Semantics for Modules in Functional-Logic Programming *

Juan M. Molina-Bravo Ernesto Pimentel
Dept. Lenguajes y Ciencias de la Computación. University of Málaga.
Campus de Teatinos. 29071 Málaga. Spain
e-mail: {jmmb,ernesto}@lcc.uma.es

Abstract

In this paper we present our work about modularity in declarative programming based on Constructor-Based Conditional Rewriting Logic. This logic has been studied by González Moreno and others and proposed as a general framework for integrating first-order functional and logic programming, giving an algebraic semantics for programs. We have extended this formalism with constructs for modularization, that allows to express hiding, export/import, genericity and inheritance; and several semantics for modules based on an immediate consequence operator $\mathcal{T}_\mathcal{P}$ defined for each program module $\mathcal{P}$ in the logic programming style.
Keywords: Functional-Logic Programming, Modules, Semantics, Rewriting Logic.

1 Introduction

Constructor-Based Conditional Rewriting Logic (CRWL), is a general approach to declarative programming combining functional and logic paradigms by means of the notion of non-deterministic lazy function from which both relations and deterministic lazy functions are particular cases. This approach, presented in [5], retains the advantages of deterministic functions while adds the possibility of modeling non-deterministic functions by means of non-confluent constructor-based term rewriting systems, where a given term may be rewritten to constructor terms (possibly with variables) in more than one way. A fundamental notion is that of *joinability*: two terms a,b are joinable iff they can be rewritten to a common —but not necessarily unique— constructor term. In [5], all these notions are introduced together with a model-theoretical semantics for CRWL-programs, two equivalent proof calculi, and a sound and complete lazy narrowing calculus.

Modularity in logic programming has been the objective of different proposals, see [3] for a survey about the subject. In functional programming and algebraic specification, the modularity issues have been generally addressed from a category-theoretic point of view ([4, 6]) giving functorial semantics to modules.

We have adopted a notion of module —together with a set of constructs— close to that proposed in [1], but dealing with the inherent characteristics of CRWL-programming. We consider [10] modules as open programs that can be combined by means of a set of operations that allow us to express some typical modular constructions. For each module $\mathcal{P}$, we introduce an immediate consequence operator $\mathcal{T}_\mathcal{P}$ in the logic programming

*This work has been partially supported by the CICYT project TIC95-0433-C03-02

style and some semantics for modules: the fixpoint semantics or $\mathcal{M}$-semantics, the $\mathcal{T}$-semantics, and the $(\mathcal{T}_\mathcal{P} \uparrow \omega)$-semantics obtained by denoting each module $\mathcal{P}$ by means of $\mathcal{T}_\mathcal{P}$ and $\mathcal{T} \uparrow \omega$ respectively, and the $\mathcal{LM}$-semantics using the set of all its term-models as the denotation of a module; and we study properties as compositionality, full abstraction and functorial behavior for these semantics. In this way, by means of the operator $\mathcal{T}_\mathcal{P}$, we relate the semantics given to modules in the main declarative paradigms.

In the next section we introduce the basic features of the CRWL approach to declarative programming. In Section 3, we introduce the model-theoretic and the fixpoint semantics. Our notion of module is given in Section 4 together with a set of operations on modules, and the $\mathcal{T}$-semantics in order to provide modules with a compositional and functorial semantics. Also the $(\mathcal{T} \uparrow \omega)$-semantics is introduced in this section. In Section 5, we introduce the $\mathcal{LM}$-semantics that is compositional and fully abstract. We finish with some conclusions in Section 6.

2 An Introduction to CRWL Programming

In this section we give a reduced presentation of CRWL —for a detailed presentation see [5, 9].

A *signature* (*with constructors*) is a pair $\Sigma = (DC_\Sigma, FS_\Sigma)$, where DC_Σ and FS_Σ are countable disjoint sets of strings h/n with $n \in \mathbf{N}$. Each c such that $c/n \in DC_\Sigma$ is a *constructor symbol* with arity n and each f such that $f/n \in FS_\Sigma$ is a *(defined) function symbol* with arity n. DC^n_Σ and FS^n_Σ denote the sets of all constructor and function symbols, respectively, with arity n. Together with a signature Σ, we also assume a countable set $\mathcal{V}$ of *variable symbols*, disjoint from all of the sets DC^n_Σ and FS^n_Σ.

In a signature Σ with a set $\mathcal{V}$ of variable symbols, each symbol in $\mathcal{V}$ and each symbol in DC^0_Σ is a Σ-term and, for each $h \in DC^n_\Sigma \cup FS^n_\Sigma$ and $t_1, \ldots, t_n$ Σ-term, $h(t_1, \ldots, t_n)$ is a Σ-term, and there is no other Σ-term. We will write $\mathbf{Term}_\Sigma$ for the set of all Σ-terms and $\mathbf{CTerm}_\Sigma$ for the subset of those Σ-terms (called *constructor terms*) built up only with symbols in DC_Σ and $\mathcal{V}$. Adding a new 0-arity constructor symbol $\bot$ to a signature Σ we obtain an extended signature $\Sigma_\bot$ whose terms are called *partial* Σ*-terms*. When the signature Σ is clear, we will omit explicit reference to it, and we will write **Term** and **CTerm** or $\mathbf{Term}_\bot$ and $\mathbf{CTerm}_\bot$ respectively. Terms $t \in \mathbf{CTerm}$ are intended to represent totally defined values and terms $t \in \mathbf{CTerm}_\bot$ represent partially defined values (to model the behavior of non-strict functions).

We consider two kinds of atomic *CRWL-formulas*: *reduction statements* $a \to b$, for $a, b \in \mathbf{Term}_\bot$, with the intended meaning "b *approximates* a possible value of a", and *joinability statements* $a \bowtie b$, for $a, b \in \mathbf{Term}_\bot$, with the intended meaning "a and b can be reduced to a common value in **CTerm**".

