# *A Rewriting Logic for Declarative Programming* *

J.C. González-Moreno and M.T. Hortalá-González and
F.J. López-Fraguas and M. Rodríguez-Artalejo

Dpto. de Informática y Automática. Fac. Matemáticas, (UCM)
Av. Complutense s/n, Madrid (SPAIN), E-28040
**Email:** {jcmoreno,teresa,fraguas,mario}@dia.ucm.es

**Abstract.** We propose an approach to declarative programming which integrates the functional and relational paradigms by taking possibly non-deterministic lazy functions as the fundamental notion. Programs in our paradigm are theories in a constructor-based conditional rewriting logic. We present proof calculi and a model theory for this logic, and we prove the existence of free term models which provide an adequate intended semantics for programs. Moreover, we develop a sound and strongly complete lazy narrowing calculus, which is able to support sharing without the technical overhead of graph rewriting and to identify safe cases for eager variable elimination.

KEYWORDS: Declarative programming, non-deterministic functions, constructor-based rewriting logic, lazy narrowing.

## 1 Introduction

The interest in combining different declarative programming paradigms, especially functional and logic programming, has grown over the last decade; see [10] for a recent survey. The operational semantics of many functional logic languages is based on term rewriting and narrowing. In some cases, constructor-based term rewriting systems (CTRSs) have been adopted, in order to allow for a model-theoretic semantics that can reflect the behaviour of partial functions in non-strict functional languages. Typical examples of this approach include the languages K-LEAF [5] and BABEL [20]. To model the semantics of non-strict partial functions, these languages use so-called *strict equality*, which regards two terms as equal iff they have the same constructor normal form. On the other hand, the usefulness of non-deterministic operations for algebraic specification and programming has been advocated by Hussmann [13], who provides several examples (including the specification of semantics for communicating sequential processes) and leaves as an interesting open question "*the integration of non-strict operations (at least non-strict constructors)*" (see [13], Section 8.2).

Following Hussmann's spirit, we propose a quite general approach to declarative programming which views possibly non-deterministic lazy functions as the

* Research supported in part by the spanish CICYT (project TIC 95-0433-C03-01 " CPD ") and by the ESPRIT BR Working Group N. 6028 " CCL ".

fundamental notion. This is motivated by the fact that both relations and deterministic lazy functions are particular cases of non-deterministic lazy functions, while relations seem not expressive enough to model the behaviour of lazy functions. Our approach retains the advantages of deterministic functions in functional logic languages, such as dynamic cut [15] or simplification [11, 12]. But, in addition, non-deterministic functions can be modelled by means of non-confluent CTRSs, where a given term may be rewritten to constructor terms in more than one way. To express conditions and goals, strict equality is replaced by the more general notion of *joinability*: two terms a, b are regarded as joinable (in symbols, a $\bowtie$ b) iff they can be rewritten to a common constructor term. The following simple example program illustrates the expressivity of our formalism. We use Prolog's syntax for lists.

*Example 1.*

| | | |
|---|---|---|
| **constructors:** | zero/0, suc/1, []/0, [ _ \| _ ]/2 | |
| **functions:** | always/2, repetitions/2, copy/2, sublist/1 | |
| **rules:** | always(X,Y) | $\rightarrow$ [X,Y \| always(X,Y)]. |
| | repetitions(N, Xs) | $\rightarrow$ true $\Leftarrow$ copy(N, X) $\bowtie$ sublist(Xs). |
| | copy(zero, X) | $\rightarrow$ []. |
| | copy(suc(N), X) | $\rightarrow$ [X \| copy(N, X)]. |
| | sublist(Xs) | $\rightarrow$ []. |
| | sublist([X \| Xs]) | $\rightarrow$ [X \| sublist(Xs)]. |
| | sublist([X \| Xs]) | $\rightarrow$ sublist(Xs). |

The deterministic function "always" produces an infinite list from two given elements. The function "repetitions" represents a Prolog-like predicate which checks if a given list Xs contains at least N repeated occurrences of some element. The deterministic function "copy" is self-explanatory. The non-deterministic function "sublist" produces lists consisting of non-deterministically chosen elements from a given list Xs. The goal:

repetitions(suc(suc(zero)), always(zero,suc(zero))) $\bowtie$ true.

can be solved by the lazy narrowing calculus presented in Section 5.

In contrast to other lazy narrowing calculi, as e.g. those in [5, 20, 1, 11, 21], completeness of our calculus holds without any confluence or non-ambiguity hypothesis, even for conditional rewrite systems with extra variables in the conditions, which may cause incompleteness of narrowing w.r.t. the semantics of equational logic [19]. To gain this generality, we restrict ourselves to left-linear, constructor-based rewrite rules (which are expressive enough for programming), and we replace equational logic by a rewriting logic. We present proof calculi and a model theory for this logic, and we prove the existence of free term models which provide an adequate intended semantics for programs. As in [13], *sharing* turns out to be essential for the soundness of narrowing w.r.t to our model-theoretic semantics. Consequently, our lazy narrowing calculus is designed to

achieve sharing without the technical overhead of graph rewriting, as used in other approaches to functional logic programming with non-deterministic functions [13, 23].

For the sake of simplicity, we restrict our presentation to the unsorted case, but all our results can be straightforwardly extended to many-sorted signatures. Proofs are omitted due to lack of space, but they can be found in [8]. The paper is organized as follows. In Section 2 we recall some technical preliminaries. In Section 3 we introduce a Constructor-Based Conditional Rewriting Logic (CRWL) by means of two equivalent proof calculi. Section 4 is concerned with the model-theoretic semantics for CRWL programs, including the existence of free term models. In Section 5 we present a lazy narrowing calculus CLNC. In Section 6 we state the soundness and strong completeness of CLNC w.r.t. the model-theoretic semantics of CRWL. In Section 7 we summarize our conclusions and we argue the implementability of the approach.

## 2 Preliminaries

We fix here some basic notions, terminology and notations, needed for the rest of the paper.

A *poset* with bottom is a set $S$ equipped with a partial order $\sqsubseteq$ and a least element $\bot$ (w.r.t. $\sqsubseteq$). We say that an element $x \in S$ is *totally defined* (written $\mathsf{def}(x)$) iff $x$ is maximal w.r.t. $\sqsubseteq$. The set of all totally defined elements of $S$ will be noted $\mathsf{Def}(S)$. $D \subseteq S$ is a *directed set* iff for all $x, y \in D$ there exists $z \in D$ with $x \sqsubseteq z$, $y \sqsubseteq z$. A subset $A \subseteq S$ is a *cone* iff $\bot \in A$, and $A$ is *downclosed*, i.e., $y \sqsubseteq x \Rightarrow y \in A$, for all $x \in A$, $y \in S$. An *ideal* $I \subseteq S$ is a directed cone. We write $\mathcal{C}(S)$, $\mathcal{I}(S)$ for the sets of cones and ideals of $S$ respectively. The set $\bar{S} =_{def} \mathcal{I}(S)$, equipped with the set-inclusion $\subseteq$ as ordering, is a poset with bottom called the *ideal completion* of $S$. There is a natural, order-preserving embedding of $S$ into $\bar{S}$, which maps each $x \in S$ into the principal ideal generated by $x$, $< x >=_{def} \{y \in S : y \sqsubseteq x\} \in \bar{S}$. A poset with bottom $C$ is a *cpo* iff $D$ has a least upper bound $\sqcup D$ (also called *limit*) for every directed set $D \subseteq C$. $u \in C$ is a *finite* element (written $\mathsf{fin}(u)$) if whenever $u \sqsubseteq \sqcup D$ for a directed $D$, there exists $x \in D$ with $u \sqsubseteq x$. A cpo $C$ is called *algebraic* if any element of $C$ is the limit of a directed set of finite elements. For any poset with bottom $S$, its ideal completion $\bar{S}$ turns to be the least cpo including $S$. Furthermore, $\bar{S}$ is an algebraic cpo whose finite elements are precisely the principal ideals $\langle x \rangle$, $x \in S$; see e.g. [18]. Note that elements $x \in \mathsf{Def}(S)$ correspond to finite and maximal elements $\langle x \rangle$ in the ideal completion.

