# A Refinement of Import/Export Declarations in Modular Logic Programming and its Semantics

Isambo Karali and Constantin Halatsis

Department of Informatics, University of Athens

**Abstract.** Encapsulation constructs with import/export declarations is the structuring facility offered in most commercial Prolog systems. However, real-life applications have shown to require a finer information exchange between encapsulated pieces of code. In this paper, a refinement of import/export declarations for modules of logic programs is presented. This offers a stricter form of communication between the modules and a larger variety of visibility states of their predicates, the standard approaches being special cases of it. The semantics of this module system has been examined and model-theoretic, fixpoint and operational ones are given and have been proved to be equivalent. Instead of using other logics, all these semantics extend the ones of Horn clause logic using concepts commonly used in it. In addition, the module system has been naturally transformed to Horn clause logic exploiting the distinction of the predicates within a module according to the interface declarations of this module. A form of equivalence with the other semantics of the system is given. In addition, the employed transformation has provided us with a basis for a preprocessor based implementation of the module system.

## 1 Introduction

The declarative style of logic programming [14] via its implementation as Prolog [8, 22] facilitates the software development phase and has become a favorite platform for a large variety of advanced applications (artificial intelligence applications, expert systems, scheduling applications etc.). Thus, a variety of Prolog implementations incorporated also an ecapsulation mechanism. At the same time, much work is being carried out in providing logic programming with a structuring mechanism with formal foundations which fits elegantly in the underlying theory, since one of the main advantages of logic programming is its clear semantics. As stated in [7], two main lines of research exist towards this direction: program composition and linguistic extensions. The former has been inspired from the work in [20] where an algebra for logic programs is introduced. Logic programs are the elements of this algebra and a variety of operators performs program composition. Various researchers [15, 11, 5, 1] have worked in this area. The module support with import/export interfaces, which is also the case of commercial Prolog systems, is a special case of this approach and provides

a fine-grained program composition. On the other hand, linguistic extensions, first appearing in [16], extend the syntax of Horn clauses by allowing implications as goals. This approach offers a rich expressive power, but its application areas are still under investigation. In the area of linguistic extensions, the work of contextual logic programming [17] is an eminent one. The work in [16] uses intuitionistic logic to formulate the introduced ideas. Other logics are also used for a module concept in logic programming in a variety of works.

The module system presented and examined in this paper is the result of the effort to make a structuring tool for large applications which require encapsulation and complicated interconnections. It provides privacy, abstraction and name clashes avoidance. The presented system relies on the area of module support with import/export declarations and it can be said to extend the functionality in predicate visibility of [10]. The encapsulation affects predicates only. Although coming out from practical needs, our module system has been given a model-theoretic, a fixpoint as well as an operational semantics, all proved to be equivalent. The semantics has been concentrated on modelling the environment of a module with respect to other modules that form the surrounding system. Instead of relying on other logics, all these semantics extend the standard semantics of Horn clause programs [23] naturally and smoothly to capture the introduced extension. In the employed formalism, modules are a kind of first order theories which differ from classical ones in that the encapsulation and the interfaces qualify the predicates. Commonly used and widely understood concepts such as Herbrand interpretations and models as well as the immediate consequence operator have been found to be sufficient to form the means of the semantics of this module system. Nevertheless, a transformation of the module system to Horn clause logic has been also defined and a sort of equivalence with the other semantics of the system has been proved. A preprocessor implementation of the module system is based on this transformation [13]. The preprocessor extends the standard Prolog functionality so that to take the module system into account. Modular Prolog programs are mapped to flat Prolog code by the preprocessor which is coded in Prolog itself.

In this paper, firstly, the module system is briefly presented. Its model-theoretic, fixpoint and operational semantics follow. The transformation of the module system to Horn clause logic comes next. In all cases, the equivalence relation is stated. Proofs of all theorems can be found in [12]. Finally, related work is included and comments on the presented module system are made.

## 2 The Module System

In this section, firstly, the module system is briefly presented from the Prolog programmer's point of view. Secondly, the system's formal framework is introduced.

A module encapsulates a set of Prolog procedures that form its *body* and exchanges information with other modules via *interface declarations*. As *home module* of an interface declaration, we consider the module where this declaration

appears. A *module header* of the form *:-module(ModuleName),* where *Module Name* is a constant, introduces a module with name *ModuleName* and denotes that the interface declarations as well as the Prolog code that follow until the end-of-file mark or another module header form the module.

The interface declarations carry out the following. A module is allowed to declare via its interface all or some of its predicates *global* to the system, i.e. visible everywhere, "its predicates" referring to the ones defined in this module. In addition, it can declare them *exported to* specified modules or to all the modules of the system. Furthermore, a module may *import* predicates, at the extensional level, *from* specific modules. In this case, to achieve usability, the other module must export the predicates to the home module of the import declaration. Moreover, a module may *merge* procedure results *from* other modules, specific or not, with its own ones. Again an export declaration in the other module must export the procedure to the home module of the merge declaration. The order of the merged results is not ensured and depends on the order of the module loading. When a merge declaration within a module refers to a predicate, the module is not allowed to declare this predicate exported or global. No more than one module should declare the same predicate global. In case a predicate exists in an export declaration it is not allowed to appear in a global one in the home module of the export declaration. If more than one merge or import declarations within a module refer to the same predicate, the declaration that appears last is considered. The preceding ones are ignored. The interface declarations are processed at compile time and cannot change at run time.