C-substitutions are mappings $\theta: \mathcal{V} \to \mathbf{CTerm}$, which have a natural extension to **CTerm**, also written as $\theta: \mathbf{CTerm} \to \mathbf{CTerm}$. Analogously, *partial C-substitutions* are mappings $\theta: \mathcal{V} \to \mathbf{CTerm}_\bot$. The set of all C-substitutions (partial C-substitutions) is written as **CSubst** ($\mathbf{CSubst}_\bot$).

A *CRWL-program* is a CRWL-theory $\mathcal{R}$ defined as a signature Σ together with a set of conditional rewrite rules of the form $f(\bar{t}) \to r \Leftarrow C$, where $f(\bar{t})$ is the left hand side (*lhs*), r the right hand side (*rhs*), and C the condition of the rule, f is a function symbol with arity n, $\bar{t}$ is a linear n-tuple (i.e. without repeated variables) of fully defined constructor terms, and C consists of finitely many (possibly zero) joinability statements between fully

defined terms. The set of possibly partial *constructor instances* of the rewrite rules of a program $\mathcal{R}$ is written

$$[\mathcal{R}]_\perp = \{(l \to r \Leftarrow C)\theta \mid (l \to r \Leftarrow C) \in \mathcal{R}, \theta \in \mathbf{CSubst}_\perp\}$$

Formal derivability of CRWL-statements from a given program $\mathcal{R}$ is governed by the so-called Goal-Oriented Proof Calculus [5], which focuses on top-down proofs of reduction and joinability statements:

(Bo)	$e \to \perp,$	for $e \in \mathbf{Term}_\perp$;
(RR)	$e \to e,$	for $e \in \mathcal{V} \cup DC^0$;
(DC)	$\dfrac{e_1 \to t_1 \ldots e_n \to t_n}{c(e_1,\ldots,e_n) \to c(t_1,\ldots,t_n)},$	for $c \in DC^n$ and $e_i, t_i \in \mathbf{Term}_\perp$;
(OR)	$\dfrac{e_1 \to t_1 \ldots e_n \to t_n \quad C \quad r \to t}{f(e_1,\ldots,e_n) \to t},$	if $t \not\equiv \perp$ and $(f(t_1,\ldots,t_n) \to r \Leftarrow C) \in [\mathcal{R}]_\perp$;
(Jo)	$\dfrac{a \to t \quad b \to t}{a \bowtie b},$	if $t \in \mathbf{CTerm}$ and $a, b \in \mathbf{Term}_\perp$;

Rule **(B0)** shows that a CRWL-reduction is related to the idea of approximation, and rule **(OR)** states that only constructor instances of rewrite rules are allowed in this calculus reflecting the so-called "call-time-choice" [7] for non-determinism (values of arguments for functions are chosen before the call is made). As a consequence of this, outermost rewriting is not sound in this framework and the lazy narrowing calculus presented in [5] incorporates sharing. When a statement φ is derivable from a program $\mathcal{R}$ we write $\mathcal{R} \vdash_{CRWL} \varphi$.

3 Model-theoretical and Fixpoint Semantics.

Given a signature Σ and a set $\mathcal{V}$ of variable symbols, we interpret CRWL-programs over algebraic structures $\mathcal{A} = (D_\mathcal{A}, \{c^\mathcal{A}\}_{c \in DC_\Sigma}, \{f^\mathcal{A}\}_{f \in FS_\Sigma})$ where $D_\mathcal{A}$ is a poset with bottom $\perp_\mathcal{A}$ (their elements are thought of as finite approximations of possibly infinite values in the poset's ideal completion [11]) and for each $f \in FS^n_\Sigma$, $f^\mathcal{A}$ is a monotonic mapping from $D^n_\mathcal{A}$ to the set of cones $\mathcal{C}(D_\mathcal{A})$ (to express non-determinism) and for each $c \in DC^n_\Sigma$, $c^\mathcal{A}$ is also a monotonic (deterministic) mapping from $D^n_\mathcal{A}$ to the set of ideals $\mathcal{I}(D_\mathcal{A})$ that computes principal ideals $\langle v \rangle$, i.e. generated by one element $v \in D_\mathcal{A}$, and such that v is maximal (totally defined) when all arguments of $c^\mathcal{A}$ are maximal. When $n = 0$, mappings reduce to elements in the respective target sets. The class of all the CRWL-algebra of signature Σ is denoted by $\mathbf{Alg}_\Sigma$.

As a specially kind of CRWL-algebras we have *CRWL-term algebras*, which are structures

$$\mathcal{A} = (\mathbf{CTerm}_\perp, \{c^\mathcal{A}\}_{c \in DC_\Sigma}, \{f^\mathcal{A}\}_{f \in FS_\Sigma})$$

with the carrier $\mathbf{CTerm}_\perp$, ordered by the *approximation ordering* "$\sqsubseteq$", defined as the least partial ordering satisfying the following properties:

(a)	$\perp \sqsubseteq t,$	$\forall t \in \mathbf{CTerm}_\perp$
(b)	$c(t_1,\ldots,t_n) \sqsubseteq c(s_1,\ldots,s_n)$	if $t_i \sqsubseteq s_i$, $\forall c \in DC^n_\Sigma$ and $t_i, s_i \in \mathbf{CTerm}_\perp$

and fixed interpretation for constructor symbols

$$c^{\mathcal{A}} = \langle c \rangle, \qquad \forall c \in DC^0_\Sigma;$$
$$c^{\mathcal{A}}(t_1, \ldots, t_n) = \langle c(t_1, \ldots, t_n) \rangle, \quad \forall c \in DC^n_\Sigma \text{ and } t_i \in \mathbf{CTerm}_\perp.$$

So, two CRWL-term algebras only differ in the interpretations of the function symbols.

