# Functional plus Logic Programming with Built-in and Symbolic Constraints*

*P. Arenas-Sánchez, F.J. López-Fraguas, M. Rodríguez-Artalejo*
Dpto. Sistemas Informáticos y Programación, UCM, Madrid
email: {puri,fraguas,mario}@sip.ucm.es

**Abstract.** In this paper we propose a lazy functional logic language, named SETA, which allows to handle *multisets*, *built-in arithmetic constraints* over the domain of real numbers, as well as various *symbolic constraints* over datatypes. As main theoretical results, we have proved the existence of free term models for all SETA programs and we have developed a correct and complete goal solving mechanism.

## 1 Introduction

Functional logic programming (FLP, in short) aims to combine into a single paradigm the nicest properties of both functional and logic programming. In many approaches to FLP (see [9] as a survey) programs are seen as constructor-based conditional rewrite systems. One of such approaches is the CRWL-framework of [7] where classical equational logic is replaced by a suitable constructor-based rewriting logic which expresses properly the behaviour of reduction for lazy, partial and possibly non-deterministic functions.

Most approaches to FLP, including CRWL, are based on *free* constructors. However, for some applications it is more convenient to represent data by means of non-free constructors, for which some equational specification is given. An extension of [7] along this line has been presented in [4, 5], where a general framework for FLP with polymorphic algebraic types is investigated. One particular interesting case is that of *multisets*, which are known to be useful to model a variety of scenarios, like the Gamma programming model [6] or action and change problems [15]. On the other hand, many problems involve computations over specific domains –like real numbers, boolean functions or finite domains– for which the constructor-based approach is not adequate at all. A crucial contribution to this issue within the field of logic programming has been the *constraint logic programming* paradigm [11] and its different instances, such as the language $CLP(\mathcal{R})$ [12] which is known to have a wide and growing range of applications.

Our aim in this work is to merge the expressive power of polymorphic algebraic types and constraints into a single language (named SETA[1]) which can be understood as an *extended instance* of the framework in [4, 5]. It is an instance in the sense that the *multiset constructor* is the unique non-free constructor we consider, but it is extended since it incorporates primitive "built-in" data (the real numbers), constraints over them and also "symbolic" constraints over

* This research has been partially supported by the Spanish National Project TIC98-0445-C03-02 "TREND" and the Esprit BRA Working Group EP-22457 "CCLII".

[1] SETA is not an acronym, but simply the spanish word for *mushroom*.

constructor terms (in particular, over multisets), including *equality*, *disequality*, *membership* and *non-membership*. Following a line similar to that of [7, 4, 5], we develop proof-theoretic and model-theoretic semantics (including the existence of free term models) of SETA programs, and then propose an operational semantics by means of a sound and complete goal solving calculus which combines lazy narrowing, unification modulo the equational axiom of multisets and constraint solving. The presence of primitive data and constraints makes things substantially different and technically more involved than in [7, 5].

The range of potential applications of SETA is very large. To those which make use in isolation of either multisets or real constraints, we must add others which can take profit of the combination of both. One of such application fields is the parsing of visual languages [10, 14], where in particular one must recognize figures in a graphic description consisting of an unordered collection (a multiset!) of items (points, lines, ...) whose positions and spatial relationships can be expressed by means of real numbers and constraints over them.

The rest of the paper is organized as follows. Next section introduces the language SETA in a multilevel description, includes an small example illustrating the capabilities of SETA and develops a proof theory in the form of a constrained goal-oriented constructor-based rewriting calculus. Section 3 presents the model theory for SETA programs whereas Section 4 contains a sketch of the goal solving calculus for SETA and the corresponding soundness and completeness results. Last section summarizes some conclusions. Due to space limitations, proofs and many other technical details have been left out, but they can be found in [2].

## 2 The Language SETA

Due to the fact that language SETA handles real numbers, the presentation of the language is based on two levels: The *primitive level* (containing everything related to real numbers) and the *symbolic level* (containing data constructors including the multiset constructor and constraints over them).