A *signature with constructors* is a countable set $\Sigma = \mathsf{DC}_\Sigma \cup \mathsf{FS}_\Sigma$, where $\mathsf{DC}_\Sigma = \bigcup_{n \in \mathbb{N}} \mathsf{DC}^n_\Sigma$ and $\mathsf{FS}_\Sigma = \bigcup_{n \in \mathbb{N}} \mathsf{FS}^n_\Sigma$ are disjoint sets of *constructor* and *defined function symbols* respectively, each of them with associated *arity*. We assume a countable set $\mathcal{V}$ of *variables*, and we omit explicit mention of $\Sigma$ in the subse-

quent notations. We write Term for the set of terms built up with aid of $\Sigma$ and $\mathcal{V}$, and we distinguish the subset CTerm of those terms (called *constructor terms*, shortly C-terms) which only make use of DC and $\mathcal{V}$. We will need sometimes to enhance $\Sigma$ with a new constant (0-arity constructor) $\bot$, obtaining a new signature $\Sigma_\bot$. We will write $\mathsf{Term}_\bot$ and $\mathsf{CTerm}_\bot$ for the corresponding sets of terms in this extended signature, the so-called *partial* terms. As frequent notational conventions we will also use c,d $\in$ DC; f,g $\in$ FS; s,t $\in \mathsf{CTerm}_\bot$; a,b,e $\in \mathsf{Term}_\bot$.

A natural *approximation ordering* " $\sqsubseteq$ " for partial C-terms can be defined as follows: $\sqsubseteq$ is the least partial ordering over $\mathsf{CTerm}_\bot$ satisfying the following properties:

- $\bot \sqsubseteq t$, for all $t \in \mathsf{CTerm}_\bot$
- $t_1 \sqsubseteq s_1, \ldots, t_n \sqsubseteq s_n \Rightarrow c(t_1,\ldots,t_n) \sqsubseteq c(s_1, \ldots, s_n)$, for all $c \in DC^n$, $t_i, s_i \in \mathsf{CTerm}_\bot$

The ideal completion of $\mathsf{CTerm}_\bot$ is isomorphic to a cpo whose elements are possibly infinite trees with nodes labelled by symbols from DC $\cup\{\bot\}$ in such a way that the arity of each label corresponds to the number of sons of the node; see [6].

*C-substitutions* are mappings $\theta : \mathcal{V} \to \mathsf{CTerm}$ which have a unique natural extension $\hat{\theta}$: Term $\to$ Term, also noted as $\theta$. The set of all C-substitutions is noted as CSubst. The bigger set $\mathsf{CSubst}_\bot$ of all partial C-substitutions $\theta : \mathcal{V} \to \mathsf{CTerm}_\bot$ is defined analogously. We note as $t\theta$ the result of applying the substitution $\theta$ to the term t, and we define the composition $\sigma\theta$ such that $t(\sigma\theta) \equiv (t\sigma)\theta$. A substitution $\theta$ such that $\theta\theta = \theta$ is called *idempotent*. The approximation ordering over $\mathsf{CTerm}_\bot$ induces a natural approximation ordering over $\mathsf{CSubst}_\bot$, defined by the condition: $\theta \sqsubseteq \theta'$ iff $X\theta \sqsubseteq X\theta'$, for all $X \in \mathcal{V}$. We will also use the *subsumption ordering* over $CSubst_\bot$, defined by: $\theta \leq \theta'$ iff $\theta' = \theta\sigma$ for some $\sigma$. Finally, the notation $\theta \leq \theta'$ $[\mathcal{U}]$, where $\mathcal{U} \subseteq \mathcal{V}$, means that $X\theta' \equiv X(\theta\sigma)$ for some $\sigma$ and for all $X \in \mathcal{U}$ (i.e, $\theta$ is more general than $\theta'$ over the variables in $\mathcal{U}$).

## 3 A Constructor Based Conditional Rewriting Logic

In this section we give a proof-theoretical presentation of CRWL, a constructor based conditional rewriting logic which we propose as logical framework for our approach to declarative programming. In spite of some obvious similarities, CRWL differs from Meseguer's rewriting logic [17] both in its semantics and its intended applications. Meseguer's logic aims at modelling change caused by concurrent actions at a very high abstraction level, while CRWL intends to model the evaluation of terms in a constructor-based language involving possibly non-deterministic lazy functions. We introduce two kinds of CRWL-formulas:

For given $a,b \in \mathsf{Term}_\bot$, the *reduction statement* $a \to b$ is intended to mean that a can be reduced to b, and the *joinability statement* $a \bowtie b$ is intended to mean that a, b can be reduced to some common totally defined value. We allow for

partial terms in CRWL-statements because we want to model the behaviour of non-strict functions. Partial C-terms $t \in \mathsf{CTerm}_\perp$ are intended to represent finite approximations of values. Thus, reduction statements $a \to t$ with $t \in \mathsf{CTerm}_\perp$ will be called *approximation statements*. Their intended meaning is that t approximates a value of a (we must say "*a value*" due to non-determinism).

CRWL-theories, which will be called simply *programs* in the rest of the paper, are defined as sets $\mathcal{R}$ of conditional rewrite rules of the form:

$$\underbrace{f(\bar{t})}_{\text{left hand side (l)}} \quad \to \quad \underbrace{r}_{\text{right hand side}} \quad \Leftarrow \quad \underbrace{C}_{\text{Condition}}$$

where $f \in \mathsf{FS}$, $\bar{t}$ must be a linear n-tuple of C-terms $t_i \in \mathsf{CTerm}$, $r \in \mathsf{Term}$ and the condition C must consist of finitely many (possibly zero) joinability statements $a \bowtie b$ with $a, b \in \mathsf{Term}$. In the sequel we use the following notation for possibly partial C-instances of rewrite rules:

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

Formal derivability of CRWL-statements from a given program $\mathcal{R}$ is governed by the following calculus:

**Definition 1.** *Basic Proof Calculus* (BPC)

**(B)** *Bottom:* $e \to \perp$

**(MN)** *Monotonicity:* $\dfrac{e_1 \to e'_1 \;\ldots\; e_n \to e'_n}{h(e_1, \ldots, e_n) \to h(e'_1, \ldots, e'_n)}$ for $h \in \mathsf{DC}^n \cup \mathsf{FS}^n$.

**(RF)** *Reflexivity:* $e \to e$

**(R)** *Reduction:* $\dfrac{C}{l \to r}$ for any instance $(l \to r \Leftarrow C) \in [\mathcal{R}]_\perp$.

**(TR)** *Transitivity:* $\dfrac{e \to e' \quad e' \to e''}{e \to e''}$

**(J)** *Join:* $\dfrac{a \to t \quad b \to t}{a \bowtie b}$ if $t \in \mathsf{CTerm}$. □

As shown by rule **(B)**, a CRWL-reduction is related to the idea of approximation. In rule **(J)**, the C-term $t$ must be total, since we wish to specify joinability as a generalization of strict equality; as we will see in the next Section, terms $t \in \mathsf{CTerm}$ always denote totally defined elements in CRWL-models.

