\
\exists y \ \text{arc}(x, y) \land \neg \text{member}(y, a) \land \\
\text{arc}(y, z) \lor \\
\exists y' \ \text{arc}(y, y') \land \neg \text{member}(y', [y|a]) \land \text{path}(y', z, [y'[y|a]])
\]
Thus, since in the above clauses \( \text{path} \) is the only non-open predicate, the definition of \( \text{path}/3 \) in \([M^0_p]\), in \([M^1_p]\) and in \([M^2_p]\) can simply be obtained by replacing the constant \( u \) all the body atoms in the above table which have \( \text{path} \) as predicate symbol as follows. In \([M^0_p]\) the definition of \( \text{path}/3 \) is

\[
\text{path}(x, z, a) \iff u
\]

In \([M^1_p]\) the definition of \( \text{path}/3 \) is

\[
\text{path}(x, z, a) \iff \text{arc}(x, z) \lor \exists y \, \text{arc}(x, y) \land \neg \text{member}(y, a) \land u
\]

In \([M^2_p]\) the definition of \( \text{path}/3 \) is

\[
\text{path}(x, z, a) \iff \text{arc}(x, z) \lor \\
\exists y \, \text{arc}(x, y) \land \neg \text{member}(y, a) \land \\
\left( \text{arc}(y, z) \lor \\
\exists y' \, \text{arc}(y, y') \land \neg \text{member}(y', [y\vert a]) \land u \right)
\]

Finally, it is worth noticing that, since the body of the definition of \( \text{critical}/1 \) does not contain any non-open predicate, we have that, for all \( n, M_c \equiv M^n_c \equiv [M^n_c] \).

4. Normal (constraint) logic programs

We now show how the results provided in the previous section may be used in order to provide a compositional semantics to modular normal logic programs (i.e. modular logic programs with negation). Intuitively, this is done as follows: given a module \( M \), we refer to its (Clark's) completion \( \text{Comp}(M) \) [10] together with an appropriate base theory consisting of the equality axioms, Clark's Equality Theory and, possibly, the Domain Closure Axiom. Since \( \text{Comp}(M) \) is a first order-module, Theorem 14 will provide an appropriate semantics.

**Notation.** Normal Modules are finite collections of normal clauses, that is, expressions of the form \( A \leftarrow L_1 \land \cdots \land L_m \), where \( A \) is an atom and each \( L_i \) is a literal (i.e. an atom or a negated atom). We also adopt the usual logic programming notation that uses "\( \lor \)" instead of \( \land \), hence a conjunction of literals \( L_1 \land \cdots \land L_n \) will be denoted by \( L_1, \ldots, L_n \) or by \( L \). It is worth noticing that – unlike in the first-order case – in a normal module there might be two or more clauses defining the same predicate symbol (i.e. with the same predicate symbol in the head).

In this context the only predicate symbol contained in the base language \( \mathcal{L}_B \) is the equality predicate \( = \). Thus, following the same notation used for first-order modules, if \( M \) is a normal module, we denote by \( \text{Def}(M) \) the set of predicates defined in \( M \), i.e. occurring in the head of at least one clause of \( M \). Further \( \text{Pred}(M) \) denotes the set
of all predicates occurring in $M$ (with the exception of the equality symbol) and, as before, the set of open predicates of $M$ is defined as $\text{Open}(M) \equiv \text{Pred}(M) \setminus \text{Def}(M)$.

The concept of program completion was introduced by Clark in [10] in order to provide a sound semantics for normal programs. Referring to the programs completion is by now a standard approach, and among the “standard” approaches it is the only one that allows one to remain within first-order logic. When dealing with three-valued logic the definition of completion is given using the operator $\leftrightarrow$ instead of $\rightarrow$, as follows.

**Definition 16.** Let $M$ be a normal module and $p(i_1) \leftarrow B_1, \ldots, p(i_r) \leftarrow B_r$ be all the clauses which define the predicate symbol $p$ in $M$. The completed definition of $p$ is

$$p(\bar{x}) \iff \bigvee_{i=1}^{r} \exists \bar{y}_i \ (\bar{x} = i_i) \land B_i,$$

where $\bar{x}$ are new variables and $\bar{y}_i$ are the variables in $p(i_i) \leftarrow B_i$.

The completion of $M$, $\text{Comp}(M)$ consists in the conjunction of the completed definition of all the predicates in $\text{Def}(M)$.

It is important to notice that here we depart from [10] in the fact that we don’t close those definitions which are not explicitly given in $M$. In a modular context, these predicates need to remain open. The completed definition of a predicate is a first order formula that contains various function symbols and the equality symbol; hence, in order to interpret it correctly, we also need an appropriate theory. In particular, following the literature, we will refer to $\text{CET}_{L_B}$, Clark’s Equality Theory for the language $L_B$, which consists of the following axioms:

- $f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$ for all distinct $f$ and $g$ in $L_B$;
- $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \rightarrow (x_1 = y_1) \land \cdots \land (x_n = y_n)$ for all $f$ in $L_B$;
- $x \neq t(x)$ for all terms $t(x)$ distinct from $x$ in which $x$ occurs together with the usual equality axioms, i.e. reflexivity, symmetry, transitivity, and $(\bar{x} = \bar{y}) \rightarrow (f(\bar{x}) = f(\bar{y}))$ for all functions symbols $f$ in $L_B$. Notice that “=” is always interpreted as two valued. Obviously, $\text{CET}_{L_B}$ depends on the underlying language $L_B$, which we assume to be fixed and to contain all the functions symbols occurring in all the modules we consider.

A known problem that semantics based on program completion face is that when $L_B$ is finite (that is, when it contains only a finite number of functions symbols) $\text{CET}_{L_B}$ is not a complete theory (see [31]). Typically, this problem is solved by adopting one of the following solutions: (a) adding to $\text{CET}_{L_B}$ some domain closure axioms which are intended to restrict the interpretation of the quantification to $L_B$-terms as in [31]), or (b) assuming that the language contains always an infinite set of function symbols (as in [20]) or (c) by considering only interpretations and models over a specific fixed domain $D$ as in [13]). This latter solution requires the adoption of axioms which are usually not first order (unless all the functions symbols are 0-ary, i.e. constants), and consequently leads to a semantics which is (usually) noncomputable. For these reasons
we adopt either solutions (a) or (b). Luckily, these two solutions yield essentially the same semantics. For an extended discussion of this subject, we refer to [20, 31].