A *valuation* over a structure $\mathcal{A} \in \mathbf{Alg}_\Sigma$ is any mapping $\eta: \mathcal{V} \to D_{\mathcal{A}}$. A valuation η is *totally defined* when $\eta(X)$ is maximal for all $X \in \mathcal{V}$. We denote by $\mathbf{Val}(\mathcal{A})$ the set of all valuations over $\mathcal{A}$, and by $\mathbf{DefVal}(\mathcal{A})$ the set of all totally defined valuations. Given a valuation η we can evaluate each partial Σ-term in $\mathcal{A}$ as follows:

$$\begin{array}{rll}
[\![\perp]\!]^{\mathcal{A}}_\eta & =_{def} \langle \perp_{\mathcal{A}} \rangle; & \\
[\![X]\!]^{\mathcal{A}}_\eta & =_{def} \langle \eta(X) \rangle, & \forall X \in \mathcal{V}; \\
[\![c]\!]^{\mathcal{A}}_\eta & =_{def} c^{\mathcal{A}}, & \forall c \in DC^0_\Sigma \bigcup FS^0_\Sigma; \\
[\![h(e_1, \ldots, e_n)]\!]^{\mathcal{A}}_\eta & =_{def} \hat{h}^{\mathcal{A}}([\![e_1]\!]^{\mathcal{A}}_\eta, \ldots, [\![e_n]\!]^{\mathcal{A}}_\eta), & \forall h \in DC^n_\Sigma \bigcup FS^n_\Sigma, n > 0.
\end{array}$$

Where $\hat{h}^{\mathcal{A}}$ is defined as

$$\hat{h}^{\mathcal{A}}(C_1, \ldots, C_n) = \bigcup_{u_i \in C_i} f(u_1, \ldots, u_n)$$

for cones $C_1, \ldots, C_n$ in $\mathcal{C}(D_{\mathcal{A}})$. In this way each partial Σ-term is evaluated to a cone.

3.1 Model-theoretical Semantics.

Approximation statements in a CRWL-algebra $\mathcal{A}$ are interpreted as approximation in the sense of $\mathcal{A}$'s partial ordering, and joinability statements as asserting the existence of some common totally defined approximation. With this idea in mind, we define

- $\mathcal{A} \models_\eta (a \to b)$ for a $\eta \in \mathbf{Val}(D_{\mathcal{A}})$, iff $[\![a]\!]^{\mathcal{A}}_\eta \supseteq [\![b]\!]^{\mathcal{A}}_\eta$;
- $\mathcal{A} \models_\eta (a \bowtie b)$ in $\eta \in \mathbf{Val}(D_{\mathcal{A}})$, iff $[\![a]\!]^{\mathcal{A}}_\eta \cap [\![b]\!]^{\mathcal{A}}_\eta$ contains a maximal element in $D_{\mathcal{A}}$;
- $\mathcal{A} \models (l \to r \Leftarrow C)$, iff $\mathcal{A} \models_\eta C$ implies $\mathcal{A} \models_\eta (l \to r)$, for all valuation $\eta \in \mathbf{Val}(D_{\mathcal{A}})$;
- $\mathcal{A}$ is a *model* of a program $\mathcal{R}$, i.e. $\mathcal{A} \models \mathcal{R}$, iff $\mathcal{A}$ satisfies all rules in $\mathcal{R}$.

For each program $\mathcal{R}$ there is a particular CRWL-term algebra $\mathcal{M}_{\mathcal{R}}$ characterized by the following interpretation for each defined function symbol $f \in FS^n_\Sigma$, $n \geq 0$

$$f^{\mathcal{M}_{\mathcal{R}}}(t_1, \ldots, t_n) =_{def} \{t \in \mathbf{CTerm}_\perp \mid \mathcal{R} \vdash_{CRWL} f(t_1, \ldots, t_n) \to t\},$$

for all $t_i \in \mathbf{CTerm}_\perp$, $i = 1, \ldots, n$, that is a model of the program and is freely generated by the set $\mathcal{V}$ of variable symbols in the category of all models of $\mathcal{R}$ with the following notion of homomorphism (*loose element-valued homomorphism* in [7])

Definition 3.1 *A homomorphism $h: \mathcal{A} \to \mathcal{B}$ from a CRWL-algebra $\mathcal{A}$ to another CRWL-algebra $\mathcal{B}$ is a monotonic function $h: \mathcal{A} \to \mathcal{I}(\mathcal{B})$, that is element-valued (produces principal ideals), strict ($h(\perp_{\mathcal{A}}) = \langle \perp_{\mathcal{B}} \rangle$) and preserves constructors $h(c^{\mathcal{A}}(u_1, \ldots, u_n)) = c^{\mathcal{B}}(h(u_1), \ldots, h(u_n))$, for all $c \in DC^n$ and $u_i \in D_{\mathcal{A}}$; and loosely preserves functions $h(f^{\mathcal{A}}(u_1, \ldots, u_n)) \subseteq f^{\mathcal{B}}(h(u_1), \ldots, h(u_n))$, for all $f \in FS^n$, and $u_i \in D_{\mathcal{A}}$. We also require totally definition: for all variable symbol $X \in \mathcal{V}$, $h(X) = \langle h_X \rangle$ is generated by a totally defined term h_X.*

According to these results, $\mathcal{M}_\mathcal{R}$ is taken in [5] as the *canonic model* of the program $\mathcal{R}$ or its *model-theoretical semantics* that we call $\mathcal{M}$-semantics: $[\![\mathcal{R}]\!] =_{def} \mathcal{M}_\mathcal{R}$.