Rule **(R)** shows another important difference w.r.t. rewriting in the usual sense, which would allow to apply arbitrary instances of rewrite rules, instead of C-instances. This reflects so-called "*call-time-choice*" for non-determinism, meaning that values of arguments for a function are chosen before the call; see [13], Chapter 1. For this reason, outermost rewriting is not sound in our framework, and the lazy narrowing calculus presented in Section 5 will incorporate sharing. The following example, inspired by Hussmann [13], illustrates this point:

*Example 2.*

| | | |
|---|---|---|
| **constructors:** | zero/0, suc/1 | |
| **functions:** | coin/0, double/1, add/2 | |
| **rules:** | coin | → zero. |
| | coin | → suc(zero). |
| | add(zero, Y) | → Y. |
| | add(suc(X), Y) | → suc(add(X,Y)). |
| | double(X) | → add(X, X). |

Observe that in this example the following statements are deduced in our logic:

$$\mathsf{double(coin)} \rightarrow \mathsf{zero}. \qquad \mathsf{double(coin)} \rightarrow \mathsf{suc(suc(zero))}.$$

But not the following:

$$\mathsf{double(coin)} \rightarrow \mathsf{add(coin,coin)}. \qquad \mathsf{double(coin)} \rightarrow \mathsf{suc(zero)}.$$

In order to prepare the use of CRWL as a logic for declarative programming, we introduce a second calculus which focuses on top-down proofs of approximation and joinability statements.

**Definition 2.** *Goal-Oriented Proof Calculus* (GPC)

**(B)** *Bottom*: $e \rightarrow \bot$.
**(RR)** *Restricted Reflexivity*: $e \rightarrow e$, if $e \in \mathcal{V} \cup DC^0$.
**(DC)** *Decomposition*:
$\dfrac{e_1 \rightarrow t_1 \ldots e_n \rightarrow t_n}{c(e_1, \ldots, e_n) \rightarrow c(t_1, \ldots, t_n)}$, for $c \in DC^n_\Sigma$, $t_i \in Term_\bot$.
**(OR)** *Outer Reduction*:
$\dfrac{e_1 \rightarrow t_1 \ldots e_n \rightarrow t_n \quad C \quad r \rightarrow t}{f(e_1, \ldots, e_n) \rightarrow t}$, if $t \not\equiv \bot$, $(f(t_1, \ldots, t_n) \rightarrow r \Leftarrow C) \in [\mathcal{R}]_\bot$.
**(J)** *Join*: $\dfrac{a \rightarrow t \quad b \rightarrow t}{a \bowtie b}$, if $t \in CTerm$. □

By induction on the structure of proofs, we can prove:

**Proposition 3.** *(Calculi Equivalence)*
*For any program $\mathcal{R}$, the calculi BPC and GPC derive the same approximation and joinability statements.* ■

In the rest of the paper, the notation $\mathcal{R} \vdash_{CRWL} \varphi$ will mean provability of $\varphi$ (an approximation or joinability statement) in any of the calculi BPC or GPC. For instance, if $\mathcal{R}$ is the program from Example 1, the following statements are derivable:

$$\mathsf{always(zero,suc(zero))} \rightarrow \mathsf{[zero,suc(zero),zero,suc(zero)} \mid \bot \mathsf{]}$$
$$\mathsf{sublist(always(zero,suc(zero)))} \rightarrow \mathsf{[zero,zero]}$$
$$\mathsf{copy(suc(suc(zero)),\ zero)} \bowtie \mathsf{sublist(always(zero,suc(zero)))}$$

# 4 Model-theoretic Semantics for CRWL programs

In this section we define models for CRWL and we establish soundness and completeness of CRWL-provability w.r.t. semantic validity in models. Moreover, we prove that every program has a free term model, which can be seen as a generalization of $\mathcal{C}$-semantics [4] for Horn clause programs.

To represent non-deterministic functions we use models with posets as carriers, and we interpret functions symbols as monotonic mappings from elements to cones. The poset's elements are best thought as finite approximations of possibly infinite values in the poset's ideal completion (see Section 2). The technical definitions are as follows:

**Definition 4.** (Non-deterministic and deterministic functions)
Given two posets with bottom D, E, we define:

- The set of all non-deterministic functions from D to E as:
  $[\mathsf{D} \to_n \mathsf{E}] =_{def} \{\ \mathsf{f}\colon \mathsf{D} \to \mathcal{C}(\mathsf{E}) \mid \forall\ \mathsf{u},\mathsf{u}' \in \mathsf{D}\colon (\mathsf{u} \sqsubseteq \mathsf{u}' \Rightarrow \mathsf{f}(\mathsf{u}) \subseteq \mathsf{f}(\mathsf{u}'))\ \}$.
- The set of all deterministic functions from D to E as the subset of $[\mathsf{D} \to_n \mathsf{E}]$ specified as: $[\mathsf{D} \to_d \mathsf{E}] =_{def} \{\ \mathsf{f} \in [\mathsf{D} \to_n \mathsf{E}] \mid \forall\ \mathsf{u} \in \mathsf{D}\colon \mathsf{f}(\mathsf{u}) \in \mathcal{I}(\mathsf{E})\ \}$.

□

Given any fixed $\mathsf{u} \in \mathsf{D}$, a deterministic function f computes a directed set $\mathsf{f}(\mathsf{u})$ of partial values; thus, such functions become continuous mappings between algebraic cpos, after performing an ideal completion. Note also that any non-deterministic function $\mathsf{f} \in [\mathsf{D} \to_n \mathsf{E}]$ can be extended to a monotonic mapping $\hat{f} : \mathcal{C}(\mathsf{D}) \to \mathcal{C}(\mathsf{E})$ defined by $\hat{f}(\mathsf{C}) =_{def} \bigcup_{\mathsf{u} \in \mathsf{C}} \mathsf{f}(\mathsf{u})$. By an abuse of notation, we will note $\hat{f}$ also as f in the sequel.

Next, we define the class of algebras which will be used as models for CRWL:

**Definition 5.** (CRWL-Algebras)
Let $\Sigma$ be any given signature. *CRWL-algebras* of signature $\Sigma$ are algebraic structures of the form:

$$\mathcal{A} = (\mathsf{D}_{\mathcal{A}},\ \{\ \mathsf{c}^{\mathcal{A}}\ \}_{\mathsf{c} \in \mathsf{DC}_{\Sigma}},\ \{\ \mathsf{f}^{\mathcal{A}}\ \}_{\mathsf{f} \in \mathsf{FS}_{\Sigma}})$$

where: $\mathsf{D}_{\mathcal{A}}$ is a poset, $\mathsf{c}^{\mathcal{A}} \in [\mathsf{D}_{\mathcal{A}}^n \to_d \mathsf{D}_{\mathcal{A}}]$ for $\mathsf{c} \in \mathsf{DC}_{\Sigma}^n$, and $\mathsf{f}^{\mathcal{A}} \in [\mathsf{D}_{\mathcal{A}}^n \to_n \mathsf{D}_{\mathcal{A}}]$ for $\mathsf{f} \in \mathsf{FS}_{\Sigma}^n$. For $\mathsf{c}^{\mathcal{A}}$ we still require an additional condition, in order to ensure preservation of finite and maximal elements in the ideal completion of $\mathsf{D}_{\mathcal{A}}$:

For all $\mathsf{u}_1, \ldots, \mathsf{u}_n \in \mathsf{D}_{\mathcal{A}}$ there is $\mathsf{v} \in \mathsf{D}_{\mathcal{A}}$ such that $\mathsf{c}^{\mathcal{A}}(\mathsf{u}_1, \ldots, \mathsf{u}_n) = \langle\ \mathsf{v}\ \rangle$.
Moreover, $\mathsf{v} \in \mathsf{Def}(\mathsf{D}_{\mathcal{A}})$ in case that all $\mathsf{u}_i \in \mathsf{Def}(\mathsf{D}_{\mathcal{A}})$. □

A *valuation* over a structure $\mathcal{A}$ is any mapping $\eta\colon \mathcal{V} \to \mathsf{D}_{\mathcal{A}}$, and we say that $\eta$ is *totally defined* iff $\eta(\mathsf{X}) \in \mathsf{Def}(\mathsf{D}_{\mathcal{A}})$ for all $\mathsf{X} \in \mathcal{V}$. We denote by $\mathsf{Val}(\mathcal{A})$ the set of all valuations, and by $\mathsf{DefVal}(\mathcal{A})$ the set of all totally defined valuations. The *evaluation* of a partial term $\mathsf{e} \in \mathsf{Term}_{\perp}$ in $\mathcal{A}$ under $\eta$ yields $[\![\ \mathsf{e}\ ]\!]^{\mathcal{A}}\eta \in \mathcal{C}(\mathsf{D}_{\mathcal{A}})$ which is defined recursively as follows:

- $[\![ \bot ]\!]^{\mathcal{A}}\eta =_{def} (\bot_{\mathcal{A}})$.
- $[\![ X ]\!]^{\mathcal{A}}\eta =_{def} < \eta(X) >$, for $X \in \mathcal{V}$.
- $[\![ h(e_1, \ldots, e_n) ]\!]^{\mathcal{A}}\eta =_{def} h^{\mathcal{A}}([\![ e_1 ]\!]^{\mathcal{A}}\eta, \ldots, [\![ e_n ]\!]^{\mathcal{A}}\eta)$, for all $h \in DC^n_\Sigma \cup FS^n_\Sigma$.

The following result is easy to prove by structural induction:

**Proposition 6.**
*Given an CRWL-algebra $\mathcal{A}$, for any $e \in Term_\bot$ and any $\eta \in Val(\mathcal{A})$:*

*a)* $[\![ e ]\!]^{\mathcal{A}}\eta \in \mathcal{C}(D_{\mathcal{A}})$.
*b)* $[\![ e ]\!]^{\mathcal{A}}\eta \in \mathcal{I}(D_{\mathcal{A}})$ *if* $f^{\mathcal{A}}$ *is deterministic for every defined function symbol* $f$ *occurring in* $e$.
*c)* $[\![ e ]\!]^{\mathcal{A}}\eta = \langle v \rangle$ *for some* $v \in D_{\mathcal{A}}$, *if* $e \in CTerm_\bot$. *Moreover,* $v \in Def(D_{\mathcal{A}})$ *if* $e \in CTerm$ *and* $\eta \in DefVal(\mathcal{A})$. ∎

We are now prepared to introduce models. The main idea is to interpret approximation statements in $\mathcal{A}$ as approximation in the sense of $\mathcal{A}$'s partial ordering, and to interpret joinability statements as asserting the existence of some common, totally defined approximation.

**Definition 7.** (Models)
Assume a program $\mathcal{R}$ and an CRWL-algebra $\mathcal{A}$. We define:

- $\mathcal{A}$ is a *model* of $\mathcal{R}$ (in symbols $\mathcal{A} \models \mathcal{R}$) iff $\mathcal{A}$ satisfies all the rules in $\mathcal{R}$.
- $\mathcal{A}$ *satisfies* a rule $l \to r \Leftarrow C$ iff every valuation $\eta$ such that $\mathcal{A} \models C\eta$ verifies $\mathcal{A} \models (l \to r)\eta$.
- $\mathcal{A}$ *satisfies* a reduction statement $a \to b$ under a valuation $\eta$ ($\mathcal{A} \models (a \to b)\eta$) iff $[\![ a ]\!]^{\mathcal{A}}\eta \supseteq [\![ b ]\!]^{\mathcal{A}}\eta$ .
- $\mathcal{A}$ *satisfies* a joinability statement $a \bowtie b$ under a valuation $\eta$ ($\mathcal{A} \models (a \bowtie b)\eta$) iff $[\![ a ]\!]^{\mathcal{A}}\eta \cap [\![ b ]\!]^{\mathcal{A}}\eta \cap Def(D_{\mathcal{A}}) \neq \emptyset$. □

CRWL-provability is sound w.r.t. models in the following sense:

**Theorem 8.** *(Soundness)*
*For any program $\mathcal{R}$ and any reduction or joinability statement $\varphi$:*

$$\mathcal{R} \vdash \varphi \Rightarrow \mathcal{A} \models \varphi\eta, \text{ for all } \mathcal{A} \models \mathcal{R} \text{ and all } \eta \in DefVal(\mathcal{A}).$$ ∎

Completeness of $\vdash_{CRWL}$ can be proved with the help of term models, that are closely related to CRWL-provability.

**Definition 9.** (Term Model)
Given a program $\mathcal{R}$, the term model $\mathcal{M}_{\mathcal{R}}$ is defined as follows:

- $D_{\mathcal{M}_{\mathcal{R}}}$ is the poset $CTerm_\bot$ with approximation ordering $\sqsubseteq$.
- $c^{\mathcal{M}_{\mathcal{R}}}(t_1, \ldots, t_n) =_{def} < c(t_1, \ldots, t_n) >$ (principal ideal), for all $t_i \in CTerm_\bot$.
- $f^{\mathcal{M}_{\mathcal{R}}}(t_1, \ldots, t_n) =_{def} \{ t \in CTerm_\bot \mid \mathcal{R} \vdash_{CRWL} f(t_1, \ldots, t_n) \to t \}$, for all $t_i \in CTerm_\bot$. □

$\mathcal{M}_\mathcal{R}$ is a well-defined CRWL-algebra. In fact, $f^{\mathcal{M}_\mathcal{R}}$ is monotonic (as required by definitions 4,5) as a consequence of the next lemma, which will be useful again in Section 6. The Lemma can be proved by induction on the structure of GPC-proofs.

**Lemma 10.** *(Monotonicity Lemma)*
*Let $\mathcal{R}$ be a program, $e \in \mathsf{Term}_\perp$, $\theta, \theta' \in CSubst_\perp$, and $t \in \mathsf{CTerm}_\perp$. If $\theta' \sqsupseteq \theta$ and $\Pi$ is a GPC-proof of $\mathcal{R} \vdash_{CRWL} e\theta \to t$, there exists a GPC-proof $\Pi'$ of $\mathcal{R} \vdash_{CRWL} e\theta' \to t$ with the same length and structure as $\Pi$.* ∎

The relationship between semantic validity in $\mathcal{M}_\mathcal{R}$ and $\vdash_{CRWL}$ provability is revealed by the following lemma, whose proof can be obtained by laborious, but routine inductions. Note that partial C-substitutions are the same as valuations over $\mathcal{M}_\mathcal{R}$.