No space outside the modules exists. The concept of *worlds* is effectively supported. That is to say, the top level loop executes within the environment of a module and all predicates visible to this module are visible and usable in the top level loop.

In order to determine which procedure definition is addressed when a predicate appears within a module, *visibility states* are introduced and define the effect of the interface declarations.

**Definition 1.** A predicate can fall into one and only one of the following *visibility states* within a module:

**merged** iff a merge declaration exists in this module and refers to this predicate
**local** iff the above does not hold and there is a definition for this predicate in this module
**imported** iff none of the above holds and there is an import declaration in this module that refers to this predicate
**possibly_global** iff none of the above holds

In the formalism of the module system presented in this paper, pure logic programming and definite program clauses are assumed. The terminology used in [14] is adopted. Modules are considered as a kind of first order theories, namely *module First Order Theories* (*m-FOTs*). Each module is a m-FOT. A m-FOT differs from a Horn clause theory in that:

1. It includes a constant identifying the m-FOT which corresponds to the name of the module.
2. Predicates in a m-FOT do not belong to a flat set. The set of predicate symbols is substituted by a 4-tuple $P = (M, L, I, GO)$. Each of the $M$, $L$, $I$, $GO$ correspond to the merged, local, imported and possibly_global predicates. $L$ is further refined to $L = (H, E, GI)$, i.e. hidden, exported and declared global predicates. A hidden predicate in a module is a local predicate which is neither exported nor declared global in this module. All $M$, $H$, $E$, $GI$, $I$ and $GO$ sets contain predicates distinct from each other. As expected, $M$, $E$ and $I$ combine the predicates with the modules that the interface declarations relate them to. To illustrate this, consider the case of the merged predicates within a module $m$. $M$ is a finite set of $(mms, mp)$ pairs and $(mpr)$ elements where $mms$ is a finite set of constants other than $m$ and $mp$, $mpr$ are predicates. An $(mms, mp)$ pair in $M$ denotes that $mp$ is a predicate merged from the modules in $mms$, though $(mpr)$ denotes that the predicate $mpr$ is merged from all the modules. As, in the following, a set of modules will be considered, the module name will be used to identify the predicate sets among the various modules, i.e. $E(m)$ represents the exported predicates of the module $m$.
3. As far as the program clauses of the m-FOTs are concerned, only predicates from $M$ or $L$ should appear in their heads.

A $(V, F)$ pair where $V$ is a finite or denumerably infinite set of variables and $F$ a finite set of function symbols is assumed to be global to all the m-FOTs. The module system affects predicates only. The m-FOTs are, then, reduced to a triple $(m, P, Progr)$ where $m$ is the constant which identifies the m-FOT and $P$ is as previously described. $Progr$ is a finite set of program clauses which forms the module's program and corresponds to the module's body.

Consider

$$\begin{aligned}
Merged(m) &= \{p|(p) \in M(m)\} \cup \{p|(ms, p) \in M(m)\} \\
Local(m) &= H(m) \cup \{p|(p) \in E(m)\} \cup \{p|(m', p) \in E(m)\} \cup GI(m) \\
Imported(m) &= \{p|(m', p) \in I(m)\} \\
PGlobal(m) &= GO(m) \\
Using(m) &= Merged(m) \cup Local(m) \cup Imported(m) \cup PGlobal(m)
\end{aligned}$$

Moreover,

$$\begin{aligned}
M(m, m') &= \{p|(ms, p) \in M(m) \text{ and } m' \in ms\} \cup \{p|(p) \in M(m)\} \\
I(m, m') &= \{p|(m', p) \in I(m)\} \\
E(m, m') &= \{p|(m', p) \in E(m)\} \cup \{p|(p) \in E(m)\}
\end{aligned}$$

## 3 Model-Theoretic Semantics

In this section, a model-theoretic semantics of the module system is given. This is based on Herbrand interpretations and provides a minimal model as model intersections.

The work in this paper is concentrated on the semantics of a module *surrounded by* a set of modules. The reason is that every formula is associated with a module, as there is no space outside the modules. Thus, in this section, we try to model what holds with respect to this module which, certainly, depends on what holds to other modules.

**Definition 2.** Consider a m-FOT $(m, P, Progr)$. Then, a set

$$SW(m) = \{(m_i, P_i, Progr_i) |\ 1 \leq i \leq n, n \in \mathrm{N},\ (m_i, P_i, Progr_i)\ a\ m\text{-}FOT,\ m_i \neq m, GI(m_i) \neq GI(m)\ and\ \forall j\ such\ that\ 1 \leq j \leq n, j \neq i : m_i \neq m_j, GI(m_i) \neq GI(m_j)\}$$

where N is the set of natural numbers, is called a *surrounding world* of $m$ and $m$ *is surrounded by* $SW(m)$ *(sym:* $m$ *srd* $SW(m)$*)*

From now on, the triple representing a m-FOT and the constant identifying it will be used interchangeably.