3.2 Fixpoint Semantics.

Given a signature Σ with a set of variable symbols $\mathcal{V}$, we consider the set $\mathbf{TAlg}_\Sigma$ of all CRWL-term algebras of signature Σ with the relationship $\mathcal{A} \sqsubseteq \mathcal{B}$ between two algebras $\mathcal{A}, \mathcal{B} \in \mathbf{TAlg}_\Sigma$ defined by $f^\mathcal{A}(t_1, \ldots, t_n) \subseteq f^\mathcal{B}(t_1, \ldots, t_n)$, for all $t_i \in \mathbf{CTerm}_\perp$ and $f \in FS^n_\Sigma$. $(\mathbf{TAlg}_\Sigma, \sqsubseteq)$ is a complete lattice, with bottom $\perp_\Sigma$ and a top $\top_\Sigma$, respectively characterized by the interpretations

$$f^{\perp_\Sigma}(t_1, \ldots, t_n) =_{def} \langle \perp \rangle$$

and

$$f^{\top_\Sigma}(t_1, \ldots, t_n) =_{def} \mathbf{CTerm}_\perp$$

, for all $t_i \in \mathbf{CTerm}_\perp$ and $f \in FS^n_\Sigma, n \geq 0$; and least upper bound $\sqcup\mathbf{S}$ and greatest lower bound $\sqcap\mathbf{S}$ of any subset $\mathbf{S} \subseteq \mathbf{TAlg}_\Sigma$, given by

$$\begin{array}{lll} f^{\sqcup\mathbf{S}}(t_1, \ldots, t_n) & =_{def} & \bigcup_{\mathcal{A} \in \mathbf{S}} f^\mathcal{A}(t_1, \ldots, t_n), \\ f^{\sqcap\mathbf{S}}(t_1, \ldots, t_n) & =_{def} & \bigcap_{\mathcal{A} \in \mathbf{S}} f^\mathcal{A}(t_1, \ldots, t_n), \end{array}$$

for all $t_i \in \mathbf{CTerm}_\perp$ and $f \in FS^n_\Sigma$.

For every CRWL-program $\mathcal{R}$, with signature Σ and variables in $\mathcal{V}$, we can define an algebra transformer $\mathcal{T}_\mathcal{R}$: $\mathbf{TAlg}_\Sigma \to \mathbf{TAlg}_\Sigma$, similar to the immediate consequence operator used in logic programming, by fixing the interpretation of each function symbol $f \in FS^n_\Sigma$, in a transformed algebra $\mathcal{T}_\mathcal{R}(\mathcal{A})$, as the result of the one step application of rules of $\mathcal{R}$ satisfied in $\mathcal{A}$:

$$\begin{array}{l} f^{\mathcal{T}_\mathcal{R}(\mathcal{A})}(t_1, \ldots, t_n) =_{def} \\ \quad \{t \mid \exists (f(s_1, \ldots, s_n) \to r \Leftarrow C) \in [\mathcal{R}]_\perp, s_i \sqsubseteq t_i, \mathcal{A} \models_{id} C, t \in [\![r]\!]^\mathcal{A}_{id}\} \cup \{\perp\}, \end{array}$$

for all $t_i \in \mathbf{CTerm}_\perp$ and $f \in FS^n_\Sigma$. $\mathcal{T}_\mathcal{R}$ is continuous and satisfies the following results

Lemma 3.2 (Model characterization) *Given a program $\mathcal{R}$,*

$$\mathcal{M} \text{ is a term-model for } \mathcal{R} \iff \mathcal{T}_\mathcal{R}(\mathcal{M}) \sqsubseteq \mathcal{M}.$$

Proposition 3.3 *For every CRWL-program $\mathcal{R}$, $\mathcal{M}_\mathcal{R}$ is the least fixpoint of $\mathcal{T}_\mathcal{R}$.*

So, if we consider the meaning of a program $\mathcal{R}$ as the least fixpoint of its associated transformer $\mathcal{T}_\mathcal{R}$, then this semantics coincides with the $\mathcal{M}$-semantics as occurs in logic programming.

4 Modules in CRWL Programming

With the aim of structuring large CRWL-programs in small pieces and composing programs from other ones using traditional techniques as importation, instantiation and inheritance we introduce in [10] the following elementary notion of module in a global signature $\Sigma = (DC_\Sigma, FS_\Sigma)$, with a countable set $\mathcal{V}$ of variable symbols.

Definition 4.1 *A module is a program $\mathcal{P}$ over the signature Σ that declares a set of rules $rl(\mathcal{P})$ and a proper subsignature of defined function symbols $sig(\mathcal{P}) \subseteq FS_\Sigma$ that includes all function symbols with a definition rule in $rl(\mathcal{P})$. So we can imagine a module $\mathcal{P}$ as a pair $(sig(\mathcal{P}), rl(\mathcal{P}))$.*

Nevertheless, we can assume the whole set FS_Σ declared in $\mathcal{P}$ —although not produced by the *sig* operator— simply by assuming an implicit rule $f(\bar{t}) \to \bot$ for all those defined function symbols not explicitly declared in $\mathcal{P}$.

4.1 Module Composition in CRWL Programming

Following the approach in [1, 2], we define three operations: union of modules, closure w.r.t. a subsignature, and deletion of a subsignature.

Definition 4.2 (Union) *Given two modules $\mathcal{P}_1 = (\sigma_1, \mathcal{R}_1)$, $\mathcal{P}_2 = (\sigma_2, \mathcal{R}_2)$, their union is defined as the module:* $\mathcal{P}_1 \cup \mathcal{P}_2 =_{def} (\sigma_1 \cup \sigma_2, \mathcal{R}_1 \cup \mathcal{R}_2)$.

The arguments in this operation can be considered as open programs that can be extended or completed with other programs, possibly with additional rules for their function symbols. So, we could imagine the union as an "open" importation of modules.

The second operation is the *closure* of a module *w.r.t. a signature*, that is defined as a new module in which the rest of the signature has been hidden, and the mentioned signature is only accessible in an extensional way.

Definition 4.3 (Closure w.r.t. a signature) *The closure of a module $\mathcal{P}$ w.r.t. a signature σ, of function symbols, is defined as the module:*

$$\overline{\mathcal{P}}^{\sigma} =_{def} (sig(\mathcal{P}) \cap \sigma, \{f(\bar{t}) \to r \mid f/n \in \sigma, t_i, r \in \mathbf{CTerm}_{\Sigma_\bot}, rl(\mathcal{P}) \vdash_{CRWL} f(\bar{t}) \to r\}).$$

where t_i stands for each component of the tuple $\bar{t}$.