**Lemma 11.** *(Characterization of validity in $\mathcal{M}_\mathcal{R}$)*
*Let $\theta$ be any partial C-substitution. Then, for all $e, a, b \in \mathsf{Term}_\perp$, $t \in \mathsf{CTerm}_\perp$:*

*a)* $t \in [\![ e ]\!]^{\mathcal{M}_\mathcal{R}}\theta \Leftrightarrow \mathcal{R} \vdash_{CRWL} e\theta \to t$. *b)* $\mathcal{M}_\mathcal{R} \models (a \bowtie b)\theta \Leftrightarrow \mathcal{R} \vdash a\theta \bowtie b\theta$. ∎

With the help of Lemma 11 we can prove the main result relating provability and models in CRWL:

**Theorem 12.** *(Adequateness of $\mathcal{M}_\mathcal{R}$)*
*$\mathcal{M}_\mathcal{R}$ is a model of $\mathcal{R}$. Moreover, for any approximation or joinability statement $\varphi$, the following conditions are equivalent:*

*a)* $\mathcal{R} \vdash \varphi$.
*b)* $\mathcal{A} \models \varphi\eta$ *for every* $\mathcal{A} \models \mathcal{R}$, *and every* $\eta \in \mathsf{DefVal}(\mathcal{A})$.
*c)* $\mathcal{M}_\mathcal{R} \models \varphi\ \mathsf{id}$, *where* id *is the identity valuation, defined by* $\mathsf{id}(X) = X$ *for all* $X \in \mathcal{V}$. ∎

Note that the completeness of $\vdash_{CRWL}$ also follows from Theorem 12. According to this result, $\mathcal{M}_\mathcal{R}$ can be regarded as the intended (canonic) model of program $\mathcal{R}$. In particular, a given $f \in \mathsf{FS}$ will denote a deterministic function iff $f^{\mathcal{M}_\mathcal{R}}(t_1, \ldots, t_n)$ is an ideal for all $t_i \in \mathsf{CTerm}_\perp$. This property is undecidable in general, but some decidable sufficient conditions are known which work quite well in practice; see e.g. the sufficient non-ambiguity conditions in [7].

To state the last result in this section, we need homomorphisms between CRWL-algebras. There exist several known homomorphism notions for algebras involving non-deterministic operations, see [13], Chapter 3. Our definition follows the idea of *loose element-valued homomorphisms*, in Hussmann's terminology.

**Definition 13.** (Homomorphisms)
Let $\mathcal{A}$, $\mathcal{B}$ two given CRWL-algebras. A *homomorphism* $h: \mathcal{A} \to \mathcal{B}$ is any deterministic function $h \in [D_\mathcal{A} \to_d D_\mathcal{B}]$ which satisfies the following conditions:

- h is *element-valued*: for all $u \in D_{\mathcal{A}}$ there is $v \in D_{\mathcal{B}}$ such that $h(u) = \langle v \rangle$.
- h is *strict*: $h(\bot_{\mathcal{A}}) = \langle \bot_{\mathcal{B}} \rangle$.
- h preserves constructors: for all $c \in DC^n$, $u_i \in D_{\mathcal{A}}$: $h(c^{\mathcal{A}}(u_1, \ldots, u_n)) = c^{\mathcal{B}}(h(u_1), \ldots, h(u_n))$.
- h *loosely* preserves defined functions: for all $f \in FS^n$, $u_i \in D_{\mathcal{A}}$: $h(f^{\mathcal{A}}(u_1, \ldots, u_n)) \subseteq f^{\mathcal{B}}(h(u_1), \ldots, h(u_n))$. □

**Theorem 14.** *(Freeness of $\mathcal{M}_{\mathcal{R}}$)*
*For any program $\mathcal{R}$, the term model $\mathcal{M}_{\mathcal{R}}$ is freely generated by $\mathcal{V}$ in the category of all models of $\mathcal{R}$; that is, given any $\mathcal{A} \models \mathcal{R}$ and any valuation $\eta \in \mathrm{DefVal}(\mathcal{A})$, there is a unique homomorphism* h: $\mathcal{M}_{\mathcal{R}} \rightarrow \mathcal{A}$ *extending $\eta$ in the sense that* $h(X) = \langle \eta(X) \rangle$ *for all* $X \in \mathcal{V}$. ■

The proof of Theorem 14 follows basically from Theorem 12. By a construction similar to that of $\mathcal{M}_{\mathcal{R}}$, using the poset of *ground* partial C-terms as carrier, we can obtain also *initial* models for programs. However, the free term model is more interesting for our purposes, since it characterizes CRWL-provability in the sense of Theorem 12.

## 5 A Lazy Narrowing Calculus

In this Section we set the basis for using CRWL as a declarative programming language. We define admissible goals and solutions for programs and we present a lazy narrowing calculus for goal-solving.

Let $\mathcal{R}$ be any program. Goals for $\mathcal{R}$ are certain finite conjunctions of CRWL-statements, and solutions are C-substitutions such that the goal affected by the substitution becomes CRWL-provable. The precise definition of *admissible goal* includes a number of technical conditions which are needed to achieve the effect of lazy evaluation with sharing during goal solving. In particular, sharing will be achieved by delaying the propagation of bindings for certain variables. Similar ideas have been used in the so-called outermost strategy for the functional logic language K-LEAF (based on flattening plus SLD-resolution) [5], in the constrained lazy narrowing calculus from [16] and in a call-by-need strategy for higher-order lazy narrowing [22]. In comparison to the present approach, [5, 16] allow for less general programs [2], while the language in [22] lacks a model-theoretic semantics and uses more restricted conditions of the form $l \rightarrow r$, where $r$ is a ground normal form.

**Definition 15.** (Admissible goals)
An *admissible goal* for a given program $\mathcal{R}$ must have the form $G \equiv \exists \overline{U}.\ S \square P \square E$, where:

- $\mathrm{evar}(G) \equiv \overline{U}$ is the set of so-called *existential variables* of the goal $G$.

[2] More precisely, this is true only for the sublanguage of [16] which omits the use of *disequality constraints*.

- $S \equiv X_1 = s_1, \ldots, X_n = s_n$ is a set of equations, called *solved part*. Each $s_i$ must be a total C-term, and each $X_i$ must occur exactly once in the whole goal. (Intuition: Each $s_i$ is a computed answer for $X_i$.)
- $P \equiv e_1 \rightarrow t_1, \ldots, e_k \rightarrow t_k$ is a multiset of *approximation conditions*, with $t_i \in \mathsf{CTerm}$. $\mathsf{pvar}(P) =_{def} \mathsf{var}(t_1) \cup \ldots \cup \mathsf{var}(t_k)$ is called the set of *produced variables* of the goal $G$. The *production relation* between $G$-variables is defined by $X \gg_P Y$ iff there is some $1 \leq i \leq k$ such that $X \in \mathsf{var}(e_i)$ and $Y \in \mathsf{var}(t_i)$. (Intuition: $e_i \rightarrow t_i$ demands narrowing $e_i$ to match $t_i$. This may produce bindings for variables in $t_i$.)
- $E \equiv a_1 \bowtie b_1, \ldots, a_m \bowtie b_m$ is a multiset of *joinability conditions*. $\mathsf{dvar}(E) =_{def} \{ X \in \mathcal{V} \mid X \equiv a_i$ or $X \equiv b_i$, for some $1 \leq i \leq m \}$ is called the set of *demanded variables* of the goal $G$. (Intuition: due to the semantics of joinability, goal solving must compute totally defined values for demanded variables.)