We need one last definition. Let $\mathcal{L}_B$ be a finite language (i.e. a language with a finite set of predicate symbols). The **Domain Closure Axiom** for the language $\mathcal{L}_B$, $\text{DCA}_{\mathcal{L}_B}$, is

$$\exists y_1 (x = f_1(y_1)) \lor \cdots \lor \exists y_r (x = f_r(y_r))$$

where $f_1, \ldots, f_r$ are all the function symbols in $\mathcal{L}_B$ and $y_1$ are tuples of variables of the appropriate arity. This axiom is also referred to as the weak domain closure axiom.\(^4\)

4.1. A compositional semantics for normal programs

It is now easy to see that in this context, the semantics for open normal logic modules finds a natural embedding in the one proposed for first order modules in Section 3. Module composition is defined exactly as for the case of first-order modules as follows: if $M_1$ and $M_2$ are normal modules we define $M_1 \oplus M_2 = M_1 \cup M_2$ provided that $\text{Def}(M_1) \cap \text{Def}(M_2) = \emptyset$ holds. Otherwise $M_1 \oplus M_2$ is undefined. Our main result becomes:

**Theorem 17** (Main, for modular logic programs). Let $M_1, \ldots, M_k$ be normal modules such that $M_1 \oplus \cdots \oplus M_k$ is defined. Then, for each allowed $\phi$ there exists an integer $n$ such that the following statements are equivalent:

1. $\text{Comp}(M_1 \oplus \cdots \oplus M_k) \cup \text{CET}_{\mathcal{L}_B} \models \phi$
2. $[\text{Comp}(M_1)^n] \oplus \cdots \oplus [\text{Comp}(M_k)^n] \cup \text{CET}_{\mathcal{L}_B} \models \phi$

where we assume that, if $\mathcal{L}_B$ is finite, $\text{CET}_{\mathcal{L}_B}$ incorporates $\text{DCA}_{\mathcal{L}_B}$.

As an example, let us consider again the problem of deciding whether a node in a graph is critical. The program given in the previous section can also be written as a modular normal program composed by the modules defining $\text{arc}$, $\text{member}$, together with the following two modules:

$$N_p \equiv \begin{cases} 
\text{path}(x, z, a) & \leftarrow \text{arc}(x, z) \\
\text{path}(x, z, a) & \leftarrow \text{arc}(x, y), \neg \text{member}(y, a), \text{path}(y, z, [\text{y} | a])
\end{cases}$$

$$N_c \equiv \begin{cases} 
\text{critical}(x) & \leftarrow x \neq y, \ x \neq z, \ \text{path}(y, z, [\text{y}]), \ \neg \text{path}(y, z, [\text{x}])
\end{cases}$$

In fact it is immediate to check that $M_p$ and $M_c$ coincide with the completion of $N_p$ and $N_c$.\(^4\)

\(^4\) As opposed to it, the strong domain closure axiom for the language $\mathcal{L}_B$ is $x = t_1 \lor x = t_2 \lor \cdots$ where $t_1, t_2, \ldots$ is the (usually infinite) sequence of all the ground $\mathcal{L}_B$-terms. This axiom is equivalent to choice (c) above, and determines uniquely the universe of the possible interpretation. Again, if $\mathcal{L}_B$ contains a non-constant function symbol then the above axiom is not a first order formula, and leads to a non-computable semantics.
Compositionality vs. non-monotonicity

In a proof-theoretic interpretation of logic programming a resolution method (resp. a semantics) can be viewed as an inference relation, which maps a program into the set of atoms which can be derived from it. In the literature, an inference relation $\models_{\text{INF}}$ is called monotonic iff it satisfies the following rule:

$$\Gamma \models_{\text{INF}} \phi \text{ implies that } \Gamma \cup \Gamma' \models_{\text{INF}} \phi.$$  \hfill (4)

Now, most semantics associated to logic programs with negation (induce inference relations which) are non-monotonic. For instance the inference relation $\models_{\text{SLDNF}}$ induced by the operational semantics provided by the SLDNF resolution method determines the following behaviour: $\{p \leftarrow \neg q\} \models_{\text{SLDNF}} p$ while $\{p \leftarrow q. q.\} \not\models_{\text{SLDNF}} p$. This reasoning applies also to (the inference relations induced by) virtually all declarative semantics; among them, Kunen’s and Fitting’s semantics. Non-monotonicity is actually a crucial aspect of normal programs and has greatly contributed to the popularity of the paradigm. As remarked in [2]: “the best argument for non-monotonic semantics of logic programs with negation is that non-monotonic logics, i.e. logics dealing with non-monotonic inference relations, are very useful, and that logic programming with negation can help in implementing them”. Thus, non-monotonicity is an aspect of logic programs with negation which we should not abandon. On the other hand – as we have stressed in the introduction – in the formulation of a semantics for modular logic programs, compositionality plays a crucial role. This raises a conflict: in fact it is immediate to see that compositionality implies that the semantics has to be – to some extent – monotonic.

In our framework, we manage to combine the two aspects by separating their domains: within a module the addition of a clause remains a non-monotonic operation, while at meta-level module’s composition is a monotonic one. Let us see a simple example of this fact, and consider a normal module

$$M \equiv \{q(a)\}$$

with $\mathcal{L}_B$ consisting of equality and the constants $a$ and $b$. Now, suppose we want to add to the database the fact that $q(b)$ holds. If we do this by simply adding a clause to $M$ we have a nonmonotonic behavior, which implies a defeat of compositionality, for instance, if we let $Q \equiv \{p \leftarrow \neg q(b)\}$, we have that

- $\text{Comp}(M) \oplus \text{Comp}(Q) \cup \text{CET}_{\mathcal{L}_B} \models p$, while
- $\text{Comp}(M \cup \{q(b)\}) \oplus \text{Comp}(Q) \cup \text{CET}_{\mathcal{L}_B} \not\models p$