The concepts of Herbrand Universe ($U$) and Herbrand Base of the language for a m-FOT, as well as the one of interpretations are left unchanged. Herbrand interpretations are considered. Nevertheless, a Herbrand Base is referred as *m-Herbrand Base* (*m-B*), because predicates are module dependent and, for a m-FOT $m$, it is built over the predicates in $Using(m)$. What changes, however, is the concept of truth value of an atomic formula. As all formulae and programs in the module system are addressed with respect to a module, they will be referred to as *m-formulae* and *m-programs*, respectively. Moreover, consider that for a m-atomic formula $A = p(t_1, \ldots, t_n)$, it is $pred(A) = p$. Then, if a m-atomic formula of a module involves a local predicate, it is the module's responsibility to assign the truth value to it. This is partially the case for merged predicates. Otherwise, it is a matter of interface declarations and of other module information. For instance, if the predicate of a m-atomic formula $A$ of a module $m$ is possibly_global, i.e. $pred(A) \in GO(m)$, then this formula will be true if there exists another module $m'$ in the surrounding world which declares $pred(A)$ global, i.e. $pred(A) \in GI(m')$, and $A$ is true within $m'$. Informally speaking, truthity of a ground m-atomic formula of a module is again identical to membership to an interpretation. However, which module's interpretation is considered depends on the visibility state of the formula's predicate within this module.

**Definition 3.** Consider a m-FOT $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in \mathrm{N}$, an interpretation $I_0$ of $m_0$ and $Is = \{I_i | 1 \leq i \leq n, I_i : interpretation\ of\ m_i \in SW(m_0)\}$. Then, a ground m-atomic formula $A$ of $m_0$ is *true in interpretation* $I_0$ *surrounded by* $Is$ *of* $SW(m_0)$ *(sym: true in* $I_0$ *srd* $Is$*)* iff

1. $pred(A) \in Merged(m_0)$ and
   either $A \in I_0$ or
   $\exists m_i \in SW(m_0) :\ pred(A) \in M(m_0, m_i) \cap E(m_i, m_0)$ and $A \in I_i$ where $I_i \in Is$ is the interpretation of $m_i$.
2. $pred(A) \in Local(m_0)$ and $A \in I_0$.

3. $pred(A) \in Imported(m_0)$ and
   $\exists m_i \in SW(m_0) : pred(A) \in I(m_0, m_i) \cap E(m_i, m_0)$ and $A \in I_i$ where $I_i \in Is$ is the interpretation of $m_i$.
4. $pred(A) \in PGlobal(m_0)$ and $\exists m_i \in SW(m_0) : pred(A) \in GI(m_i)$ and $A \in I_i$ where $I_i \in Is$ is the interpretation of $m_i$.

Assuming this definition, the definition of the truth value of m-atomic formulae and m-clauses is kept the same as it is for first order atomic formulae and Horn clauses. Considering the null formula $\Delta$, it can be said that it is true everywhere.

As mentioned before, the approach is concentrated on examining what holds in a module with respect to what holds in some other modules. Thus, it is natural and useful to restrict these other modules to the information relevant to this central module only. In other words, considering a $m_0$ *srd* $SW(m_0)$, the surrounding world $SW(m_0)$ is restricted to $SW'(m_0)$ by restricting each module $m_i$ in $SW(m_0)$. Each module in the surrounding world is restricted to these interface declarations and code only that are used either directly by $m_0$ or indirectly by some other module restriction. If, after the reduction, a module is empty, it is not included in $SW'(m_0)$ at all. In terms of the formalism previously presented, for each $(m_i, (M, (H, E, GI), I, GO), Progr) \in SW(m_0)$, if its restriction is not empty, it is denoted as $(m_i, (M', (H', E', GI'), I', GO'), Progr')$ where $E'$, $GI'$ and $Progr'$ contain only the "useful" information for $m_0$ and the other module restrictions. $H'$, $M'$, $I'$ and $GO'$ contain predicates that appear in $Progr'$. The construction of $SW'(m_0)$ can be carried out in a systematic way.

The restriction of the surrounding world is followed in the concept of models. A model for a module $m_0$ surrounded by $SW(m_0)$ should model $m_0$ taking into account all requested information from $SW(m_0)$. In addition, $SW'(m_0)$ should be modelled as well. A model in our framework consists of an interpretation of the central module which is considered with respect to a set of interpretations, one for each module of the surrounding world.

Considering a central module surrounded by a set of modules contrasts to other approaches [17, 6, 2] where a set of units is considered as a world and effort is made to model such a world. In these approaches, all units are treated uniformly thus the formalism becomes more compact. However, our intention is to model the environment of one module. Thus, if we tried to model this central module and its surrounding world uniformly as a set of modules, we wouldn't be complete. The reason is that some candidate models would have been rejected because they don't model the whole information, although some of it is irrelevant to the central module. Thus, considering the concept of the surrounding world and restricting it have been found necessary.

**Definition 4.** Consider a m-FOT $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in \mathrm{N}$, an interpretation $M_0$ of $m_0$ and $Ms = \{M_i | 1 \leq i \leq n, M_i : interpretation\ of\ m_i \in SW(m_0)\}$. Then, $M_0$ *srd* $Ms$ is a *model for* $m_0$ *srd* $SW(m_0)$ *(sym, also:* $M_0$ *model for* $m_0$ *srd Ms of* $SW(m_0)$*)* iff all the following hold:

1. Every m-clause of the m-program of $m_0$ is true in $M_0$ *srd* $Ms$.

2. $\forall m_i \in SW'(m_0)$ and $\forall$ m-atomic formula $A$ of $m_i$ such that $pred(A) \in Accord\_set(m_0, m_i)$, the following holds: if $A \in M_i \in Ms$ and $M_i$ is the interpretation of $m_i$ then $A \in M_0$ where:

$$\begin{aligned} Accord\_set(m_0, m_i) = &(M(m_0, m_i) \cap E'(m_i, m_0)) \cup \\ &(I(m_0, m_i) \cap E'(m_i, m_0)) \cup \\ &(GO(m_0) \cap GI'(m_i)) \end{aligned}$$

and

$$E'(m_i, m_0) = \{p | (m_0, p) \in E'(m_i)\} \cup \{p | (p) \in E'(m_i)\}$$

Informally speaking, the accordance set of $m_0$ to $m_i$ represents the predicates that $m_0$ asks from $m_i$ and $m_i$ offers to $m_0$.