Additionally, any admissible goal must fulfil the following conditions:

- The tuple $(t_1, \ldots, t_k)$ must be *linear*. (Intuition: each produced variable is produced only once.)
- All the produced variables must be existential, i.e. $\mathsf{pvar}(P) \subseteq \mathsf{evar}(G)$. (Intuition: produced variables are used to compute intermediate results.)
- The transitive closure of the production relation $\gg_P$ must be irreflexive, or equivalently, a strict partial order. (Intuition: Bindings for produced variables are computed hierarchically.)
- The solved part contains no produced variables. (Intuition: the solved part includes no reference to intermediate results.) □

We assume by convention that in an *initial goal* $G$ only the joinability part $E$ is present, and there are no existential variables in $G$ [3].

**Definition 16.** (Solutions)

Let $G \equiv \exists \overline{U}.\ S \square P \square E$ be an admissible goal, and $\theta$ a partial C-substitution.

- $\theta$ is *allowable* for $G$ iff $X\theta$ is a total C-term for every $X \notin pvar(P)$.
- $\theta$ is a *solution* for $G$ iff $\theta$ is allowable for $G$, $X_i\theta \equiv s_i\theta$ for all $X_i = s_i \in S$, and $(P \square E)\theta$ has a "*witness*" $\mathcal{M}$. A witness is defined as a multiset containing a GPC-proof for each condition $e\theta \rightarrow t\theta \in P\theta$ and $a\theta \bowtie b\theta \in E\theta$ (see Def. 2).
- We write $Sol(G)$ for the set of all solutions for G. □

Our definition of solution must cover the case of intermediate goals of a computation. This explains why *partial* C-substitutions are considered, because produced variables (which are not present initially, are existential and can eventually disappear during the computation) may need to be given only partial values, since they serve to express approximations. But notice that with our condition on

[3] To accept any admissible goal as initial goal seems not very natural, but it would cause no major problem, except minor technical changes in some of the results below.

allowable substitutions, solutions of both initial and final goals (where only the solved part $S$ will be present) are *total* C-substitutions.

Due to the Adequateness Theorem 12, it is immediate to give a model-theoretic characterization of solutions, equivalent to the proof-theoretic definition. It is enough for our purposes to do this for initial goals.

**Lemma 17.** *Let $\mathcal{R}$ be a program, $G$ an initial goal, $\theta$ a C-substitution. The following statements are equivalent:*

*a)* $\theta \in Sol(G)$ *c)* $\mathcal{M}_{\mathcal{R}} \models (G\theta)\,\mathsf{id}$
*b)* $\mathcal{R} \vdash G\theta$ *d)* $\mathcal{A} \models (G\theta)\eta$, *for all* $\mathcal{A} \models \mathcal{R}, \eta \in \mathsf{DefVal}(\mathcal{A})$ ∎

We present now a *Constructor-based Lazy Narrowing Calculus* (CLNC) for solving initial goals, obtaining *solutions* in the sense of Definition 16. The CLNC-calculus consists of a set of *transformation rules* for goals. Each transformation rule takes the form $G \Vdash G'$, specifying one of the possible ways of performing one step of goal solving. Derivations are sequences of $\Vdash$-steps. For writing failure rules we use $FAIL$ representing an irreducible inconsistent goal. We recall that in a goal $\exists\overline{U}.\ S \square P \square E$, $S$ is a set while $P, E$ are multisets. Consequently, in the transformation rules no particular selection strategy (e.g. "sequential left-to-right") is assumed for the conditions in $S$,$P$ or $E$. In addition, to the purpose of applying the rules, we see conditions $a \bowtie b$ as symmetric. The notation $svar(e)$, used in some transformation rules, stands for the set of all variables $X$ occurring in $e$ at some position whose ancestor positions are all occupied by constructors.

## The CLNC calculus

### Rules for $\bowtie$

(DC1) **Decomposition:**
$\exists\overline{U}.\ S \square P \square c(\overline{a}) \bowtie c(\overline{b}), E \Vdash \exists\overline{U}.\ S \square P \square \ldots, a_i \bowtie b_i, \ldots, E.$

(ID) **Identity:**
$\exists\overline{U}.\ S \square P \square X \bowtie X, E \Vdash \exists\overline{U}.\ S \square P \square E$ if $X \notin pvar(P)$.

(BD) **Binding:**
$\exists\overline{U}.\ S \square P \square X \bowtie s, E \Vdash \exists\overline{U}.\ X = s, (S \square P \square E)\sigma$ if $s \in CTerm, X \notin s$, $X \notin var(s)$, $X \notin pvar(P), var(s) \cap pvar(P) = \emptyset$, where $\sigma = \{X/s\}$.

(IM) **Imitation:**
$\exists\overline{U}.\ S \square P \square X \bowtie c(\overline{e}), E \Vdash \exists\overline{X}, \overline{U}.\ X = c(\overline{X}), (S \square P \square \ldots, X_i \bowtie e_i, \ldots E)\sigma$
if $c(\overline{e}) \notin CTerm$ or $var(c(\overline{e})) \cap pvar(P) \neq \emptyset, X \notin pvar(P), X \notin svar(c(\overline{e}))$,
where $\sigma = \{X/c(\overline{X})\}$, $\overline{X}$ are new variables.

(NR1) **Narrowing:**
$\exists\overline{U}.\ S \square P \square f(\overline{e}) \bowtie a, E \Vdash \exists\overline{X}, \overline{U}.\ S \square \ldots, e_i \to t_i, \ldots P \square C, r \bowtie a, E$ where $R : f(\overline{t}) \to r \Leftarrow C$ is a variant of a rule in $\mathcal{R}$, with $\overline{X} = var(R)$ new variables.

## Rules for $\rightarrow$

(DC2) **Decomposition:**
$\exists \overline{U}.\ S \square c(\overline{e}) \rightarrow c(\overline{t}), P \square E \Vdash \exists \overline{U}.\ S \square \ldots, e_i \rightarrow t_i, \ldots, P \square E.$

(OB) **Output Binding:**

(OB1) $\exists \overline{U}.\ S \square X \rightarrow t, P \square E \Vdash \exists \overline{U}.\ X = t, (S \square P \square E)\sigma$
if $t \notin Var$, $X \notin pvar(P)$, where $\sigma = \{X/t\}$.

(OB2) $\exists X, \overline{U}.\ S \square X \rightarrow t, P \square E \Vdash \exists \overline{U}.\ (S \square P \square E)\sigma$
if $t \notin Var, X \in pvar(P)$, where $\sigma = \{X/t\}$.

(IB) **Input Binding:**
$\exists X, \overline{U}.\ S \square t \rightarrow X, P \square E \Vdash \exists \overline{U}.\ S \square (P \square E)\sigma$ if $t \in CTerm$, where $\sigma = \{X/t\}$.

(IIM) **Input Imitation:**
$\exists X, \overline{U}.\ S \square c(\overline{e}) \rightarrow X, P \square E \Vdash \exists \overline{X}, \overline{U}.\ S \square (\ldots, e_i \rightarrow X_i, \ldots, P \square E)\sigma$
if $c(\overline{e}) \notin CTerm, X \in dvar(E)$, where $\sigma = \{X/c(\overline{X})\}$, $\overline{X}$ new variables.

(EL) **Elimination:** $\exists X, \overline{U}.\ S \square e \rightarrow X, P \square E \Vdash \exists \overline{U}.\ S \square P \square E$ if $X \notin var(P \square E)$.