Now, it is important to notice that we are not allowed to add the clause $q(b)$ via a module composition operation. In fact $M \oplus q(b)$ is not defined, as the condition on name clashes is violated. If we wanted to be able to add the knowledge $q(b)$ via a module composition operation (thus in a compositional way) we would have had to start with a modified version of $M$, namely with the following:

$$N \equiv \begin{cases} q(a), \\ q(x) \leftarrow q'(x). \end{cases}$$
Here the predicate $q'$ is an open predicate which can be used to extend our knowledge on $q$. Now, $N \oplus \{q'(b)\}$ is defined and going from $N$ to $N \oplus \{q'(b)\}$ we have a monotonic behavior. In fact

- $\text{Comp}(N) \oplus \text{Comp}(Q) \cup \text{CET}_2 \not\models p$
- $\text{Comp}(N) \oplus \text{Comp}(\{q'(b)\}) \oplus \text{Comp}(Q) \cup \text{CET}_2 \not\models p$

In our framework, the negation-as-failure mechanism can still be profitably employed in a non-monotonic manner, as long as the negated atom and its descendants in the proof tree are not open. This has to be so: the failure of proving an atom whose proof tree could be augmented by module's composition cannot be taken as "sufficient evidence" for assuming true the negation of the atom itself (as usually done by negation as failure). It is worth noticing that it is easy to extend the negation as failure mechanism in order to force it to take into account the presence of open atoms.

### 4.2. Normal CLP modules

The Constraint Logic Programming paradigm (CLP for short) has been proposed by Jaffar and Lassez [17] in order to integrate a generic computational mechanism based on constraints with the logic programming framework. Such an integration results in a framework which - for programs without negation - preserves the existence of equivalent operational, model-theoretic and fixpoint semantics. Indeed, as discussed in [24], most of the results which hold for definite (i.e., negation-free) logic programs can be lifted to CLP in a quite straightforward way. As we will shortly see, when negation is involved, such a lifting might present some difficulties.

We refer to the recent survey [18] by Jaffar and Maher for the notation and the necessary background material about CLP. A CLP program is a collection of CLP clauses which are formulae of the form $A \leftarrow c \land L_1 \land \cdots \land L_k$ where $A$ is an atom, $L_1, \ldots, L_k$ are literals and $c$ is a constraint, i.e., a first order formula in a specific language $\mathcal{L}_c$. Here there is no need to enter the details over the semantics of the paradigm (we refer to [18]); intuitively, from the operational point of view constraints are considered as built-ins and are handled by a constraint solver, while the "rest" (the logic part) serves exactly as a logic program. From the declarative point of view, the semantics of the constraints is determined in either one of the following two ways:

(a) by providing a consistent first-order base $\text{Theory}$, that their interpretation has to satisfy (e.g., Peano's arithmetic); or
(b) by giving a base structure $\mathbb{B}$ over which they are interpreted, (e.g., the natural numbers).

It is clear that if we follow the first approach then the results of the previous section can be naturally used to provide a compositional semantics to normal CLP modules. All we have to do is to incorporate in the base theory $\Delta$ the theory that provides a meaning to the constraints and to refer to the module completion (which is defined exactly as in the case of normal logic programs), and immediately obtain the following:

**Theorem 18.** Let $M_1, \ldots, M_k$ be normal CLP modules such that $M_1 \oplus \cdots \oplus M_k$ is defined. Assume that the meaning of the constraints is determined by a first-order
consistent theory $\Delta$. Then, for each allowed $\phi$ there exists an integer $n$ such that the following statements are equivalent:

- $\text{Comp}(M_1 \oplus \cdots \oplus M_k) \cup \Delta \models \phi$,
- $[\text{Comp}(M_1)^n] \oplus \cdots \oplus [\text{Comp}(M_k)^n] \cup \Delta \models \phi$,

where we assume that $\Delta$ incorporates the equality axioms.

Thus if we follow choice (a) above our results apply with almost no modification. Regrettably, approach (b) is certainly more popular in the CLP community (even though also the first one is considered standard (see [18])). The problem with the latter approach is that the given structure determines uniquely the universe of the models, and this – in the presence of negation – leads to a semantics which is again usually noncomputable. As already done in [20, 30], we can avoid this problem by referring to some elementary extension of the given structure itself. This will be done in the next section.

5. If the interpretation of constraints is determined by a structure

In the previous sections we have always assumed that the interpretation of the base predicates was determined by a first-order base theory $(A)$. Now, as already mentioned in Section 4.2, this is not the only possible approach; in the literature we find situations in which the interpretation of the base predicates is provided by a suitable structure. In particular, this happens frequently in the case of CLP normal modules. In this section we are going to show how also in this different setting it is possible to obtain a computable compositional semantics (indeed a counterpart of Theorem 14). The task is not trivial: firstly because in order to obtain a computable semantics we have to resort to the use of an elementary extension of the given structure, and, secondly, because there’s much more machinery involved in the proofs.

Notation. Let us first establish some notation. Let $M$ be a first-order module on the language $\mathcal{L}_B$, and assume that the meaning of the base predicates is determined by a base structure $\mathbb{B}$, then the models of $M$ we will be allowed to consider are only those that share with $\mathbb{B}$ the universe and the interpretation of the base predicates. Such models are called expansions of $\mathbb{B}$, or $\mathbb{B}$-models according to the following definition. Let $\mathbb{B} \equiv \langle \text{Dom}, \text{Rel}, \text{Fun} \rangle$ be a structure for the language $\mathcal{L}_B$, and let $\mathbb{S} \equiv \langle \text{Dom}', \text{Rel}', \text{Fun} \rangle$ be a structure for $\mathcal{L}_B \cup \{p_1, \ldots, p_k\}$, where $\{p_1, \ldots, p_k\}$ are new predicate symbols. We say that $\mathbb{S}$ is a $\mathbb{B}$-structure iff $\mathbb{S}$ is a conservative expansion of $\mathbb{B}$, i.e., if $\text{Rel}'_{\mathbb{S}} \equiv \text{Rel}$. Similarly, we say that $\mathbb{M}$ is a $\mathbb{B}$-model of $M$ if it is both a model of $M$ and an $\mathbb{B}$-structure. Let $M$ be a first-order module on the base language $\mathcal{L}_B$. We say that the first-order formula $\phi$ follows from $M$ wrt. the structure $\mathbb{B}$

$$M \models_{\mathbb{B}} \phi$$

if $\text{Val}(\phi, \mathbb{M}) = \text{true}$ for every $\mathbb{B}$-model $\mathbb{M}$ of $M$, i.e., if $\phi$ is true in all the models of $M$ whose universe coincides with $\text{Dom}$, and whose interpretation of functions and predicates in $\mathcal{L}_B$ coincides with the one given by $\mathbb{B}$. 