### 2.1 The Primitive Level

$\Sigma_p$ denotes the *primitive signature* defined as the triple $\langle PT, PO, PP\rangle$, where $PT$ has *Real* as unique *Primitive Type*, $PO = \{0, 1 :\rightarrow Real, +, * : (Real, Real) \rightarrow Real\}$ is a set of type declarations for *Primitive Operations*, and $PP = \{==, /\!\!=, \leq : (Real, Real)\}$ is a set of type declarations for *Primitive Predicates*.

Given a denumerable set $x, y, \ldots \in DV$ of data variables, the set $T_p(DV)$ of *primitive terms* $t^p, s^p, \ldots$ is built from $DV$ and $PO$. The set $R_p(DV)$ of *primitive constraints* $\varphi^p$ is defined as $\varphi^p ::= True \mid t_1^p \Diamond t_2^p \mid \varphi_1^p \wedge \varphi_2^p \mid \exists x \varphi^p$, where[2] $x \in DV$, $\Diamond \in \{==, /\!\!=, \leq\}$, $t_i^p \in T_p(DV)$, $\varphi_i^p \in R_p(DV)$, $1 \leq i \leq 2$.

In the following, we will use $\mathcal{R}$ to denote the field of real numbers $(\mathbb{R}, 0^{\mathbb{R}}, 1^{\mathbb{R}}, +^{\mathbb{R}}, *^{\mathbb{R}})$. $\eta^p$ will denote a primitive valuation (i.e., a mapping from $DV$ to $\mathbb{R}$), and $(\mathcal{R}, \eta^p) \models_{\mathcal{R}} \varphi^p$ will express the validity of the primitive constraint $\varphi^p \in R_p(DV)$ in $\mathcal{R}$ under $\eta^p$. Finally, given a set $\Gamma_p$ of primitive constraints and a constraint $\varphi^p \in R_p(DV)$, the notation $\Gamma_p \models_{\mathcal{R}} \varphi^p$ will mean that $\varphi^p$ is a logical consequence of $\Gamma_p$ when both are interpreted over $\mathcal{R}$.

[2] Note that the constraints $\geq$, $<$ and $>$ may be easily defined from $/\!\!=$ and $\leq$.

### 2.2 The Symbolic Level

Let $TV$ be a countable set of *type variables* $\alpha, \beta, \ldots$, and $TC = \bigcup_{n \geq 0} TC^n$ a countable alphabet of *type constructors* $K, K', \ldots$ including the multiset type constructor $Mset \in TC^1$. *Polymorphic types* $\tau, \tau', \ldots \in Type_{TC}(TV)$ are built from $TV$ and $TC$. The set of type variables occurring in $\tau$ is written $tvar(\tau)$.

We define a *polymorphic signature* $\Sigma$ over $TC$ as $\Sigma = \Sigma_p \cup \langle TC, DC, FS, PS \rangle$, where:

- $DC$ is a set of type declarations for *data constructors*, of the form $c : (\tau_1, \ldots, \tau_n) \rightarrow \tau$ with $\bigcup_{i=1}^{n} tvar(\tau_i) \subseteq tvar(\tau)$ and $\tau \notin TV \cup PT$. We assume that $DC$ contains the type declarations $\{\!\!\{\ \}\!\!\} :\rightarrow Mset(\alpha)$ (representing the *empty multiset*) and $\{\!\!\{\cdot|\cdot\}\!\!\} : (\alpha, Mset(\alpha)) \rightarrow Mset(\alpha)$ (representing the *multiset constructor*[3]). The multiset constructor is governed by the equation (MSET) : $\{\!\!\{x, y|xs\}\!\!\} \approx \{\!\!\{y, x|xs\}\!\!\}$. Here, we have used $\{\!\!\{x, y|xs\}\!\!\}$ as abbreviation for $\{\!\!\{x|\{\!\!\{y|xs\}\!\!\}\}\!\!\}$. In the sequel we will continue using such notation.
- $FS$ is a set of type declarations for *defined function symbols*, of the form $f : (\tau_1, \ldots, \tau_n) \rightarrow \tau$.
- $PS = \{==, /=: (\alpha, \alpha), \in, \notin: (\alpha, Mset(\alpha))\}$ is a set of type declarations for *predicate symbols*. $==$ and $/=$ stand for *strict equality* and *disequality* respectively, whereas $\in$, $\notin$ represent *membership* and *non-membership* respectively.