(NR2) **Narrowing:**
$\exists \overline{U}.\ S \square f(\overline{e}) \rightarrow t, P \square E \Vdash \exists \overline{X}, \overline{U}.\ S \square \ldots, e_i \rightarrow t_i, \ldots, r \rightarrow t, P \square C, E$ if $t \notin Var$ or $t \in dvar(E)$, where $R : f(\overline{t}) \rightarrow r \Leftarrow C$ is a variant of a rule in $\mathcal{R}$, with $\overline{X} = var(R)$ new variables.

## Failure rules

(CF1) **Conflict:** $\exists \overline{U}.\ S \square P \square c(\overline{a}) \bowtie d(\overline{b}), E \Vdash FAIL$ if $c \not\equiv d$.

(CY) **Cycle:** $\exists \overline{U}.\ S \square P \square X \bowtie a, E \Vdash FAIL$ if $X \not\equiv a, X \in svar(a)$.

(CF2) **Conflict:** $\exists \overline{U}.\ S \square c(\overline{a}) \rightarrow d(\overline{t}), P \square E \Vdash FAIL$ if $c \not\equiv d$.

The following remarks attempt to clarify some relevant aspects of the CLNC calculus.

- In all rules involving a substitution $\sigma$ (namely (BD), (IM), (OB), (IB), (IIM)), $\sigma$ replaces a variable by a C-term. This means, in particular, that for a condition $f(\overline{e}) \rightarrow X$, in no case the substitution $\{X/f(\overline{e})\}$ is applied[4]. Instead, the following possibilities are considered:

(i) The rule (EL) deletes the condition if $X$ does not appear elsewhere, because in this case any value (even $\perp$) is valid for the (existential) variable $X$ to satisfy the goal. As a consequence, the evaluation of $f(\overline{e})$ is not needed and is indeed not performed. Hence, the rule (EL) is crucial for respecting the non-strict semantics of functions.

(ii) The rule (NR2) uses a rule of the program for reducing $f(\overline{e})$, but only if $X$ is detected as a *demanded* variable, which in particular implies that $X$'s value in a solution cannot be $\perp$ [5], and therefore requires the evaluation of

[4] To perform the *eager* replacement $\{X/f(\overline{e})\}$ would be unsound due to the "call-time-choice nondeterminism of functions.

[5] In fact, the condition for a variable to be in the set *dvar* could be relaxed to weaker conditions still entailing $X \not\equiv \perp$.

$f(\overline{e})$. After one or more applications of (NR2) it will be the case (if the computation is going to succeed) that (IB) or (IIM) are applicable, thus propagating (partially, in the case of (IIM)) the obtained value to all the occurrences of $X$. As a result, *sharing* is achieved, and computations are lazy.

(iii) If neither (EL) nor (NR2) are applicable, nothing can be done with $f(\overline{e}) \to X$ but waiting until one of them becomes applicable. This will eventually happen, as our completeness results show.

- The absence of cycles of produced variables implies that no occur-check is needed in (OB), (IB), (IIM).
- The rules (BD), (OB), (IB) correspond to safe cases for eager variable elimination (via binding). Special care is taken with produced variables. For instance, the goal $\exists N.\ \Box X \to s(N) \Box X \bowtie N$ is admissible, but if (BD) could be applied (which is not allowed in CLNC, since $N$ is a produced variable) we would obtain $\exists N.\ X = N \Box N \to s(N) \Box$ , which is not admissible due to the presence of the produced variable $N$ in the solved part and, more remarkable, the creation of a cycle $N \gg_P N$, with the subsequent need of occur check to detect unsolvability of $N \to s(N)$.
- Narrowing rules include a *don't know* choice of the rule $R$ of the program $\mathcal{R}$ to be used. The rest of the rules are completely deterministic (modulo the symmetry of $\bowtie$) and, what is more important, if several $\Vdash$-rules are applicable to a given goal, a *don't care* choice among them can be done, as a consequence of the Progress Lemma 21 below. This kind of *strong completeness* doesn't hold in general for other lazy narrowing calculi, as shown in [21].

As an additional consequence of Lemma 21 below, a goal is $\Vdash$-irreducible iff it is $FAIL$ or takes the form $\exists \overline{U}.\ S \Box\ \Box$ (we call *solved forms* to these goals). It is easy to see that solved forms are satisfiable. Each solved form $\exists \overline{U}.\ S \Box\ \Box$ , with $S \equiv X_1 = t_1, \ldots, X_n = t_n$, defines an associated *answer substitution* $\sigma_S = \{X_1/t_1, \ldots, X_n/t_n\}$, which is idempotent. Notice that $\sigma_S \in Sol(\exists \overline{U}.\ S \Box\ \Box)$.

## 6 Soundness and Completeness

In this section we state the soundness and completeness of CLNC w.r.t. the declarative semantics of CRWL. Our first result proves correctness of a single CLNC step. It says that $\Vdash$-steps preserve admissibility of goals, fail only in case of unsatisfiable goals and do not introduce new solutions. In the latter case, some care must be taken with the possible elimination of existential variables.

**Lemma 18.** *(Correctness lemma)*

*(INV) If $G \Vdash G'$ and $G$ is admissible, then $G'$ is admissible.*
*(CR1) If $G \Vdash FAIL$ then $Sol(G) = \emptyset$*
*(CR2) If $G \Vdash G'$ and $\theta' \in Sol(G')$ then there exists $\theta \in Sol(G)$ with $\theta = \theta'[\backslash evar(G)]$*

■

It is easy now to obtain the following result, stating that computed answers for a goal $G$ are indeed solutions of $G$. We recall that, according to Lemma 17, we can give both proof-theoretic and model-theoretic readings to this result. The same remark holds for the Completeness Theorem 22 below.

**Theorem 19.** *(Soundness of CLNC)*
*If $G$ is an initial goal and $G \Vdash^+ \exists\overline{U}.\, S\square\ \square$, then $\sigma_S \in Sol(G)$.* ∎

We address now the question of completeness of CLNC. For reasoning about termination we introduce a well founded *ordering over multisets of proofs* which will allow to express how far is a goal from a solved form.

**Definition 20.** (Multiset ordering for proofs)
Let $\mathcal{R}$ be a program. If $\mathcal{M} \equiv \{\{\Pi_1, \ldots, \Pi_n\}\}$ and $\mathcal{M}' \equiv \{\{\Pi'_1, \ldots, \Pi'_m\}\}$ are multisets of GPC-proofs of approximation and joinability statements, we define

$$\mathcal{M} \lhd \mathcal{M}' \Leftrightarrow \{\{\mid \Pi_1 \mid, \ldots, \mid \Pi_n \mid\}\} \prec \{\{\mid \Pi'_1 \mid, \ldots, \mid \Pi'_m \mid\}\}$$

where $\mid \Pi \mid$ is the *length* (i.e., the number of inference steps) of $\Pi$, and $\prec$ is the *multiset extension* [3] of the usual ordering over $\mathbb{N}$. □

The well founded ordering $\lhd$ will be used for comparing witnesses of goal solutions. The overall idea for proving completeness is the following: for given goal $G$ and solution $\theta$, if $G$ is not in solved form, then any rule of CLNC applicable to $G$ (and there will be at least one by Lemma 21 (PR1) below) can be used for performing a $\Vdash$-step, while still capturing $\theta$ (Lemma 21 (PR2) below) and descending in the ordering $\lhd$ (again Lemma 21 (PR2) below). This is expessed by the following result, whose proof is technically involved[6].