A motivating example. Recall the definition of module OddEven given in Example 1 and consider now the following first-order module NonStandard

\[ \text{non-standard } \equiv \exists x (\neg \text{odd}(x) \land \neg \text{even}(x)) \]

where \text{odd} and \text{even} are (by definition) considered open predicates. On one hand, if we refer to the setting used in the previous sections and we let the interpretation of constraints to be determined by a first-order theory of arithmetics \( A \), we have that

\[ \text{OddEven } \oplus \text{NonStandard } \cup A \not\models \neg \text{non-standard}. \]

On the other hand, if the interpretation of the constraint is determined by the (standard) structure \( \mathbb{N} \), and thus the semantics is determined by the relation \( \models_{\mathbb{N}} \), we have that

\[ \text{OddEven } \oplus \text{NonStandard } \models_{\mathbb{N}} \neg \text{non-standard}. \]

This second situation is highly undesirable because the falsity of \text{non-standard} is not computable (one would need \( \omega + 1 \) inference steps in order to determine it). In general, given a structure \( \mathcal{B} \), the relation \( \models_{\mathcal{B}} \) is not computable. This immediately implies that with the tools we have introduced in this paper there is no possibility that we can appropriately model the semantics induced by \( \models_{\mathcal{B}} \). In order to solve this problem, we can refer to an elementary extension of the given of \( \mathcal{B} \) itself. We need the following definition.

**Definition 19 (Elementary extension).** Let \( \mathcal{B} \equiv (\text{Dom}, \text{Rel}, \text{Fun}) \) and \( \mathcal{B}' \equiv (\text{Dom}', \text{Rel}', \text{Fun}') \) be two structures. We say that \( \mathcal{B}' \) is an elementary extension of \( \mathcal{B} \) if \( \text{Dom}' \supseteq \text{Dom} \) and, for any allowed formula \( \phi[x] \) in \( \mathcal{L}_{\mathcal{B}} \), we have that \( \text{Val}(\phi[t], \mathcal{B}) \equiv \text{Val}(\phi[t], \mathcal{B}') \), for any \( t \in \text{Dom} \).

Therefore, if \( \mathcal{B}' \) is an elementary extension of \( \mathcal{B} \) then reasoning over \( \mathcal{B}' \) is basically just like reasoning over \( \mathcal{B} \); the only difference is that in \( \mathcal{B}' \) we have more "witnesses" and thus universally quantified formulas might assume a different truth value. Notice that, if we take any non-trivial extension \( \mathbb{N}' \) of \( \mathbb{N} \), we immediately have that \( \text{OddEven } \oplus \text{NonStandard } \not\models_{\mathbb{N}'} \neg \text{non-standard} \); we might well say that the falsehood of \text{non-standard} in the \( \mathbb{N} \)-models of \( \text{OddEven } \oplus \text{NonStandard} \) is determined by the limits of the universe of \( \mathbb{N} \).

5.1. Further preliminaries: Fitting's operator revisited

As we have mentioned before, the proofs we are going to provide will need some additional preliminary notions. In particular we are now going to revisit Fitting's results [13], and define a modular version of Fitting's operator. The results we are going to state in this subsection are not new (unless otherwise specified they are – more or less – immediate extensions of the results of [13]), but will be needed in the sequel. We
state them for the sake of self-containedness and for maintaining a consistent notation throughout the paper. We start with a modular version of Fitting's operator.

**Definition 20.** Let $M$ be a module over $\mathcal{L}_B$. If $S \equiv \langle \text{Dom}, \text{Rel}, \text{Fun} \rangle$ is a structure for a language $\mathcal{L} \supseteq \mathcal{L}_B$, then $\Phi_M(S)$ is the structure $\langle \text{Dom}, \text{Rel}', \text{Fun} \rangle$, for the language $\mathcal{L} \cup \text{Def}(M)$ defined as follows:

(a) if $\text{Pred}(p) \in \text{Def}(M)$ then $\text{Val}(p(\bar{i}), \Phi_M(S))$ is true (resp. false) iff there exists a definition $p(\bar{x}) \leftrightarrow \phi[\bar{x}] \in M$ and $\text{Val}(\phi[\bar{i}], S)$ is true (resp. false).

(b) if $\text{Bed}(p) \notin \text{Def}(M)$ then $\text{Val}(p(\bar{i}), \Phi_M(S)) \equiv \text{Val}(p(\bar{i}), S)$.

The main difference between this definition and the one provided in [13] lies in presence of rule (b) and the fact that here $\Phi$ is an operator on the poset of structures, while in [13] it is a mapping over the poset of interpretations over a specific language. Notice that $\Phi_M$ leaves unchanged the interpretation of the base predicates and functions. This implies the following remark, whose proof is immediate.

**Remark 21.** Let $B$ be a structure for $\mathcal{L}_B$. In the notation of Definition 20, if $S$ is a $B$-structure then $\Phi_M(S)$ is a $B$-structure as well.

The key feature of the operator $\Phi$ is stated in the following theorem. The proof follows immediately from the definition of $\Phi$ and from the results in [13].

**Theorem 22.** A structure $S$ is a model of $M$ iff it is a fixpoint of $\Phi_M$.

Thanks to the particular (modular) definition of $\Phi$, this result applies also in a compositional fashion.

**Theorem 23.** Let $M$ and $N$ be modules over $\mathcal{L}_B$ such that $M \oplus N$ is defined. Then a structure $S$ is a model of $M \oplus N$ iff $S$ is a fixpoint of $\Phi_M \Phi_N$, i.e., if $S = \Phi_M(\Phi_N(S))$.