3. Items 1 and 2 hold $\forall m_i \in SW'(m_0)$, considering the $m_i$ *srd* $(SW'(m_0) - \{m_i\}) \cup \{m_0\}$ and the $M_i$ *srd* $(Ms - \{M_i\}) \cup \{M_0\}$ where $M_i \in Ms$ is the interpretation of $m_i$.

**Proposition 5.** *Let $M_0$ srd $Ms$ be a model for $m_0$ srd $SW(m_0)$ and $A$ a ground m-atomic formula of $m_0$. Then, $A$ true in $M_0$ srd $Ms \Rightarrow A \in M_0$.*

**Definition 6.** Let $m_0$ *srd* $SW(m_0)$ and $F$ either a m-atomic formula of $m_0$ or a m-clause of $m_0$. Then, $F$ is a *logical consequence of* $m_0$ *srd* $SW(m_0)$ iff $\forall M_0$ *srd* $Ms$ model for $m_0$ *srd* $SW(m_0)$, $F$ is true in $M_0$ *srd* $Ms$.

**Definition 7.** Let $M_0$ *srd* $Ms$, $M_0'$ *srd* $Ms'$ be models for $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in \mathrm{N}$. Then $M_0$ *srd* $Ms \leq M_0'$ *srd* $Ms'$ iff $M_0 \subseteq M_0'$ and $\forall i$ such that $1 \leq i \leq n$: $M_i \subseteq M_i'$, where $M_i \in Ms, M_i' \in Ms'$ are the interpretations of $m_i \in SW(m_0)$. Thus, a partial order is defined on the set of models of a $m_0$ *srd* $SW(m_0)$.

**Proposition 8.** *The m-Herbrand base is a model for a m-FOT surrounded by the m-Herbrand bases of the surrounding world. Thus, the set of models for a m-FOT surrounded by other m-FOTs is nonempty.*

**Theorem 9.** *Let $M1_0$ srd $M1s$, $M2_0$ srd $M2s$ be models for $m_0$ srd $SW(m_0) = \{m_1, \ldots, m_n\}, n \in N$. Then $M_0 = M1_0 \cap M2_0$ is a model for $m_0$ srd $Ms$ of $SW(m_0)$, where $Ms = \{M1_i \cap M2_i | 1 \leq i \leq n, M1_i \in M1s, M2_i \in M2s$ and $M1_i, M2_i$ the interpretations of $m_i \in SW(m_0)\}$.*

**Proposition 10.** *Let $m_0$ srd $SW(m_0) = \{m_1, \ldots, m_n\}, n \in N$. Let $Models = \{M_0$ srd $Ms | M_0$ srd $Ms$ is a model for $m_0$ srd $SW(m_0)\}$. Then, $M_{MIN}$ srd $M_{MIN}s$ where*

$$\begin{aligned} M_{MIN} &= \cap\{M_0 | M_0 \text{ srd } Ms \in Models\} \text{ and} \\ M_{MIN}s &= \{M_i | 1 \leq i \leq n, \\ &\quad M_i = \cap\{Ms_i | Ms_i \in Ms,\ M_0 \text{ srd } Ms \in Models \text{ and} \\ &\quad Ms_i \text{ is the interpretation of } m_i \in SW(m_0)\}\} \end{aligned}$$

*is the minimal model for $m_0$ srd $SW(m_0)$ (with respect to the defined partial order).*

## 4 Fixpoint Semantics

In this section, a fixpoint semantics of the module system is given in terms of a continuous transformation associated with a module surrounded by a surrounding world. The transformation's least fixpoint involves the minimal model of the system into consideration. The employed transformation can be considered as an extension of the immediate consequence operator introduced in [23].

**Definition 11.** Let $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in \mathrm{N}$.
Let $m\text{-}B(m_i)$ denote the m-Herbrand base of the m-FOT $m_i$, $0 \leq i \leq n$.
Let $Lat = \{(I_0, Is) | I_0 \subseteq m\text{-}B(m_0), Is = \{I_i | 1 \leq i \leq n, I_i \subseteq m\text{-}B(m_i)\}\}$.
Then, consider a transformation $T : Lat \rightarrow Lat$ such that $T : (I_0, Is) \mapsto (I_0{}', Is')$ where:

$$I_0{}' = \{A | A \in m\text{-}B(m_0), \exists C \textit{ ground instance of a clause in } Progr(m_0) : C \equiv A\text{:-}A_1, \ldots, A_k \textit{ and } A_1, \ldots, A_k \textit{ true in } I_0 \textit{ srd } Is, k \geq 0\} \cup \{A | A \in I_j', 1 \leq j \leq n, pred(A) \in Accord\_set(m_0, m_j)\}$$

$$I_i' = \{A | A \in m\text{-}B(m_i), \exists C \textit{ ground instance of a clause in } Progr'(m_i) : C \equiv A\text{:-}A_1, \ldots, A_k \textit{ and } A_1, \ldots, A_k \textit{ true in } I_i \textit{ srd } (Is - \{I_i\}) \cup \{I_0\}, k \geq 0\} \cup \{A | A \in I_j', \ 0 \leq j \leq n, j \neq i, pred(A) \in Accord\_set(m_i, m_j), m_i \in SW'(m_0)\}$$