We require that $DC \cup FS$ does not include multiple type declarations for the same symbol. We will write $h \in DC^n \cup FS^n$ to indicate the arity of a symbol according to its type declaration. In the following, $DC_\perp$ will denote $DC$ extended by a new declaration $\perp :\rightarrow \alpha$. The bottom constant constructor $\perp$ is intended to represent an undefined value. Analogously, $\Sigma_\perp$ will denote the result of replacing $DC$ by $DC_\perp$ in $\Sigma$.

*Total expressions* $e, es, r, \ldots \in E_\Sigma(DV)$ are built from $DV$, $PO$, $DC$ and $FS$. The set $E_{\Sigma_\perp}(DV)$ of *partial expressions* is defined in the same way, but using $DC_\perp$ in place of $DC$. *Total data terms* $T_\Sigma(DV) \subseteq E_\Sigma(DV)$ and *partial data terms* $T_{\Sigma_\perp}(DV) \subseteq E_{\Sigma_\perp}(DV)$ are built by using only variables, primitive operations and data constructors only. In the sequel, we reserve $t, ts, s, l, ls, m, ms$, to denote possibly partial data terms, and we write $dvar(e)$ for the set of all data variables occurring in an expression $e$.

The set $\varphi, \varphi', \ldots \in R_{\Sigma_\perp}(DV)$ of *partial constraints* is defined as $\varphi ::= True \mid e_1 \diamondsuit e_2 \mid \varphi_1 \wedge \varphi_2 \mid \exists x \varphi$, where $\diamondsuit \in \{==, /=, \leq, \in, \notin\}$, $x \in DV$, $e_i \in E_{\Sigma_\perp}(DV)$, $\varphi_i, \varphi \in R_{\Sigma_\perp}(DV)$, $1 \leq i \leq 2$. We say that $\varphi \in R_{\Sigma_\perp}(DV)$ is a *total constraint* if $\perp$ does not occur in $\varphi$. The set of all *total constraints* is denoted by $R_\Sigma(DV)$.

The rewriting calculus in Subsection 2.3 will deal with an *extended* notion of expressions and constraints, for which all real numbers are available as new constants. Formally, the set $E^*_{\Sigma_\perp}(DV)$ of *extended partial expressions* is defined as $e ::= \perp \mid x \mid 0 \mid 1 \mid d \mid e_1 \diamondsuit e_2 \mid h(e_1, \ldots, e_n)$, where $h \in DC^n \cup FS^n$, $e_i \in E^*_{\Sigma_\perp}(DV)$, $1 \leq i \leq n$, $d \in \mathbb{R}$, $\diamondsuit \in \{+, *\}$, $x \in DV$. If we eliminate $\perp$ in the definition above, we obtain the set $E^*_\Sigma(DV)$ of *extended total expressions*. Analogously, by ignoring $FS$ we can define the sets $T^*_{\Sigma_\perp}(DV)$ and $T^*_\Sigma(DV)$ of *extended*

[3] The intended meaning of $\{\!\!\{x|xs\}\!\!\}$ is to add a new copy of the element $x$ to the multiset $xs$.

*partial* and *total data terms*, respectively. The sets $R^*_{\Sigma_\perp}(DV)$ and $R^*_{\Sigma}(DV)$ of *extended partial constraints* and *extended total constraints* are defined similarly to the sets $R_{\Sigma_\perp}(DV)$ and $R_{\Sigma}(DV)$ respectively, but now all considered expressions must be extended. An *extended primitive term* $T^*_p(DV)$ will be an extended total data term not containing symbols in $DC$. Similarly, the set $R^*_p(DV) \subseteq R^*_{\Sigma}(DV)$ of *extended primitive constraints* is composed of all those extended total constraints containing only primitive symbols and variables.