**Proof.** Follows immediately from the previous theorem and the facts that (a) $\text{Def}(M) \cap \text{Def}(N) = \emptyset$ and (b) $S$ is a model of $M \oplus N$ iff it is a model of both $M$ and $N$. 

We now need to provide a semantic-based order on structures. Notice that in a structure $\langle \text{Dom}, \text{Rel}, \text{Fun} \rangle$, we can assume that $\text{Rel}$ is represented as a set of elements of the form $p(\bar{i})$ or $\neg p(\bar{i})$, where $p$ is a relation symbol and $\bar{i}$ is a tuple of elements of $\text{Dom}$ (of course we have to assume that $\text{Rel}$ never contains $p(\bar{i})$ and $\neg p(\bar{i})$ at the same time). Thus the notation $\text{Rel} \subseteq \text{Rel}'$ is meaningful. Now, given two structures $\langle \text{Dom}, \text{Rel}, \text{Fun} \rangle$ and $\langle \text{Dom}', \text{Rel}', \text{Fun}' \rangle$ we say that

$$\langle \text{Dom}, \text{Rel}, \text{Fun} \rangle \subseteq \langle \text{Dom}', \text{Rel}', \text{Fun}' \rangle$$

iff $\text{Dom} \equiv \text{Dom}'$, $\text{Fun} \equiv \text{Fun}'$ and $\text{Rel} \subseteq \text{Rel}'$. Clearly, if $S \subseteq S'$ then we have that:

- if $S$ is a $B$-structure then $S'$ is a $B$-structure as well;
for any allowed formula \( \phi \), if \( \text{Val}(\phi, S) = \text{true} \) (resp. \( \text{false} \)) then \( \text{Val}(\phi, S') = \text{true} \) (resp. \( \text{false} \)) as well;
- \( \Phi_M(S) \subseteq \Phi_M(S') \), i.e., \( \Phi_M \) is a monotonic operator.

Let \( B \) be a structure for the language \( L_B \), we adopt the following notation
- \( \Phi^\beta_M(B) \equiv \text{the trivial conservative expansion of } B \text{ to the language } L_B \) in which for each \( p \) in \( \text{Def}(M) \setminus L_B \) we have that, for all \( \bar{t} \), \( \text{Val}(p(\bar{t}), B) \equiv \text{undefined} \);
- \( \Phi^{\alpha+1}_M(B) \equiv \Phi_M(\Phi^\alpha_M(B)) \);
- \( \Phi^\gamma_M(B) \equiv \cup_{\beta<\gamma} \Phi^\beta_M \) (see next note) when \( \gamma \) is a limit ordinal.

Of course, we have to show that this notation is consistent, i.e. that for each \( \alpha \), \( \Phi^\alpha_M \) is a \( B \)-structure. This is trivial if \( \alpha \) is a successor ordinal (the thesis follows from the definition of \( \Phi \)), but requires more caution when \( \alpha \) is a limit ordinal. In fact we need the following.

**Note 1.** Let \( M \) be a module over \( L_B \), \( B \) be a structure for \( L_B \) and \( \alpha \) be an ordinal. We have that

(i) if \( \beta<\alpha \) then \( \Phi^\beta_M \subseteq \Phi^\alpha_M \),
(ii) \( \Phi^\alpha_M \) is a structure.

**Proof** (sketch). The proof proceeds by induction on \( \alpha \). For the base case (\( \alpha = 1 \)), (i) follows from the fact that \( \text{Def}(M) \cap \text{Pred}(B) = \emptyset \) and therefore that \( \Phi^0_M(B) \subseteq \Phi^1_M(B) \); (ii) follows immediately from the definition of \( \Phi_M \). For the induction step, if \( \alpha \) is a successor ordinal then (i), and (ii) are immediate consequences of the definition and of the monotonicity property of \( \Phi_M \). If \( \alpha \) is a limit ordinal then (i) is immediate, while in order to prove (ii) we have to show that, assuming that \( \Phi^\alpha_M \equiv \langle \text{Dom}, \text{Rel}, \text{Fun} \rangle \), there exists no \( p(\bar{t}) \) such that both \( p(\bar{t}) \) and \( \neg p(\bar{t}) \) belong to \( \text{Rel} \). Now, by inductive hypothesis we know that \( \{ \Phi^\delta_M \mid \delta<\alpha \} \) is a (possibly transfinite) increasing chain; thus, proceeding by contradiction, if we had that \( p(\bar{t}) \in \text{Rel} \) and \( \neg p(\bar{t}) \in \text{Rel} \) we would find an ordinal \( \gamma<\alpha \) such that \( \Phi^\gamma_M = \langle \text{Dom}', \text{Rel}', \text{Fun}' \rangle \) and that \( p(\bar{t}) \in \text{Rel}' \) and \( \neg p(\bar{t}) \in \text{Rel}' \). This would contradict the inductive hypothesis. \( \square \)

From (i) and (ii) it follows immediately that for each \( \alpha \), \( \Phi^\alpha_M \) is a \( B \)-structure.

**Example 24.** Let us consider again module \( \text{OddEven} \), together with the structure \( \mathbb{N} \), we have that, concerning solely predicates \( \text{odd} \) and \( \text{even} \), the structures \( \Phi^n_{\text{OddEven}}(\mathbb{N}) \) determine the following interpretations:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n ) Instances of even (x) of odd (x) which are true or false in ( \Phi^n_{\text{OddEven}}(\mathbb{N}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none (everything is undefined)</td>
</tr>
<tr>
<td>2</td>
<td>even(0), ( \neg \text{odd}(0) )</td>
</tr>
<tr>
<td>3</td>
<td>even(0), ( \neg \text{odd}(0) ), ( \neg \text{even}(1) ), odd(1)</td>
</tr>
</tbody>
</table>

Finally, we have the following:
Theorem 25. Let $M$ be a module over $\mathcal{L}_B$, and let $\mathfrak{B}$ be a structure for $\mathcal{L}_B$. Then there exists an ordinal $\alpha$ such that $\Phi_\alpha^M$ is the least $\mathfrak{B}$-model of $M$, i.e. such that
- $\Phi_\alpha^M(\mathfrak{B})$ is a $\mathfrak{B}$-model of $M$;
- for any $\mathfrak{B}$-model $M$ of $M$, $\Phi_\alpha^M(\mathfrak{B}) \subseteq M$.
Consequently, for any allowed formula $\phi$, we have that
- $M \models_B \phi$ iff $\text{Val}(\phi, \Phi_\alpha^M(\mathfrak{B})) = \text{true}$.