The closure of a module is a program with a possibly infinite set of rules equivalent to the union of the graphs of all the functions defined in $\mathcal{P}$ and contained in σ. As a syntactic simplification we will write $\overline{\mathcal{P}}$ instead of $\overline{\mathcal{P}}^{sig(\mathcal{P})}$.

Module closure provides us with a means of expressing encapsulation and restricting the signature (of defined function symbols) available outside a module. We can define an *export* —with encapsulation— operation $\Box$, in this simple way:

$$\sigma \Box \mathcal{M} = \overline{\mathcal{M}}^{\sigma}.$$

We can obtain a "closed" *import* operation $\ll$ between modules as:

$$\mathcal{M} \ll \mathcal{N} = \mathcal{M} \cup \overline{\mathcal{N}}.$$

This represents *global importation* from $\mathcal{N}$; but we can express *selective importation* of a signature σ by combining importation with exportation, and restricting the visible signature of the imported module:

$$\mathcal{M} \ll (\sigma \Box \mathcal{N}) = \mathcal{M} \cup \overline{\mathcal{N}}^{\sigma}.$$

Multiple importation or selective importation from a number of modules can be written as

$$(\ldots(\mathcal{M} \ll (\sigma_1 \Box \mathcal{N}_1)) \ll \ldots) \ll (\sigma_k \Box \mathcal{N}_k) \text{ or } \mathcal{M} \ll ((\sigma_1 \Box \mathcal{N}_1) \cup \ldots \cup (\sigma_k \Box \mathcal{N}_k)).$$

because the importation order is not relevant. So, a typical module $\mathcal{M}$, in the sense of standard modular programming, is built up from a plain module $\mathcal{P}$, several partially imported modules $\mathcal{N}_1, \ldots, \mathcal{N}_k$ and an exported signature σ as

$$\mathcal{M} = \sigma \Box (\mathcal{P} \ll (\sigma_1 \Box \mathcal{N}_1) \cup \ldots \cup (\sigma_k \Box \mathcal{N}_k)).$$

Our third operation is the *deletion of a signature* in a module.

Definition 4.4 (Deletion) *Given a module $\mathcal{P}$, the deletion of a signature of function symbols σ is the module: $\mathcal{P} \setminus \sigma =_{def} (sig(\mathcal{P}) \setminus \sigma, rl(\mathcal{P}) \setminus \sigma)$, where $rl(\mathcal{P}) \setminus \sigma$ denotes the set of those rules in $\mathcal{P}$ defining functions not appearing in σ.*

Note that the deletion of a signature removes all rules defining function symbols in the signature, maintaining the occurrences of these symbols in the rhs of the other rules; this represents a sort of *parameterization* in a signature that we denote by

$$\mathcal{M}(\sigma) = \mathcal{M} \setminus \sigma.$$

The *instantiation* of this module with another module $\mathcal{N}$ could be represented by

$$\mathcal{M}(\sigma) \ll \mathcal{N} =_{def} \mathcal{M} \setminus \sigma \cup \overline{\mathcal{N}},$$

and this instantiation could be partial depending on the function symbols defined in $\mathcal{N}$. Also, with this operation, it is possible to model a sort of inheritance between modules. *Inheritance with overriding* $\mathcal{M}$ *isa* $\mathcal{N}$ may be captured by means of the union and deletion of a signature

$$\mathcal{M} \, isa \, \mathcal{N} = \mathcal{M} \cup (\mathcal{N} \setminus sig(\mathcal{M})).$$

In this case, overriding is captured by deleting the signature of the inherited class before adding it to the derived class.

4.2 A Compositional Semantics for CRWL-Program Modules.

In order to state the compositionality and full abstraction for a semantics, we need a pair (Ob, Op) where Op is a set of operations and Ob a notion of observable behavior. A semantics $\mathcal{S}$ is *compositional* w.r.t. (Ob, Op) iff it is a *congruence* w.r.t. Op, i.e.

$$\mathcal{S}(\mathcal{P}_i) = \mathcal{S}(\mathcal{Q}_i)(i = 1 \ldots n) \Rightarrow \mathcal{S}(O(\mathcal{P}_1, \ldots, \mathcal{P}_n)) = \mathcal{S}(O(\mathcal{Q}_1, \ldots, \mathcal{Q}_n))$$

, for all programs $\mathcal{P}_i$ and $\mathcal{Q}_i$ and for all $O \in Op$, and *preserves* Ob, i.e.

$$\mathcal{S}(\mathcal{P}) = \mathcal{S}(\mathcal{Q}) \Rightarrow Ob(\mathcal{P}) = Ob(\mathcal{Q}),$$

for all programs $\mathcal{P}$ and $\mathcal{Q}$.

In our context the natural choices for the observable behavior of a program module $\mathcal{P}$ and for the set of operations are $Ob(\mathcal{P}) =_{def} \mathcal{M}_{\mathcal{P}}$ and $Op =_{def} \{\cup, \overline{(\cdot)}^{\sigma}, (\cdot) \setminus \sigma\}$. Notice that $\mathcal{M}_{\mathcal{P}}$ captures the graph of all functions defined in $\mathcal{P}$, whereas functions not included in

the program are considered totally undefined ($\bot$). With this notion, the $\mathcal{M}$-semantics for CRWL-modules is not compositional w.r.t. (Ob,Op) —see [10] for an counter-example. We get compositionality if we think about programs as open in the sense that we can build up programs from other programs adding rules for new functions and for already defined functions, and take them as algebra transformers as is suggested in [12] and proposed in [8, 1] for logic programming. In other words, it can be proved [10] that the $\mathcal{T}$-*semantics* defined by