We define possibly *partial data substitutions* $\delta$ as mappings from $DV$ to $T^*_{\Sigma_\perp}(DV)$. In the rest of the paper, $DSub(A)$, where $A$ is a subset of $T^*_{\Sigma_\perp}(DV)$, will denote the set of all data substitutions mapping $DV$ to $A$.

An *environment* is defined as any set $V$ of type-annotated data variables $x : \tau$, such that $V$ does not include two different annotations for the same variable. By considering that any $d \in \mathbb{R}$ has an associated type declaration $d :\rightarrow Real$, it is possible to determine when an extended partial expression $e \in E^*_{\Sigma_\perp}(DV)$ has type $\tau$ in an environment $V$. The set $E^{*^\tau}_{\Sigma_\perp}(V)$ (resp. $E^{*^\tau}_{\Sigma}(V)$) of all extended partial expressions (resp. extended total expressions) that admit type $\tau$ w.r.t. $V$ is defined in the usual way; see e.g. [4]. Note that $E^{*^\tau}_{\Sigma_\perp}(V)$ has $T^{*^\tau}_{\Sigma_\perp}(V)$ and $T^{*^\tau}_{\Sigma}(V)$ as subsets. We can talk also about the sets $E^{\tau}_{\Sigma_\perp}(V)$ and $T^{\tau}_{\Sigma_\perp}(V)$ (resp. $E^{\tau}_{\Sigma}(V)$ and $T^{\tau}_{\Sigma}(V)$) of partial expressions and terms (resp. total expressions and terms) which admit type $\tau$ in $V$.

A constraint $\varphi \in R^*_{\Sigma_\perp}(DV)$ is well-typed w.r.t. an environment $V$ iff one of the following items hold:

- $\varphi \equiv True$ or $\varphi \equiv e_1 \leq e_2$ and $e_i \in E^{*^{Real}}_{\Sigma_\perp}(V)$, $1 \leq i \leq 2$.
- $\varphi \equiv e_1 \diamondsuit e_2$, $\diamondsuit \in \{==, /=\}$ and $e_i \in E^{*^\tau}_{\Sigma_\perp}(V)$, $1 \leq i \leq 2$, for some $\tau \in Type_{TC}(TV)$. Or $\varphi \equiv e_1 \diamondsuit e_2$, $\diamondsuit \in \{\in, \notin\}$ and $e_1 \in E^{*^\tau}_{\Sigma_\perp}(V)$, $e_2 \in E^{*^{Mset(\tau)}}_{\Sigma_\perp}(V)$, for some $\tau \in Type_{TC}(TV)$.
- $\varphi \equiv \varphi_1 \wedge \varphi_2$ and $\varphi_i$ are well-typed w.r.t. $V$, $1 \leq i \leq 2$.
- $\varphi \equiv \exists x \varphi'$ and there exists $\tau \in Type_{TC}(TV)$ such that $\varphi'[x/y]$ is well-typed w.r.t. $V[y : \tau]$, where $y$ is a fresh variable and $V[y : \tau]$ denotes the environment resulting of adding to $V$ the type-annotation $y : \tau$.

Assuming a type declaration $f : (\tau_1, \ldots, \tau_n) \rightarrow \tau \in FS$, a *defining rule* for $f$ has the form $f(t_1, \ldots, t_n) \rightarrow r \Leftarrow \varphi$, where the left-hand side is linear, $t_i \in T_\Sigma(DV)$ does not contain any primitive symbol in $PO$, $1 \leq i \leq n$, $r \in E_\Sigma(DV)$, $\varphi \in R_\Sigma(DV)$ and $dvar(r) \subseteq \bigcup_{i=1}^n dvar(t_i)$. A program rule is *well-typed* iff there exists an environment $V$ such that $t_i \in T^{\tau_i}_{\Sigma}(V)$, $1 \leq i \leq n$, $r \in E^{\tau}_{\Sigma}(V)$ and $\varphi$ is well-typed w.r.t. $V$.