Proof (sketch). By Note 5.1, we have that the sequence $\Phi_0^M(\mathfrak{B}), \Phi_1^M(\mathfrak{B}), \ldots, \Phi_\alpha^M(\mathfrak{B}), \ldots$ is monotonically increasing and, by well-known results on lattice theory, it converges to the $\subseteq$-least fixpoint $\Phi_\alpha^M(\mathfrak{B})$, which – by Theorem 22 – coincides with the $\subseteq$-least $\mathfrak{B}$-model of $M$. □

Since $\Phi_\alpha^M$ is monotonic but not continuous, in the above theorem, $\alpha$ could be greater than $\omega$, thus this semantics is in general uncomputable (this is also shown by the example at the beginning of this section). Thus, in order to obtain a computable semantics we have to resort to the concept of elementary extension of a structure. 5

Before doing so, we need two instrumental lemmata. The first one is a re-visit to Lemma 5; in fact its proof may be obtained as a straightforward translation of the one of Lemma 5.

Lemma 26. If $M$ is module, $[M] \models_B \phi$ iff $\text{Val}(\phi \circ [M]_B) = \text{true}$.

The second one shows that also in this context the unfolding operation, when applied to a closed module is correct, in the sense that it maintains the set of (allowed) logical consequence.

Lemma 27. Let $M_0, M_1, \ldots, M_n$ be a sequence of closed modules on the base language $\mathcal{L}_B$ such that for each $i \in [1, n]$ there exists definitions $c_i$ and $c'_i$ such that
(i) $c'_i$ is obtained from $c_i$ via an unfolding operation, using a definition of $M_0$ as unfolding definition.
(ii) $M_i = M_{i-1} \setminus \{c_i\} \cup \{c'_i\}$
Then, for any allowed formula $\phi$, $M_0 \models_B \phi$ iff $M_n \models_B \phi$.

Proof. By Theorem 25 it is sufficient to show that for each ordinal $\alpha$ there exists an ordinal $\beta$ such that $\Phi_\alpha^B(\mathfrak{B}) \subseteq \Phi_\beta^{M'}(\mathfrak{B})$ and $\Phi_\beta^{M'}(\mathfrak{B}) \subseteq \Phi_\alpha^B(\mathfrak{B})$. This is proved in [4, Lemma 9.1]. □

5 Diversion: It is interesting to notice that there exists a strict correspondence between the top-down construction provided by the sequence of unfoldings and the bottom-up one provided by $\Phi_\alpha$, namely we have that, for any allowed formula $\phi$, and for each natural $k$

$$\text{Val}(\phi, \Phi_\alpha^M(\mathfrak{B})) = \text{true} \text{ if and only if } [M^k] \models_B \phi$$

In fact, if we did not exceed $\omega$, we could avoid referring to $\Phi_\alpha$ altogether; unfortunately if $\alpha > \omega$ then $[M^\alpha]$ is not definable.
5.2. A compositional semantics

At last, we are able to show how the results of Section 3 may be restated for the case in which the interpretation of the base predicates is determined by a structure. First, we re-define a partial order based on the expressiveness of modules.

**Definition 28.** Let $M$ and $N$ be two modules on the same base language $\mathcal{L}_B$, and such that $\text{Def}(M) \equiv \text{Def}(N)$. Let $\mathbb{B}$ be a structure for $\mathcal{L}_B$. We say that

$M$ is **compositionally more expressive** than $N$ w.r.t. $\mathbb{B}$, $M \succ_{\mathbb{B}} N$,

iff for any other module $Q$ (on $\mathcal{L}_B$) such that $M \oplus Q$ and $N \oplus Q$ are defined we have that for any allowed formula $\phi$, if $N \oplus Q \models_{\mathbb{B}} \phi$ then $N \oplus Q \models_{\mathbb{B}} \phi$. We also say that

$M$ and $N$ are **compositionally equivalent** w.r.t. $\mathbb{B}$, $M \equiv_{\mathbb{B}} N$

iff $M \succ_{\mathbb{B}} N \succ_{\mathbb{B}} M$.

Again, the relation $\prec_{\mathbb{B}}$ is clearly an order relation. Lemma 9 extends immediately to this context.

**Lemma 29.** Let $M$, $N$ and $Q$ be modules such that $M \oplus Q$ is defined. If $M \triangleleft_{\mathbb{B}} N$ then $M \oplus Q \triangleleft_{\mathbb{B}} N \oplus Q$.

Again, it is easy to relate the relations $\to$ and $\succ_{\mathbb{B}}$.

**Lemma 30.** Let $M$ and $N$ be two modules on the base language $\mathcal{L}_B$, and $\mathbb{B}$ be a structure for $\mathcal{L}_B$. If $M \nrightarrow N$ then $M \succ_{\mathbb{B}} N$.

**Proof.** Assume that $M \nrightarrow N$. First, we prove the following: if $S$ is any structure for $\mathcal{L}_B \cup \text{Def}(M)$ then

$$\Phi_N(S) \subseteq \Phi_M(S) \quad (5)$$

Proof of (5). Take any $p(\bar{\bar{t}})$ which is true in $\Phi_N(S)$ (if $p(\bar{\bar{t}})$ is false in $\Phi_N(S)$ then the same reasoning applies). If $\text{Pred}(p) \not\subseteq \text{Def}(N)$ then, by the definition of $\Phi$, $p(\bar{\bar{t}})$ is true in $S$, and, since $\text{Pred}(p) \not\subseteq \text{Def}(M)$, it is also true in $\Phi_M(S)$. Otherwise, in $\text{Pred}(p) \subseteq \text{Def}(N)$, then there exists a definition $p(\bar{\bar{x}}) \leftrightarrow \phi(\bar{\bar{x}}) \in N$ such that $\text{Val}(\phi(\bar{\bar{t}}), S) \equiv \text{true}$. Since $M \nrightarrow N$, there exists a definition $p(\bar{\bar{x}}) \leftrightarrow \psi(\bar{\bar{x}}) \in M$, where $\phi$ is obtainable from $\psi$ by replacing with $u$ some of its subformulas. By Remark 10, we have that $\text{Val}(\psi(\bar{\bar{t}}), S) \equiv \text{true}$ as well. Thus, by the definition of $\Phi$, $p(\bar{\bar{t}})$ is true in $\Phi_M(S)$. Proof of (5) is completed.