$$[\![\mathcal{P}]\!] =_{def} \mathcal{T}_{\mathcal{P}},$$

where $\mathcal{T}_{\mathcal{P}}$ means $\mathcal{T}_{rl(\mathcal{P})}$, is compositional w.r.t. (Ob,Op), as it is stated in the next theorem. Previously, we have to note that the set $[\mathbf{TAlg}_\Sigma \rightarrow \mathbf{TAlg}_\Sigma]$ of all continuous functions from $\mathbf{TAlg}_\Sigma$ to $\mathbf{TAlg}_\Sigma$, pointwise ordered by $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ iff $\forall \mathcal{A} \in \mathbf{TAlg}_\Sigma \,.\, (\mathcal{T}_1(\mathcal{A}) \sqsubseteq \mathcal{T}_2(\mathcal{A}))$, with the lub and the glb, of a set $\{\mathcal{T}_i\}_{i\in I}$ of functions, pointwise defined as

$$(\sqcup_{i\in I}\mathcal{T}_i)(\mathcal{A}) = \sqcup_{i\in I}(\mathcal{T}_i(\mathcal{A})) \quad \text{and} \quad (\sqcap_{i\in I}\mathcal{T}_i)(\mathcal{A}) = \sqcap_{i\in I}(\mathcal{T}_i(\mathcal{A}))$$

respectively, and with bottom $\mathbb{T}_\bot$ and top $\mathbb{T}_\Sigma$ so that

$$\mathbb{T}_\bot(\mathcal{A}) = \bot_\Sigma \quad \text{and} \quad \mathbb{T}_\Sigma(\mathcal{A}) = \top_\Sigma \;\; \forall\, \mathcal{A} \in \mathbf{TAlg}_\Sigma,$$

is a complete lattice as a consequence of $(\mathbf{TAlg}_\Sigma, \sqsubseteq)$ being a complete lattice.

Theorem 4.5 (Compositionality of $[\![\cdot]\!]$) *For all programs $\mathcal{P}$, $\mathcal{P}_1$ and $\mathcal{P}_2$ defined over Σ, and all signature of function symbols $\sigma \subseteq FS_\Sigma$ we have*

(a) $$[\![\mathcal{P}_1 \cup \mathcal{P}_2]\!] = [\![\mathcal{P}_1]\!] \sqcup [\![\mathcal{P}_2]\!];$$

(b) $$[\![\overline{\mathcal{P}}^\sigma]\!] = \lambda\mathcal{A} \,.\, ([\![\mathcal{P}]\!] \uparrow \omega(\bot))|_\sigma;$$

(c) $$[\![\mathcal{P} \setminus \sigma]\!] = [\![\mathcal{P}]\!] \sqcap \mathbb{T}_{sig(\mathcal{P})\setminus\sigma};$$

where, for $\mathcal{A} \in \mathbf{TAlg}_\Sigma$ and $\sigma \subseteq FS_\Sigma$, $\mathcal{A}|_\sigma$ is the CRWL-algebra characterized by

$$f^{\mathcal{A}|_\sigma}(t_1,\ldots,t_n) = \begin{cases} f^{\mathcal{A}}(t_1,\ldots,t_n) & \text{for all } f \in \sigma, \text{ and } t_i \in \mathbf{CTerm}_{\Sigma_\bot} \\ \{\bot\} & \text{otherwise} \end{cases}$$

and, for each signature $\sigma \subseteq FS_\Sigma$, $\mathbb{T}_\sigma$ is the constant function defined as

$$\forall \mathcal{A} \in \mathbf{TAlg}_\Sigma, \quad f^{\mathbb{T}_\sigma(\mathcal{A})}(t_1,\ldots,t_n) = \begin{cases} \mathbf{CTerm}_{\Sigma_\bot} & \text{if } f \in \sigma \\ \{\bot\} & \text{otherwise} \end{cases}$$

4.3 The Functorial Nature of the $\mathcal{T}$-semantics.

In functional languages like ML —see [6]— functors are used to denote parameterized structures, characterized by an input and an output signature. The application of the functor to a structure matching the input signature yields a structure with the output signature. Therefore, these functors can be considered as algebra transformers. In our context, the distinction between input and output signatures has no sense because we work with a global signature, but our immediate consequence operator has a functorial behavior.

First, we have to note that, given a signature Σ with a set of variable symbols $\mathcal{V}$, $\mathbf{TAlg}_\Sigma$ is the set of objects for a category whose homomorphisms are totally defined.

Proposition 4.6 *The set* $\mathbf{TAlg}_\Sigma$ *of all CRWL-term algebras with signature* Σ *together with the sets of all totally defined homomorphism between two algebras constitutes a category* $\mathbf{TermAlg}_\Sigma$.

Proof. There is no problem in proving that the composition of two totally defined homomorphism $h: \mathcal{A} \to \mathcal{B}$ and $k: \mathcal{B} \to \mathcal{C}$, defined as $(k \cdot h)(t) = \hat{k}(h(t))$, is a totally defined homomorphism that satisfies the associative property. Also, it is easy to see that, for each $\mathcal{A} \in \mathbf{TAlg}_\Sigma$, the homomorphism $id_\mathcal{A}: \mathcal{A} \to \mathcal{A}$, defined as $id_\mathcal{A}(t) = \langle t \rangle$, is totally defined and verifies $h \cdot id_\mathcal{A} = h$ and $id_\mathcal{A} \cdot k = k$ for all homomorphisms $h: \mathcal{A} \to \mathcal{B}$ and $k: \mathcal{B} \to \mathcal{A}$. ■

To prove that $\mathcal{T}_\mathcal{P}$ is a functor we need a previous lemma.

Lemma 4.7 *Given an homomorphism* $h: \mathcal{A} \to \mathcal{B}$ *the following statements are true*

(a) *h induces a substitution* $\theta_h \in \mathbf{CSubst}_\perp$ *defined as* $\theta_h(X) = h_X$, *where* $h(X) = \langle h_X \rangle$, *such that* $h(t) = \langle t\theta_h \rangle$ *for all* $t \in \mathbf{CTerm}_\perp$;

(b) $t \in [\![r]\!]^\mathcal{A}_{id} \Rightarrow u\theta_h \in [\![r\theta_h]\!]^\mathcal{B}_{id}$;

(c) *if h is totally defined then* $\mathcal{A} \models_{id} s \bowtie t \Rightarrow \mathcal{B} \models_{id} s\theta_h \bowtie t\theta_h$.