We define *programs* as pairs $\mathcal{P} = \langle \Sigma, \mathrm{R} \rangle$, where $\Sigma$ is a polymorphic signature and R is a finite set of defining rules for defined functions symbols in $\Sigma$. We say that a program $\mathcal{P}$ is *well-typed* iff all program rules in R are well-typed. Note that primitive operations are not allowed in the left-hand sides of program rules. This implies no loss of expressivity, due the availability of equality constraints.

As argued in [4, 5, 3], the expressive power of the multiset constructor can be used to tackle any kind of problem which is related to the widely applicable idea of *multiset rewriting*; see e.g. [6, 15, 14]. We present here a small example

which shows the advantages of combining multisets, real numbers and constraints to generate and recognize graphic figures, which is very related to the issue of parsing visual languages [10, 14].

*Example 1.* Consider the problem of building quadrilaterals from given points in the plane, in such a way that the resulting figures have no common vertices. Points and quadrilaterals can be respectively represented by means of the data constructors:

$$P : (Real, Real) \to Point, \quad Q : (Point, Point, Point, Point, Point) \to Quadrilateral.$$

The intended meaning of two consecutive points $P$, $P'$ in a term of the form $Q(P1, P2, P3, P4, P1)$ is that there exists a line from $P$ to $P'$. Figures will be multisets of quadrilaterals. In order to solve the problem, we define the functions:

$$\begin{array}{l}
\mathsf{figure} : Mset(Point) \to Mset(Quadrilateral) \\
\quad figure(\{\!\!\{\ \}\!\!\}) \to \{\!\!\{\ \}\!\!\} \\
\quad figure(\{\!\!\{ p1, p2, p3, p4 \,|\, ps \}\!\!\}) \to \{\!\!\{ Q(p1, p2, p3, p4, p1) \,|\, figure(ps) \}\!\!\} \\
\quad\quad \Leftarrow p1 \notin ps \wedge p2 \notin ps \wedge p3 \notin ps \wedge p4 \notin ps \wedge quadrilateral(p1, p2, p3, p4) == True
\end{array}$$

where the function quadrilateral checks if four points generate a quadrilateral by using the following result: "The four midpoints of the lines composing a quadrilateral form a parallelogram". The code for this function is the following:

$$\begin{array}{l}
\mathsf{quadrilateral} : (Point, Point, Point, Point) \to Bool \\
\quad quadrilateral(p1, p2, p3, p4) \to True \Leftarrow \\
\qquad midpoint(p1, p2, m1) == True \ \wedge \ midpoint(p2, p3, m2) == True \wedge \\
\qquad midpoint(p3, p4, m3) == True \ \wedge \ midpoint(p4, p1, m4) == True \wedge \\
\qquad parallelogram(m1, m2, m3, m4) == True
\end{array}$$

Of course, *True* is a boolean constant. Functions midpoint and parallelogram are defined as:

$$\begin{array}{l}
\mathsf{midpoint} : (Point, Point, Point) \to Bool \\
\quad midpoint(P(x1, y1), P(x2, y2), P(x3, y3)) \to True \\
\qquad \Leftarrow 2 * x3 == x1 + x2 \wedge 2 * y3 == y1 + y2
\end{array}$$

$$\begin{array}{l}
\mathsf{parallelogram} : (Point, Point, Point, Point) \to Bool \\
\quad parallelogram(P(x1, y1), P(x2, y2), P(x3, y3), P(x4, y4)) \to True \\
\qquad \Leftarrow x1 - x4 == x2 - x3 \wedge y1 - y4 == y2 - y3
\end{array}$$