Now, let $Q$ be any module such that $M \oplus Q$ is defined, then $N \oplus Q$ is defined as well and, by the monotonicity of $\Phi_Q$ and (5), for any $\mathbb{B}$-structure $S$, $\Phi_Q \Phi_N(S) \subseteq \Phi_Q \Phi_M(S)$. By the Theorem 23 and well-known results on lattice theory, this implies that, if $M$ and $N$ are the least $\mathbb{B}$-models of, respectively, $M \oplus Q$ and $N \oplus Q$, then $N \subseteq M$, hence the thesis. □
We are now ready for the main theorem of this section.

**Theorem 31 (Main 2).** Let $M_1, \ldots, M_k$ be first-order modules on the base language $\mathcal{L}_B$ such that $M_1 \oplus \ldots \oplus M_k$ is defined. Let also $B$ be a structure for $\mathcal{L}_B$. Then, there exists an elementary extension $B'$ of $B$ such that, for each allowed $\phi$ the following statements are equivalent:

- $M_1 \oplus \ldots \oplus M_k \models_B \phi$;
- $\exists n [M_1^n]\oplus \cdots \oplus [M_k^n] \models_{B'} \phi$.

**Proof.** We give here a simplified proof for the case in which $k = 2$, the extension to the general case can be done as in the proof of Theorem 14.

$(\Rightarrow)$ This implication holds for any structure $B'$. From Remark 12 we know that $M^n_1 \models [M_1^n]$ and therefore (via the same Remark) that $M^n_1 \oplus M^n_2 \models [M_1^n] \oplus [M_2^n]$, so by Lemma 30 if $[M_1^n] \oplus [M_2^n] \models_{B'} \phi$ then $M_1^n \oplus M_2^n \models_{B'} \phi$. Therefore, by the correctness of the unfolding operation, Lemma 27 and Lemma 29 $M_1 \oplus M_2 \models_{B'} \phi$.

$(\Leftarrow)$ By Theorem [30, Lemma 3.21 and Lemma 26], we have that there exists an elementary extension $B'$ of $B$ such that, for any allowed formula $\phi$ there exists an integer $n$ such that

$$M_1 \oplus M_2 \models_{B'} \phi \quad \text{implies that} \quad [(M_1 \oplus M_2)^n] \models_{B'} \phi$$  \hfill (6)

Now,

$$[(M_1 \oplus M_2)^n] \quad \text{by Lemmata 13 and 30}$$

$$\leq_{B'} [(M_1^n \oplus M_2^n)^n] \quad \text{by Remark 12}$$

$$\leq_{B'} (M_1^n \oplus M_2^n)^n \quad \text{by Lemma 27}$$

$$\sim_{B'} [M_1^n] \oplus [M_2^n]$$

This, together with (6) proves the thesis. \(\square\)

As shown in [20], given $\mathcal{L}_B$ and $B$, one can actually build\(^6\) a $B'$ for which this theorem holds. Intuitively, the basic idea is that the domain of $B'$ has to be sufficiently rich to avoid the problem of running out witnesses, and this can be guaranteed by letting $B'$ be an $\aleph_1$-saturated structure.

6. Conclusions

In this paper we have proposed a semantics for first order programs which is compositional with respect to the module composition $\oplus$ operator. This semantics is built via a first-order unfolding operator and allows to characterize (compositionally) the set of logical consequences of the module in three valued logics. Further, we have

---

\(^6\) This is done via ultrapower construction. Thus $B'$ is not actually computable.
shown how our results may be applied to modular normal programs and normal CLP. The semantics we have proposed may be regarded as a compositional counterpart of Kunen's semantics for normal programs [20] and its first-order version due to Sato [30].

Another recent proposal for a compositional semantics for logic programs is the one of G. Ferrand and A. Lallouet [12]. In their paper, Ferrand and Lallouet propose two compositional semantics, one based on Fitting's model semantics [13] and one based on the Well-Founded semantics [32]. The notion of program unit they use is similar to our notion of module. The main differences between their proposal and ours are simply the projections of the differences between Kunen's semantics and Fitting's model (resp. the Well-Founded) semantics. Basically, both in Fitting's model semantics and in the Well-Founded semantics we have that interpretations and models are restricted to a fixed universe (typically, the Herbrand universe of the program). As a result, these semantics cannot be axiomatized within first-order logics and are in general noncomputable (they may require more than \(\omega\) iterations in order to be built). Indeed, Fitting's semantics coincides with the relation \(\models_B\) we have introduced and discussed at the beginning of Section 5. For instance, using the example program of Section 5, both in Fitting's model and in the Well-Founded semantics of OddEven + NonStandard the interpretation of the atom \(\text{nonstandard}\) is false. Actually, the Well-Founded semantics is even more "distant" from our system than Fitting's model semantics is. For a insightful comparison, we refer to [2,11]. Concerning the methodology employed in order to achieve compositionality, in [12] the semantics of a module \(M\) is defined in a natural way as the function which maps the interpretation \(S\) of the imported literals into the (Fitting's or Well-Founded) model of \(M\) containing \(S\). In our opinion, the main disadvantage of this approach is that it is uncomputable. In contrast, our semantics for modular normal and first-order logic programs is based upon arbitrary three-valued models and characterized by a countably infinite sequence of approximations, and is thus recursively enumerable.