**Lemma 21.** *(Progress Lemma)*

*(PR1) If $G \not\equiv FAIL$ is not a solved form, then there exists a CLNC-transformation applicable to $G$.*

*(PR2) If $\mathcal{M}$ is a witness of $\theta \in Sol(G)$, and $T$ is a CLNC-transformation applicable to $G$, then there exist $G', \theta'$ and $\mathcal{M}'$ such that*

*(i) $G \Vdash G'$ by means of the CLNC-transformation $T$*
*(ii) $\mathcal{M}'$ is a witness of $\theta' \in Sol(G')$*
*(iii) $\mathcal{M}' \lhd \mathcal{M}$*
*(iv) $\theta = \theta'[\backslash(evar(G) \cup nvar(G'))]$,*
*where $nvar(G') = var(G') \setminus var(G)$ ( $\subseteq evar(G')$)* ∎

We can now prove that any solution for a goal is subsumed by a computed answer, which constitutes the main result of this section.

**Theorem 22.** *(Completeness of CLNC)*
*Let $\mathcal{R}$ be a CRWL, $G$ an initial goal, $\theta \in Sol(G)$. Then there exists a solved form $\exists\overline{U}.\, S\square\ \square$ such that $G \Vdash^+ \exists\overline{U}.\, S\square\ \square$ and $\sigma_S \leq \theta[var(G)]$.* ∎

[6] Among other technicalities, the proof uses Lemma 10.

# 7 Conclusions and Final Example

We have achieved a logical presentation of a quite general approach to declarative programming, based on the notion of non-deterministic lazy function. Besides proof calculi and a proof theory for a constructor-based conditional rewriting logic CRWL, we have presented a sound and strongly complete lazy narrowing calculus CLNC, which is able to support sharing and to identify safe cases for eager replacement. All this shows the potential of our approach as a firm foundation for the development of functional logic languages. On the other hand, the practicability of the approach is guaranteed since existing implementations of functional logic languages can be used with only trivial changes (mainly relaxing the use of extra variables) for executing CWRL programs. As an example we show here a CWRL program which has been executed using the functional logic programming system BabLog[7] [2], based on implementation techniques borrowed from [14]. We present the program in BabLog's syntax [8]. The use of non-deterministic functions allows a very natural formulation of a grammar for simple arithmetic expressions. Any concrete token list *tokens* can be parsed by solving the goal *tokens* == exp. Note that == and := are BabLog's syntax for $\bowtie$ and $\rightarrow$, respectively.

*Example 3.*

```
data tok :=
    n int | + | - | * | / | ( | ).
fun exp : list tok.
    exp := term ++ [alt + - | exp].
    exp := term.
fun term : list tok.
    term := factor ++ [ alt * / | term].
    term := factor.
fun factor : list tok.
    factor := [ ( | exp ] ++ [ ) ].
    factor := [ n N ].
fun alt : A → A → A.
    alt X Y := X.
    alt X Y := Y.
fun ++ : list A → list A → list A.
    [ ] ++ L := L.
    [X|Xs] ++ L := [X | Xs ++ L].
```

Future work will include more experimentation with implementations and extension of the whole approach to cover typed higher order programming via applicative CTRSs, using ideas and techniques from [7, 9].

**Acknowledgement:** We would like to thank Ana Gil-Luezas and Puri Arenas-Sánchez for useful comments and contributions to implementation work.

## References

1. S. Antoy, R. Echahed, and M. Hanus. *A Needed Narrowing Strategy.* Proc. 21st ACM Symp. on Princ. of Prog. Lang., Portland, pp. 268-279, 1994.
2. P. Arenas-Sánchez and A. Gil-Luezas. *BabLog User's Manual.* Tech. Rep. DIA 94/12, 1994.

[7] Available in `ftp://bach.mat.ucm.es/pub/langs/HOBABEL/bablog.tar.gz`

[8] With some liberalities like the use of $+, -, *, \ldots$ as constant constructor symbols, and the use of infix notation for the list concatenation operation ++.

3. N. Dershowitz and Z. Manna: *Proving Termination with Multiset Orderings.* Comm. of the ACM 22(8), 1979, 465-476.
4. M. Falaschi, G. Levi, M. Martelli and C. Palamidessi. *A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs.* Information and Computation, 102(1), pp. 86-113, 1993.
5. E. Giovannetti, G. Levi, C. Moiso and C. Palamidessi. *Kernel-LEAF: A Logic plus Functional Language.* JCSS 42 (2), pp. 139-185, 1991.
6. J. A. Goguen, J. W. Thatcher, E. G. Wagner and J. B. Wright. *Initial Algebra Semantics and Continuous Algebras.* Journal of the ACM 24(1), pp. 68-95, 1977.
7. J. C. González-Moreno, M. T. Hortalá-González and M. Rodríguez-Artalejo. *On the completeness of Narrowing as the Operational Semantics of Functional Logic Programming.* Proc. CSL'92, Springer LNCS 702, pp. 216-230, 1993.
8. J. C. González-Moreno, M. T. Hortalá-González , F.J. López-Fraguas and M. Rodríguez-Artalejo. *A Rewriting Logic for Declarative Programming*, Tech. Rep. DIA 95/10, 1995.
9. J. C. González-Moreno. *A Correctness Proof for Warren' s HO into FO Translation.* Proc. GULP'93, Gizzeria (CZ), Italy, 1993.
10. M. Hanus. *The Integration of Functions into Logic Programming: A Survey.* JLP (19&20). Special issue "*Ten Years of Logic Programming*", pp. 583-628, 1994.
11. M. Hanus. *Lazy Unification with Simplification.* Proc. ESOP'94, Springer LNCS 778, pp. 272-286, 1994.
12. M. Hanus. *Combining Lazy Narrowing and Simplification.* Proc. PLILP'94, Springer LNCS 844, pp. 370-384, 1994.
13. H. Hussmann. *Non-determinism in Algebraic Specifications and Algebraic Programs.* Birkhäuser Verlag, 1993.
14. R. Loogen, F.J.López-Fraguas and M.Rodríguez-Artalejo. *A Demand Driven Computation Strategy for Lazy Narrowing.* Proc. PLILP'93, Springer LNCS 714, pp. 184-200, 1993.
15. R. Loogen and S. Winkler. *Dynamic detection of determinism in functional logic languages.* TCS 142, pp. 59-87, 1995.
16. F. J. López-Fraguas. *Programación Funcional y Lógica con Restricciones.* PhD Thesis, Univ. Complutense Madrid, 1994. (In Spanish)
17. J. Meseguer. *Conditional rewriting logic as a unified model of concurrency.* TCS 96, pp. 73-155, 1992.
18. B. Möller. *On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types.* Acta Informatica 22, pp. 537-578, 1985.
19. A. Middeldorp and E. Hamoen. *Completeness Results for Basic Narrowing.* Applicable Algebra in Engineering, Comm. and Comp. 5, pp. 213-253, 1994.
20. J. J. Moreno-Navarro and M. Rodríguez-Artalejo. *Logic Programming with Functions and Predicates: The Language BABEL.* JLP 12, pp. 191-223, 1992.
21. S. Okui, A. Middeldorp and T. Ida. *Lazy Narrowing: Strong Completeness and Eager Variable Elimination.* Proc. CAAP'95, Springer LNCS 915, pp. 394-408, 1995.
22. C. Prehofer. A Call-by-Need Strategy for Higher-Order Functional Logic Programming. Proc. ILPS'95, MIT Press, pp. 147-161, 1995.
23. A. Sarmiento-Escalona. *Una aproximación a la Programación Lógica con Funciones Indeterministas*, PhD Thesis, Univ. La Coruña, 1992. (In Spanish)