Considering the goal:

$$\begin{array}{l}
G \equiv figure(\{\!\!\{ P(-3, 0), P(3, 0), P(4, 3), P(5, -4), \\
\qquad P(8, 0), P(12, 3), P(14, -2), P(11, -6) \}\!\!\}) == l
\end{array}$$

and using the lazy narrowing calculus presented in Sect. 4, we can obtain various computed answers, as e.g.

$$\begin{array}{l}
l = \{\!\!\{ Q(P(-3, 0), P(4, 3), P(3, 0), P(5, -4), P(-3, 0)), \\
\quad Q(P(8, 0), P(12, 3), P(14, -2), P(11, -6), P(8, 0)) \}\!\!\}
\end{array}$$

### 2.3 A Constrained Goal-Oriented Rewriting Calculus

In the rest of this section we present a *Constrained Goal-Oriented constructor-based Rewriting Calculus* (named *CGORC*) which is intended as a proof theoretical specification of program's semantics.

Given a finite set $\Gamma$ of constraints in *solved form*, the rewriting calculus *CGORC* allows to derive statements of the form $e \to t$ (named *approximation statements*) and constraints $\varphi$, where $e \in E^*_{\Sigma_\perp}(DV)$, $t \in T^*_{\Sigma_\perp}(DV)$, $\varphi \in R^*_{\Sigma_\perp}(DV)$ and $\Gamma \subseteq R^*_{\Sigma_\perp}(DV)$.

A constraint $\varphi \in R^*_{\Sigma_\perp}(DV)$ is in *solved form* iff $\varphi \in R^*_p(DV)$ or $\varphi \equiv x/\!\!= t$ or $\varphi \equiv s \notin xs$, where $x, xs \in DV$, $t, s \in T^*_{\Sigma_\perp}(DV)$, $x \not\equiv t$.

The intended meaning of an approximation statement $e \to t$ is that the possibly partial data term $t$ approximates $e$'s value. As notation, $\Gamma \vdash_{\mathcal{P}} \chi$, where $\chi$ is either an approximation statement or a constraint, and $\Gamma \subseteq R^*_{\Sigma_\perp}(DV)$ is in solved form, will denote the derivability of $\chi$ from $\Gamma$ in *CGORC* .

The constraint symbols $==$ and $/\!\!=$ are overloaded and they must be treated differently depending on the level they belong. For this reason, we need the notion of *the primitive part* of a finite set $\Gamma \subseteq R^*_{\Sigma_\perp}(DV)$ in solved form. Intuitively, this is the part $PP(\Gamma)$ of $\Gamma$ depending on those variables forced (by $\Gamma$) to take primitive (i.e., numeric) values. Formally, we define the set $primvar(\Gamma)$ of *primitive variables* in $\Gamma$ as the least set of variables verifying:

- ▷ If $\varphi^p \in R^*_p(DV) \cap \Gamma$ and $\varphi^p \neq x/\!\!= y$, then $lib(\varphi^p) \subseteq primvar(\Gamma)$, where $lib(\varphi^p)$ denotes the set of free variables of $\varphi^p$.
- ▷ If $x/\!\!= y \in \Gamma$ or $y/\!\!= x \in \Gamma$ and $x \in primvar(\Gamma)$ then $y \in primvar(\Gamma)$.

and we define the *primitive part* of $\Gamma$ as the set

$$PP(\Gamma) = \{\varphi^p \mid \varphi^p \in R^*_p(DV) \cap \Gamma, \varphi^p \not\equiv x/\!\!= y, x, y \in DV\} \cup \{x/\!\!= y \mid x/\!\!= y \in \Gamma, x, y \in primvar(\Gamma)\}.$$

For instance, if $\Gamma = \{x \leq 2, x/\!\!= y, y/\!\!= z, y \notin xs\}$ then $primvar(\Gamma) = \{x, y, z\}$, and $PP(\Gamma) = \{x \leq 2, x/\!\!= y, y/\!\!= z\}$.