In [24], M. Maher presents a transformation system for normal programs and a compositional version of the Perfect Model Semantics. From the point of view of modularity the main difference between this paper and [24] is that in [24] modules are required to have a hierarchical calling pattern. Namely, mutual recursion among modules is prohibited (this can be seen as a consequence of the fact that the Perfect Model Semantics itself requires the program to be stratified). From the semantical point of view, the differences between this paper and [24] originate from the differences between the Perfect Model Semantics and Kunen's semantics. First of all, the first is based on two-valued logics, and imposes some restriction on the syntax of modules: programs are required to be stratified or locally stratified, which — intuitively speaking — means that recursion "through" negation is to some extent prohibited. Furthermore, in the Perfect Model Semantics the rule for inferring falsity is the Closed World Assumption: "if \(A\) cannot be proved, infer \(\neg A\);" for instance, we have that \(p\) is false in the Perfect Model of the program \(\{p \leftarrow p\}\). As opposed to this, in Kunen's semantics \(\neg A\) is inferred iff there is a proof for \(\neg A\) in \(\text{Comp}(P) \cup \text{CET}\), (and \(p\) is, in the above program,
undefined). As a consequence, the Perfect Model Semantics is again in general not computable.

Acknowledgements

I would like to thank Frank Teusink: part of this work has been done together. Further, I would like to thank the anonymous referees for their helpful comments. This work was supported in part by the Italian "Comitato naz.le Scienze e Tecnologia dell’Informazione" under the project "Programmazione Logica".

Appendix A

In this appendix, we provide the proof of Lemma 13. First, we need an additional technical Lemma.

Lemma A.1. Let $M$ and $N$ be modules on $\mathcal{L}_b$, such that $M \oplus N$ is defined. Then 
\[
[M^{n+1}] \circ [(M \oplus N)^n] \rightarrow M \circ [(M \oplus N)^n].
\]

Proof. We proceed by induction on the index $n$. For $n = 0$ the thesis holds trivially, because $[M^1] \circ [(M \oplus N)^0] \equiv M \circ [(M \oplus N)^0]$. Assume we proved the thesis for $n$. We want to prove that
\[
[M^{n+2}] \circ [(M \oplus N)^{n+1}] \rightarrow M \circ [(M \oplus N)^{n+1}]
\]
First, we need to prove the following identity:
\[
(M \circ [M^{n+1}]) \circ [(M \oplus N)^{n+1}]
\]
\[
\equiv M \circ ((([M^{n+1}] \circ [(M \oplus N)^{n+1}]) \oplus \[(M \oplus N)^{n+1}]_{\text{def}(N)})
\]
where we denote by $\[(M \oplus N)^{n+1}]_{\text{def}(N)}$ the restriction of $[(M \oplus N)^{n+1}]$ to those formulae that define the predicates of $\text{Def}(N)$. In order to prove this let us focus on the leftmost occurrence of the module $A$ in the above formula, and consider an atom $A$ in the body of a definition of $M$. If $\text{Pred}(A)$ is defined in $M$ then $A$ will be unfolded via $[M^{n+1}]$ and successively via $[(M \oplus N)^{n+1}]$. Otherwise, if $\text{Pred}(A)$ is not defined in $M$ then $A$ will be left unchanged by the application of the unfolding via $[M^{n+1}]$. It might successively be modified by the unfolding via $[(M \oplus N)^{n+1}]$. This is exactly what would happen if we unfolded $A$ via
\[
\(([M^{n+1}] \circ [(M \oplus N)^{n+1}]) \oplus \[(M \oplus N)^{n+1}]_{\text{def}(N)})
\]
And this is what we do (to $A$) on the RHS of (A.1). This proves (A.1).

Secondly, one should observe that
\[
\[(M \oplus N)^{n+1}]_{\text{def}(N)} \equiv [M \oplus N]_{\text{def}(N)} \circ [(M \oplus N)^n] \equiv (M \oplus N)^n
\]
(A.2)
We are now able to prove the thesis.

\[
[M^{n+2}] \circ [(M \oplus N)^{n+1}] \\
\equiv (M \circ [M^{n+1}]) \circ [(M \oplus N)^{n+1}] \quad \text{by (A.1)} \\
\equiv M \circ ( [(M^{n+1}) \circ ((M \oplus N)^{n+1})] \oplus [(M \oplus N)^{n+1}]_{Def(N)} ) \quad \text{by (A.2)} \\
\equiv M \circ ((M^{n+1}) \circ ((M \oplus N)^{n+1}) \oplus (N \circ ((M \oplus N)^{n})])
\]

Now, by Remark 12

\[
\equiv M \circ ((M^{n+1}) \circ ((M \oplus N)^{n}) \oplus (N \circ ((M \oplus N)^{n})))
\]

By the inductive step \( [M^{n+1}] \circ ((M \oplus N)^{n}) \mapsto M \circ ((M \oplus N)^{n}) \), so by Remark 12,

\[
\equiv M \circ ((M \circ ((M \oplus N)^{n})) \oplus (N \circ ((M \oplus N)^{n})))
\]

\[
\equiv M \circ ((M \circ N) \circ ((M \oplus N)^{n}))
\]

\[
\equiv M \circ ((M \oplus N)^{n+1}]
\]

Hence the thesis. \( \square \)

We can finally prove Lemma 13.

**Lemma 13.** Let \( M \) and \( N \) be modules on the base language \( \mathcal{L}_B \) such that \( M \oplus N \) is defined. Then \( [(M^n) \circ [N^n]] \mapsto [(M \oplus N)^n] \).

**Proof.** We proceed by induction on \( n \). For the base case, where \( n = 1 \), the thesis holds trivially, because \( [(M^1) \circ [N^1]] = [(M \oplus N)^1] \).

Now, assume the thesis holds for \( n \). Then

\[
[(M^{n+1}) \circ [N^{n+1}]]
\]

\[
\equiv [(M^{n+1}) \circ [N^{n+1}]) \circ ((M^{n+1}) \circ [N^{n+1}])]
\]

By Remark 12 it follows that

\[
\equiv [(M^{n+1}) \circ [N^{n+1}]) \circ (M^n) \circ [N^n]].
\]

By the inductive step and Remark 12,

\[
\equiv [(M^{n+1}) \circ [N^{n+1}]) \circ (M^n) \circ [N^n]]
\]

and, by Lemma A.1,

\[
\equiv [(M \circ (M \oplus N)^n) \circ (N \circ (M \oplus N)^n)]
\]

\[
\equiv [(M \circ N) \circ (M \oplus N)^n]
\]

\[
\equiv [(M \oplus N)^{n+1}]
\]

Hence the thesis holds for \( n + 1 \). \( \square \)
References