*or*

$$I_i' = I_i, \textit{ if } m_i \notin SW'(m_0)$$

and $Is' = \{I_i' | 1 \leq i \leq n\}$

*Lat* is a complete lattice with partial order $\leq$ such that $(I_0, Is) \leq (I_0{}', Is')$ iff $I_0 \subseteq I_0{}'$ and $\forall i$ such that $1 \leq i \leq n$: $I_i \subseteq I_i'$, where $I_i \in Is, I_i' \in Is'$ are interpretations of $m_i \in SW(m_0)$. In other words, the order defined for models is extended to interpretations. If $X \subseteq Lat$, then

$$lub(X) = (\cup_{(I_0,Is) \in X} I_0, \{Un_i | 1 \leq i \leq n \textit{ and } Un_i = \cup_{(I_0,Is) \in X \textit{ and } I_i \in Is} I_i\})$$

$$glb(X) = (\cap_{(I_0,Is) \in X} I_0, \{Int_i | 1 \leq i \leq n \textit{ and } Int_i = \cap_{(I_0,Is) \in X \textit{ and } I_i \in Is} I_i\})$$

**Theorem 12.** *Let* $m_0$ *srd* $SW(m_0)$ *and* $T$ *as previously defined.* $T$ *is continuous.*

**Theorem 13.** *Consider* $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in N$.
*Then,* $I_0$ *srd* $Is$ *is a model for* $m_0$ *srd* $SW(m_0) \Rightarrow T((I_0, Is)) \leq (I_0, Is)$

**Corollary 14.** *Let* $m_0$ *srd* $SW(m_0)$ *and* $M_{MIN}$ *srd* $M_{MIN}s$ *its minimal model. Then,* $lfp(T) \leq (M_{MIN}, M_{MIN}s)$.

**Theorem 15.** *Consider* $m_0$ *srd* $SW(m_0) = \{m_1, \ldots, m_n\}, n \in N$.
*Then,* $T((I_0, Is)) = (I_0, Is) \Rightarrow I_0$ *srd* $Is$ *is a model for* $m_0$ *srd* $SW(m_0)$.

**Corollary 16.** *Let* $m_0$ *srd* $SW(m_0)$ *and* $M_{MIN}$ *srd* $M_{MIN}s$ *its minimal model. Then,* $(M_{MIN}, M_{MIN}s) \leq lfp(T)$.

**Corollary 17 Equivalence.** *Let* $m_0$ *srd* $SW(m_0)$ *and* $M_{MIN}$ *srd* $M_{MIN}s$ *its minimal model.*
*Then,* $(M_{MIN}, M_{MIN}s) = lfp(T) = T \uparrow \omega$.

## 5 Operational Semantics

In this section the operational behaviour of the module system is presented. More precisely, the concept of derivability of a m-formula from a module surrounded by a surrounding world is expressed in terms of a set of inference rules, adopting a frequently used framework. The notation $m_0$ *srd* $SW(m_0) \vdash F[\theta]$ is used to denote that there is a derivation of $F$ from $m_0$ *srd* $SW(m_0)$ with substitution $\theta$, where $F$ is a m-formula of $m_0$ such as the null formula, a m-atomic formula or conjunction of these. The inference rules are of the form:

$$\frac{\textit{Assumption}}{\textit{Conclusion}}$$

Such a rule states that *Conclusion* holds whenever *Assumption* holds. In the following, $\varepsilon$ denotes the identity substitution and $\theta$, $\sigma$ are substitutions. Moreover, $A$ is a m-atomic formula of $m_0$ and $Gs$ is a conjunction of those. The result of applying $\theta$ to $A$ is written $A\theta$.

**Null Formula**

$$\frac{}{m_0 \textit{ srd } SW(m_0) \vdash \Delta[\varepsilon]}$$

**Conjunction**

$$\frac{m_0 \textit{ srd } SW(m_0) \vdash A[\theta] \wedge m_0 \textit{ srd } SW(m_0) \vdash Gs\theta[\sigma]}{m_0 \textit{ srd } SW(m_0) \vdash (A, Gs)[\theta\sigma]}$$

**m-Atomic Formula(I)**

$$\frac{\exists A_0 \text{:-} B_0 \in Progr(m_0) \wedge \theta = mgu(A, A_0) \wedge m_0 \textit{ srd } SW(m_0) \vdash B_0\theta[\sigma]}{m_0 \textit{ srd } SW(m_0) \vdash A[\theta\sigma]}$$

**m-Atomic Formula(II)**

$$\frac{\begin{array}{l}\exists m_i \in SW'(m_0) : pred(A) \in Accord\_set(m_0, m_i) \wedge \\ m_i \textit{ srd } (SW'(m_0) - \{m_i\}) \cup \{m_0\} \vdash A[\theta]\end{array}}{m_0 \textit{ srd } SW(m_0) \vdash A[\theta]}$$

Actually, it is only the last rule that corresponds to the extension introduced by the module system and expresses the change in the proof environment that is carried out when an "external" predicate appears in a m-atomic formula.