**Rules of the *CGORC* Calculus:** Now we are ready to define the rewriting calculus *CGORC*. In the following, the notation $h(\bar{e}_n)$ is a shorthand for $h(e_1, \ldots, e_n)$, where $h \in DC^n \cup FS^n$. The calculus *CGORC* is composed of the following inference rules:

• ***CGORC*-rules for $\to$**

$$(\mathsf{PR})^1_{\to}\ \frac{PP(\Gamma) \models_{\mathcal{R}} t^p == s^p}{\Gamma \vdash_{\mathcal{P}} t^p \to s^p}\quad (\mathsf{PR})^2_{\to}\ \frac{\Gamma \vdash_{\mathcal{P}} e_1 \to t^p_1, \Gamma \vdash_{\mathcal{P}} e_2 \to t^p_2, \Gamma \vdash_{\mathcal{P}} t^p_1 \Diamond t^p_2 \to t^p_3}{\Gamma \vdash_{\mathcal{P}} e_1 \Diamond e_2 \to t^p_3}$$

$dvar(t^p \to s^p) \subseteq primvar(\Gamma)$, $\Diamond \in \{+, *\}$, $e_1 \Diamond e_2 \in E^*_{\Sigma_\perp}(DV) - T^*_{\Sigma_\perp}(DV)$.

$$(\mathsf{B})\ \frac{}{\Gamma \vdash_{\mathcal{P}} e \to \perp}\quad (\mathsf{RR})\ \frac{}{\Gamma \vdash_{\mathcal{P}} x \to x}\quad (\mathsf{DC})\ \frac{\Gamma \vdash_{\mathcal{P}} e_1 \to t_1, \ldots, \Gamma \vdash_{\mathcal{P}} e_n \to t_n}{\Gamma \vdash_{\mathcal{P}} c(\bar{e}_n) \to c(\bar{t}_n)}$$

$x \in DV, x \notin primvar(\Gamma)$ and $c \in DC^n$, $n \geq 0$.

$$(\mathsf{OMUT})\ \frac{\Gamma \vdash_{\mathcal{P}} e \to t_1, \Gamma \vdash_{\mathcal{P}} es \to \{\!|\, t_2 | ts \,|\!\}, \Gamma \vdash_{\mathcal{P}} \{\!|\, t_2, t_1 | ts \,|\!\} \to ts'}{\Gamma \vdash_{\mathcal{P}} \{\!|\, e | es \,|\!\} \to ts'} \quad ts' \not\equiv \perp$$

$$(\mathsf{OR})\ \frac{\Gamma \vdash_{\mathcal{P}} e_1 \to s_1\delta, \ldots, \Gamma \vdash_{\mathcal{P}} e_n \to s_n\delta, \Gamma \vdash_{\mathcal{P}} r\delta \to t, \Gamma \vdash_{\mathcal{P}} \varphi\delta}{\Gamma \vdash_{\mathcal{P}} f(\bar{e}_n) \to t}$$

$t \not\equiv \perp$, $f(\bar{s}_n) \to r \Leftarrow \varphi \in \mathrm{R}$ and $\delta \in DSub(T^*_{\Sigma_\perp}(DV))$.

• *CGORC*-**rules for** $R^*_{\Sigma_\perp}(DV)$

$$(\mathsf{PR})_\varphi \ \frac{PP(\Gamma) \models_{\mathcal{R}} \varphi^p}{\Gamma \vdash_{\mathcal{P}} \varphi^p} \qquad \varphi^p \in R^*_p(DV),\ lib(\varphi^p) \subseteq primvar(\Gamma)$$

If rule $(\mathsf{PR})_\varphi$ is not applicable, then we can use the following rules, in which the superscript $S$ means that a symmetric rule is implicitly assumed.