Proof. Statements (a) and (b) are proved by induction on the structure of t and r respectively, and (c) is derived from (b) and the fact that θ_h is a totally defined valuation, which maps maximal elements into maximal elements. ■

With this lemma we can prove the following result, extending the immediate consequence operator in such a way that it could be considered as an endofunctor in the category $\mathbf{TermAlg}_\Sigma$.

Proposition 4.8 *For each module* $\mathcal{P}$ *we can extend the transformer* $\mathcal{T}_\mathcal{P}$ *to a functor from* $\mathbf{TermAlg}_\Sigma$ *to* $\mathbf{TermAlg}_\Sigma$ *that coincides with* $\mathcal{T}_\mathcal{P}$ *on the objects of the category and maps each homomorphism into itself:* $\mathcal{T}_\mathcal{P}(h) = h$, *for all homomorphism* $h: \mathcal{A} \to \mathcal{B}$.

Proof. We only need to prove that $h(f^{\mathcal{T}_\mathcal{P}(\mathcal{A})}(t_1, \ldots, t_n)) \subseteq f^{\mathcal{T}_\mathcal{P}(\mathcal{B})}(t_1\theta_h, \ldots, t_n\theta_h)$ for all homomorphism $h: \mathcal{A} \to \mathcal{B}$ in order to show that h is also an homomorphism from $\mathcal{T}_\mathcal{P}(\mathcal{A})$ to $\mathcal{T}_\mathcal{P}(\mathcal{B})$. But this follows from the above lemma taking an element in the left cone and proving that it is in the right one. The other functorial properties are immediate. ■

The functorial semantics used in algebraic specification languages, in a simplified way, denotes modules by functors which maps initial models of the input specification into initial models of the output specification [4]. Thus, a module importing another module can be considered as a transformer, which maps initial models of imported modules into initial models of the resulting module. In our context, we can express this behavior by requiring free term models of imported modules to be mapped into free term models of the resulting module (both with the same global signature). This requirement is not satisfied by our $\mathcal{T}$-semantics. However, we can see that the $\mathcal{T}_\mathcal{P} \uparrow \omega$ operator is still a functor like $\mathcal{T}_\mathcal{P}$, and moreover it fits the requirement.

Proposition 4.9 *For each module* $\mathcal{P}$ *the power* $\mathcal{T}_\mathcal{P} \uparrow \omega$ *of the transformer* $\mathcal{T}_\mathcal{P}$ *is also a functor from* $\mathbf{TermAlg}_\Sigma$ *to* $\mathbf{TermAlg}_\Sigma$, *defined for each* $\mathcal{A} \in \mathbf{TAlg}_\Sigma$ *as the function that produces the term algebra* $(\mathcal{T}_\mathcal{P} \uparrow \omega)(\mathcal{A})$ *characterized by the functions* $f^{(\mathcal{T}_\mathcal{P} \uparrow \omega)(\mathcal{A})}(t_1, \ldots, t_n) = \bigcup_{n \in \mathbf{N}} f^{\mathcal{T}_\mathcal{P}{}^n(\mathcal{A})}(t_1, \ldots, t_n)$, *and for each homomorphism* $h: \mathcal{A} \to \mathcal{B}$ *as* $(\mathcal{T}_\mathcal{P} \uparrow \omega)(h) = h$.

Proof. This proposition, like the one above, only need to be proved that

$$h(f^{(\mathcal{T}_\mathcal{P}\uparrow\omega)(\mathcal{A})}(t_1,\ldots,t_n)) \subseteq f^{(\mathcal{T}_\mathcal{P}\uparrow\omega)(\mathcal{B})}(t_1\theta_h,\ldots,t_n\theta_h)$$

and this is easy to do knowing that $\mathcal{T}_\mathcal{P}$ is a functor. ■

With our constructs for modularization we can consider several modes of importing a module $\mathcal{Q}$: *open importation* (or union) $\mathcal{P} \cup \mathcal{Q}$, where we can think about both modules as imported by the union module; *closed importation* $\mathcal{P} \ll \mathcal{Q} = \mathcal{P} \cup \overline{\mathcal{Q}}$; and *instantiation* $\mathcal{P}(\sigma) \ll \mathcal{Q} = (\mathcal{P} \setminus \sigma) \cup \overline{\mathcal{Q}}$. In all these modes the resultant module is obtained as the union of $\mathcal{Q}$, open or closed (but with the same free term model), with other module. So, the corresponding immediate consecuence operator associated to is, basically, the union of two operators, one corresponding to $\mathcal{Q}$ or $\overline{\mathcal{Q}}$, and the other one corresponding to the other module. With this idea we can proof the following proposition.

Proposition 4.10 *Given a module $\mathcal{N}$ that import another module $\mathcal{Q}$ in any of the above modes the functor $\mathcal{T}_\mathcal{N} \uparrow \omega$ maps the free term model of $\mathcal{Q}$ into the free term model of $\mathcal{N}$*