With respect to the model-theoretic semantics previously presented, the following relations hold.

**Theorem 18.** *Consider $m_0$ srd $SW(m_0)$ and $A$ a ground m-formula of $m_0$. Then, $m_0$ srd $SW(m_0) \vdash A \Rightarrow A$ is true in $M_0$ srd $Ms$, $\forall M_0$ srd $Ms$ model for $m_0$ srd $SW(m_0)$.*

**Theorem 19.** *Consider $m_0$ srd $SW(m_0)$ and $A$ a ground m-formula of $m_0$. Then, $A$ is true in $M_{MIN}$ srd $M_{MINS} \Rightarrow m_0$ srd $SW(m_0) \vdash A$, where $M_{MIN}$ srd $M_{MINS}$ is the minimal model of $m_0$ srd $SW(m_0)$.*

**Corollary 20 Equivalence.** *Consider $m_0$ srd $SW(m_0)$ and $A$ a ground m-atomic formula of $m_0$. Then, $A$ is a logical consequence of $m_0$ srd $SW(m_0)$ $\Leftrightarrow m_0$ srd $SW(m_0) \vdash A$.*

## 6 Transformation to Horn Clause Logic

The previously presented module system can be easily transformed to Horn clause logic. Each module may be mapped onto a flat Horn clause program. Thus, a module surrounded by a surrounding world is mapped onto a Horn clause theory that is the union of the images of the modules of the system. Each m-clause in a module's body is mapped onto a Horn clause. Every m-atomic formula of the m-clause is substituted by an atomic formula according to the visibility state of the predicate of the m-atomic formula within this module. The interface declarations of a module are mapped onto Horn clauses that provide a link between the inter-module atomic formulae.

The mapping of the m-atomic formulae is defined as follows.

**Definition 21.**

$$map: \; M\text{-}FOTs \times M\text{-}atomic\text{-}formulae \rightarrow atomic\text{-}formulae$$

where $M\text{-}FOTs$ is the set of m-FOTs, $M\text{-}atomic\text{-}formulae$ is the set of m-atomic formulae, $atomic\text{-}formulae$ is the set of first order atomic formulae and $map$ is defined in the following way:

$$\begin{aligned}
map(mn, Atom) &= m(mn, Atom), && if\ pred(Atom) \in Merged(mn) \\
map(mn, Atom) &= l(mn, Atom), && if\ pred(Atom) \in Local(mn) \\
map(mn, Atom) &= e(mn, otherm, Atom), && if\ pred(Atom) \in Imported(mn) \\
map(mn, Atom) &= g(Atom), && if\ pred(Atom) \in PGlobal(mn)
\end{aligned}$$

where $mn$ is a m-FOT, $Atom$ a m-atomic formula of $mn$ and $otherm$ is a m-FOT such that $pred(Atom) \in I(mn, otherm)$.

The mapping is extended to m-clauses and m-programs by considering the image of each m-atomic formula in the m-clause and the image of each m-clause in the m-program.

As previously mentioned, the interface declarations provide a set of clauses which derive as follows.

**Definition 22.** Consider a m-FOT $(mn, P, Progr)$ and $(V, F)$ a pair of variables and function symbols global to the whole system. *Interface_clauses(mn)* is a set of clauses such that:

1. $\forall(em, prd) \in E(mn), \exists$ an interface clause which is:

$$e(em, mn, Atom) \ :- \ l(mn, Atom)$$

where $Atom = prd(t_1, \ldots, t_n)$, $n$ is the arity of $prd$ and $t_1, \ldots, t_n \in V$.
2. $\forall(prd) \in E(mn), \exists$ an interface clause which is:

$$e(em, mn, Atom) \ :- \ l(mn, Atom)$$

where $Atom$ as above and $em \in V$.
3. $\forall(mms, prd) \in M(mn)$ and $\forall mm \in mms, \ \exists$ an interface clause which is:

$$m(mn, Atom) \ :- \ e(mn, mm, Atom)$$

where $Atom$ as above.
4. $\forall(prd) \in M(mn), \exists$ an interface clause which is:

$$m(mn, Atom) \ :- \ e(mn, mm, Atom)$$

where $Atom$ as above and $mm \in V$.
5. $\forall prd \in GI(mn), \exists$ an interface clause which is:

$$g(Atom) \ :- \ l(mn, Atom)$$

where $Atom$ as above.

Then, considering the above definitions, the mapping is also extended to m-FOTs in the following way.

**Definition 23.** Assuming $(V, F)$ a pair of variables and function symbols global to the system, each m-FOT $(mn, P, Progr)$ is mapped onto a Horn clause theory with function symbols $F' = F \cup Using(mn)$, variables $V$ and program $Progr' = map(mn, Progr) \cup Interface\_clauses(mn)$. The set of predicate symbols is $P' = \{m, l, e, g\}$ where $m$,$l$ are 2-ary predicate symbols, $e$ a 3-ary predicate symbol and $g$ is a unary predicate symbol.

To comment on the mapping, it can be said that it exploits and reflects the distinction of the predicates among the various modules as well as within a module according to the interface declarations of this module. Moreover, it achieves parameterization with respect to the modules of the surrounding world by transforming a module body independently from any other module's transformation. In addition, it treats interfaces separately from any code. Techniques employed in other work, where structuring extensions have been introduced to logic programming and have been mapped onto flat code [1, 17, 18, 19, 21, 9], are not sufficient or relevant to our case. A straightforward predicate renaming can be deduced from the substitution of the m-atomic formulae performed by the mapping.

It is interesting to see the relation of the mapping with the other semantics of the module system. More precisely, the relation with the model-thoretic semantics has been investigated.

Consider $m_0$ *srd* $SW(m_0)$ where $m_0$ is $(m_0, (M, (H, E, GI), I, GO), Progr)$. Then, reduce its $E$ and $GI$ sets to their elements which contain predicates used by $SW'(m_0)$ and add the remaining elements to $H$. At the same time, for both the modified $m_0$ as well as all modules in $SW'(m_0)$, eliminate the concept of "all" modules from the interfaces by explicitly enumerating the modules in $SW'(m_0) \cup \{m_0\}$ (minus the home module of the interface). Then, consider $P1_{RESTR}$ to be the program of $map(m_0) \cup (\cup_{m \in SW'(m_0)} map(m))$ where $m_0$ *srd* $SW'(m_0)$ is modified as previously described. From $P1_{RESTR}$ derive $P_{RESTR}$ by substituting each pair of clauses:

$$m(InM, Atom) \ :- \ e(InM, FromM, Atom)$$

and

$$e(InM, FromM, Atom) \ :- \ l(FromM, Atom)$$

by the clause

$$m(InM, Atom) \ :- \ l(FromM, Atom)$$

where $InM$, $FromM$ are module names and $Atom$ an atomic formula. If a merge interface clause cannot be paired, it is not included in $P_{RESTR}$.

Considering $m_0$ *srd* $SW(m_0)$ and $P_{RESTR}$ derived as previously described, the following theorems have been proved.

**Theorem 24.** *Let* $m_0$ *srd* $SW(m_0)$ *and Atom a m-atomic formula of* $m_0$. *Then, Atom logical consequence of* $m_0$ *srd* $SW(m_0) \Leftrightarrow \forall(map(m_0, Atom))$ *logical consequence of* $P_{RESTR}$.

**Theorem 25.** *Let* $P_{RESTR}$ *as previously defined and* $MOD$ *a Herbrand model for* $P_{RESTR}$. *Consider*

$$MOD_{RESTR}(m_i) = \{Atom | Atom \in MOD, \ \exists Atom' \text{ a m-atomic formula of } m_i \text{ such that } map(m_i, Atom') = Atom\},$$

$\forall m_i \in SW'(m_0) \cup \{m_0\}$.
*Consider also*

$$MODm(m_i) = \{Atom' | Atom' \text{ is a m-atomic formula of } m_i \text{ such that } map(m_i, Atom') \in MOD_{RESTR}(m_i)\}, \ \forall m_i \in SW'(m_0) \cup \{m_0\}$$
$$MODm(m_i) = \emptyset, \ \forall m_i \in SW(m_0) - SW'(m_0)$$

*Thus,* $map(m_i, MODm(m_i)) = MOD_{RESTR}(m_i), \forall m_i \in SW'(m_0) \cup \{m_0\}$.
*Then,* $MODm(m_0)$ *is a model for* $m_0$ *srd* $MODms$ *of* $SW(m_0)$ *where* $MODms = \{MODm(m_i) | m_i \in SW(m_0)\}$.

**Theorem 26 Minimal Model Characterization in Terms of the Mapping.**

*Consider* $M_{MIN}$, *the minimal model of* $P_{RESTR}$. *Then,* $M_{MIN}m(m_0)$ *srd* $M_{MIN}ms$ *is the minimal model for* $m_0$ *srd* $SW(m_0)$.

## 7 Related Work and Discussion

There is also other work related to import/export program composition. The work of [20] should be mentioned where an algebra for building logic programs out of pieces is introduced. The elementary terms of the algebra are breeze blocks and building bricks. The building bricks are logic programs. The breeze blocks (include/exclude/rename) correspond to import/export lists of conventional module systems. Breeze blocks are considered as functions on predicate symbols. Building bricks are given a meaning as a monotone map from interpretations to interpretations. In [1], the introduced union, intersection and encapsulation operations are used to define two operators as powerful as import/export relations at the extensional level. Logic modules with import/export interfaces are studied in [11] within the framework of the abstract semantics they introduce. A logic module is a quadruple $(P, Im, Ex, Int)$. $Im$, $Ex$ and $Int$ are disjoint sets of the imported, exported and internal predicates, respectively. $P$ is a logic program with no clause having head with a predicate in $Im$. However, in their approach the intensional view is adopted, i.e. program clauses are imported/exported.

In [4] composition of logic programs is modelled by the composition of the admissible Herbrand models of the programs. Admissibility of a model is considered under an admissible set of hypotheses where each of them occurs in the body of a clause of the program. A hypothesis is an element of the Herbrand base of the program. Then, a Herbrand model is the admissible Herbrand model under these hypotheses if and only if it is the minimal model of the union of the program and these hypotheses. Admissible models are claimed to model even logic modules with import/export declarations and modules with import declarations have been studied in [3]. The concept of the surrounding world of our approach can be parameterized in terms of the concept of the hypotheses. Nevertheless, the conditions for admissibility of hypotheses must change to meet the enhanced requirements of the visibility states. At the same time, the admissible model's definition can be compared with the model concept presented in Sect. 3. However, only one admissible model is considered under some hypotheses, though more than one models are considered for a m-FOT surrounded by a surrounding world. Actually, the approach presented in [4] seems very interesting and it is in our near future work to try to abstract (parameterize) the semantics of our module system either by adopting and adjusting this approach or by developing a new one inspired from that work.