Proof. In all cases $\mathcal{M}_\mathcal{Q} \sqsubseteq \mathcal{T}_\mathcal{N}(\mathcal{M}_\mathcal{Q})$, the set $\{\mathcal{T}_\mathcal{N}{}^n(\mathcal{M}_\mathcal{Q}) \mid n \in \mathbf{N}\}$ is a chain and its supremum $\bigsqcup \mathcal{T}_\mathcal{N}{}^n(\mathcal{M}_\mathcal{Q}) = (\mathcal{T}_\mathcal{N} \uparrow \omega)(\mathcal{M}_\mathcal{Q})$ is a fixpoint for $\mathcal{T}_\mathcal{N}$ and a model for $\mathcal{N}$; so $\mathcal{M}_\mathcal{N} \sqsubseteq (\mathcal{T}_\mathcal{N} \uparrow \omega)(\mathcal{M}_\mathcal{Q})$. As $\mathcal{M}_\mathcal{N}$ is a model of $\mathcal{Q}$ and a fixpoint for $\mathcal{T}_\mathcal{N}$, we have that $(\mathcal{T}_\mathcal{N} \uparrow \omega)(\mathcal{M}_\mathcal{Q}) \sqsubseteq \mathcal{M}_\mathcal{N}$ and so $(\mathcal{T}_\mathcal{N} \uparrow \omega)(\mathcal{M}_\mathcal{Q}) = \mathcal{M}_\mathcal{N}$. ■

5 A Compositional and Fully Abstract Semantics.

To introduce the notion of full abstraction we need some way of distinguishing program modules. Two modules $\mathcal{P}$ and $\mathcal{Q}$ are distinguishable under a notion of observable behavior Ob and a set of operations Op if there exists a context $C[\![\cdot]\!]$ (constructed by using operations in Op), such that $C[\![\mathcal{P}]\!]$ and $C[\![\mathcal{Q}]\!]$ have different external behavior, i.e. $Ob(C[\![\mathcal{P}]\!]) \neq Ob(C[\![\mathcal{Q}]\!])$. We denote by $\mathcal{P} \cong \mathcal{Q}$ when $\mathcal{P}$ and $\mathcal{Q}$ are indistinguishable under (Ob, Op). A semantics $\mathcal{S}$ is *fully abstract* w.r.t. (Ob, Op) iff for all modules $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{P} \cong \mathcal{Q}$ implies $\mathcal{S}(\mathcal{P}) = \mathcal{S}(\mathcal{Q})$

Although the $\mathcal{T}$-semantics is compositional w.r.t. (Ob, Op) it is not fully abstract, as we can see with the following counter-example. Let Σ be a signature $(\{c/0, d/0\}, \{f/0\})$ and let $\mathcal{P}$ and $\mathcal{Q}$ be the programs:

$$rl(\mathcal{P}) = \{f \to c,\ f \to d\}, \quad rl(\mathcal{Q}) = \{f \to c,\ f \to d \Leftarrow f \bowtie c\}.$$

They are indistinguishable under Op, but they have not the same meaning with the $\mathcal{T}$-semantics; in fact, the graph of f interpreted in $\mathcal{T}_\mathcal{P}(\bot_\Sigma)$ is $\{(f, \{\bot, c, d\})\}$ whereas in $\mathcal{T}_\mathcal{Q}(\bot_\Sigma)$ is $(f, \{\bot, c\})$. This is because of the $\mathcal{T}$-semantics distinguishes more than the model-theoretic semantics, since the $\mathcal{T}$-operator captures what is happening in each reduction level. We obtain a compositional and fully abstract semantics w.r.t. (Ob, Op), by considering the set of all its term models as the meaning of a module $\mathcal{P}$

$$\{\!\{\mathcal{P}\}\!\} =_{def} \{\mathcal{M} : \mathcal{M} \text{ is a term-model of } \mathcal{P}\}$$

which will be called *loose model-theoretic semantics* or $\mathcal{LM}$-semantics. We derive compositionality and fully abstraction for this semantics as two theorems proved in [9].

6 Conclusions.

Functional-logic programs can be semantically characterized by applying the usual logical approach, based on an immediate consequence operator. This semantic characterization is convenient (compositional) for a particular (elementary) notion of module, when a set of basic operations are considered for composing modules. In this paper, we have shown a more functional reading for this semantics, proving the functorial nature of the immediate consequence operator $\mathcal{T}_{\mathcal{R}}$ and its power $\mathcal{T}_{\mathcal{R}} \uparrow \omega$ that have a transformational behavior between free term models as is usual in the algebraic and functional style, but unfortunately it is not compositional.

References

[1] A. Brogi. *Program Construction in Computational Logic.* Ph.D. Thesis: TD-2/93. Univ. Pisa-Genova-Udine, 1993.

[2] A. Brogi and F. Turini. Fully Abstract Compositional Semantics for an Algebra of Logic Programs. *TCS* 149 (2), pp. 201–229, 1995.

[3] M. Bugliesi, E. Lamma and P. Modularity in Logic Programming. *Journal of Logic Programming* 19,20, pp. 443–502, 1994.

[4] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2.* Springer-Verlag, 1990.

[5] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas and M. Rodríguez-Artalejo. A Rewriting Logic for Declarative Programming. In H. R. Nielson (ed.) *Proc. ESOP'96*, LNCS 1058, pp. 156–172. Springer, 1996.

[6] R. Harper, D.B. MacQueen and R. Milner. *Standard ML.* Report ECS-LFCS-86-2, Univ. of Edinburgh, 1986.

[7] H. Hussmann. *Non-determinism in Algebraic Specifications and Algebraic Programs.* Birkäuser Verlag, 1993.

[8] P. Mancarella and D. Pedreschi. An Algebra of Logic Programs. In R. A. Kowalski, A. Bowen (ed.) *Proc. Fifth ICLP*, pp. 1006–1023. MIT Press, 1988.

[9] J.M. Molina Bravo and E. Pimentel. *Modularity in CRWL-programming.* Tech. Rep., Dep. Lenguajes y C.C., Univ. de Málaga, 1996.

[10] J.M. Molina Bravo and E. Pimentel. "Modularity in Functional-Logic Programming". To appear in *Fourteenth Proc. of ICLP*, MIT Press, 1997.

[11] B. Möller. On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types. *Acta Informatica* 22, pp. 537–578, 1985.

[12] R. O'Keefe. "Towards an Algebra for Constructing Logic Programs". In *Proc. Second IEEE Symposium on Logic Programming*, pp. 152–160, 1985.