To comment on our module system, the following can be said. It can support complicated interconnections among a set of modules which is something that is needed in various application areas. In addition, higher encapsulation structures can be built on top of the existing module system according to the requirements of the specific application. Moreover, the module system can offer the encapsulation required in various advanced application areas, such as multi-agent systems, as well as in sophisticated platforms, such as task oriented languages. Another advantage is that information may be distributed since the module system may collect it to form a single block. Furthermore, incremental

module loading and unloading makes the system useful for the debugging phase. The system, although inspired from practical needs, has formal semantics which smoothly and naturally extends the ones of Horn clause logic. More precisely, standard model-theory [23] based on Herbrand interpretations and models as well as the immediate consequence operator have been suitably adjusted. What is more is that, no migration to other logics is required. In addition, the operational semantics extends the SLD-resolution by a simple rule that performs environment change, when required. A single module with no interfaces is a special case of the approach ($SW(m_0) = \emptyset$). Moreover, the approach taken is such that if a larger variety of interface declarations is needed, it can be extended by increasing the number of visibility states. The semantics can be easily adjusted to model the new system. Standard import/export declarations can be considered a special case of the approach. Visibility states are the pivot in the transformation of modular programs into flat code, implemented as a preprocessor. The preprocessor is written in Prolog, thus an integrated system can be derived. A form of equivalence with the other semantics of the system has been also provided for the transformation. The semantics given to the system can also model the world of the top level loop execution when various modules are loaded to the system. This is formalized as a central module surrounded by a surrounding world. The central module provides the environment of the top level loop execution and the surrounding world corresponds to the set of the loaded modules.

## 8 Conclusions

In this paper, a module system for logic programming was presented. This module system provides a finer program composition than import/export declarations and has been found useful to be a structuring tool in various application areas. This work showed that such a module system, although coming out from practice, has clear model-theoretic, fixpoint and operational semantics extending the ones of Horn clause logic using simple, well-known and widely understood concepts. In all the approaches we consider a central module surrounded by a surrounding world of other modules. In addition, modular logic programs are mapped to flat ones and this mapping provides a basis for a preprocessor implementation of the module system which meets the requirement of independent transformation. Equivalence relations have been proved for all the approaches of the semantics of the module system.

## References

1. A. Brogi. *Program Composition in Computational Logic.* PhD thesis, Università di Pisa, 1993.
2. A. Brogi, E. Lamma, and P. Mello. A general framework for structuring logic programs. Technical Report 4/1, CNR Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, 1990.

3. A. Brogi, E. Lamma, and P. Mello. Composing open logic programs. *Journal of Logic and Computation*, 3(91-11:4):1–25, 1992.
4. A. Brogi, E. Lamma, and P. Mello. Compositional model-theoretic semantics for logic programs. *New Generation Computing*, 11:1–21, 1992.
5. A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Composition operators for logic theories. In J. W. Lloyd, editor, *Proceedings of the Computational Logic Symposium*, pages 117–134, November 1990.
6. M. Bugliesi. A declarative view of inheritance in logic programming. In *Joint International Conference and Symposium on Logic Programming*, pages 112–127, 1992.
7. M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19,20:443–502, 1994.
8. W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, second edition, 1984.
9. R. Dietrich. A preprocessor based module system for Prolog. In *TAPSOFT'89 International Joint Conference on Theory and Practice in Software Development*, volume 2, pages 126–139, 1989.
10. *ECL$^i$PS$^e$: User Manual*, March 1993.
11. H. Gaifman and E. Shapiro. Fully abstract compositional semantics for logic programs. In *6th Conference on Principles of Programming Languages*, pages 134–142, 1989.
12. I. Karali and C. Halatsis. The semantics of a module support for logic programming. Technical report, Department of Informatics, University of Athens, 1993.
13. I. Karali, E. Pelecanos, and C. Halatsis. A versatile module system for Prolog mapped to flat Prolog. In *ACM Symposium on Applied Computing*, pages 578–585, 1993.
14. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
15. P. Mancarella and D. Pedreschi. An algebra for logic programs. In *5th International Conference in Logic Programming*, pages 1006–1023, 1988.
16. D. Miller. A theory of modules for logic programming. In *Proceedings of the 1986 Symposium on Logic Programming*, pages 106–114, 1986.
17. L. Monteiro and A. Porto. Contextual logic programming. In *6th International Conference in Logic Programming*, pages 284–299, 1989.
18. L. Monteiro and A. Porto. A transformational view of inheritance in logic programming. In *7th International Conference in Logic Programming*, pages 481–494, 1990.
19. Y. Moscowitz and E. Shapiro. Lexical logic programs. In *8th International Symposium in Logic Programming*, pages 349–363, 1991.
20. R. O'Keefe. Towards an algebra for constructing logic programs. In *IEEE Symposium on Logic Programming*, pages 152–160, 1985.
21. D. T. Sannella and L. A. Wallen. A calculus for the construction of modular Prolog programs. *Journal of Logic Programming*, 12(1):147–177, January 1992.
22. L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1986.
23. M. Van Emden and A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23:733–742, 1976.