$$(\mathsf{HIP}) \ \frac{}{\Gamma \vdash_{\mathcal{P}} \varphi} \ \varphi \in \Gamma \qquad (\mathsf{CONJ}) \ \frac{\Gamma \vdash_{\mathcal{P}} \varphi_1, \Gamma \vdash_{\mathcal{P}} \varphi_2}{\Gamma \vdash_{\mathcal{P}} \varphi_1 \wedge \varphi_2} \qquad (\mathsf{EX}) \ \frac{\Gamma \vdash_{\mathcal{P}} \varphi'[x/t]}{\Gamma \vdash_{\mathcal{P}} \exists x \varphi'}$$

$$(\mathsf{C}) \ \frac{\Gamma \vdash_{\mathcal{P}} e \to t, \Gamma \vdash_{\mathcal{P}} r \to t}{\Gamma \vdash_{\mathcal{P}} e == r} \qquad t \in T^*_\Sigma(DV)$$

$$(\mathsf{LEQ}) \ \frac{\Gamma \vdash_{\mathcal{P}} e \to t^p, \Gamma \vdash_{\mathcal{P}} r \to s^p, \Gamma \vdash_{\mathcal{P}} t^p \leq s^p}{\Gamma \vdash_{\mathcal{P}} e \leq r}$$

$$(\mathsf{MEMB})_\to \ \frac{\Gamma \vdash_{\mathcal{P}} es \to \{\!|t|ts|\!\}, \Gamma \vdash_{\mathcal{P}} e \in \{\!|t|ts|\!\}}{\Gamma \vdash_{\mathcal{P}} e \in es} \qquad es \in E^*_{\Sigma_\perp}(DV) - T^*_{\Sigma_\perp}(DV)$$

$$(\mathsf{NMEMB})_\to \ \frac{\Gamma \vdash_{\mathcal{P}} es \to ts, \Gamma \vdash_{\mathcal{P}} e \notin ts}{\Gamma \vdash_{\mathcal{P}} e \notin es} \qquad es \in E^*_{\Sigma_\perp}(DV) - T^*_{\Sigma_\perp}(DV)$$

$$(\mathsf{MEMB})_1 \ \frac{\Gamma \vdash_{\mathcal{P}} e == t}{\Gamma \vdash_{\mathcal{P}} e \in \{\!|t|ts|\!\}} \qquad (\mathsf{NMEMB})_1 \ \frac{}{\Gamma \vdash_{\mathcal{P}} e \notin \{\!|\ |\!\}}$$

$$(\mathsf{MEMB})_2 \ \frac{\Gamma \vdash_{\mathcal{P}} e \in ts}{\Gamma \vdash_{\mathcal{P}} e \in \{\!|t|ts|\!\}} \qquad (\mathsf{NMEMB})_2 \ \frac{\Gamma \vdash_{\mathcal{P}} e \not\models t, \Gamma \vdash_{\mathcal{P}} e \notin ts}{\Gamma \vdash_{\mathcal{P}} e \notin \{\!|t|ts|\!\}}$$

$$(\mathsf{NEQ})_\to \ \frac{\Gamma \vdash_{\mathcal{P}} e \to t, \Gamma \vdash_{\mathcal{P}} r \to s, \Gamma \vdash_{\mathcal{P}} t \not\models s}{\Gamma \vdash_{\mathcal{P}} e \not\models r}$$

If $e \not\equiv \{\!|\ |\!\}$, $r \not\equiv \{\!|\ |\!\}$ and $e \not\models r$ contains some symbol $f \in FS$, or $e$ (resp. $r$) has the multiset constructor as outermost symbol.

$$(\mathsf{NEQ})_1 \ \frac{}{\Gamma \vdash_{\mathcal{P}} c(\bar{t}_n) \not\models d(\bar{s}_m)} \qquad c \in DC^n,\ d \in DC^m,\ c \not\equiv d,\ c, d \not\equiv \{\!|\cdot|\cdot|\!\}$$

$$(\mathsf{NEQ})_2 \ \frac{\Gamma \vdash_{\mathcal{P}} t_i \not\models s_i}{\Gamma \vdash_{\mathcal{P}} c(\bar{t}_n) \not\models c(\bar{s}_n)} \qquad c \in DC^n,\ c \not\equiv \{\!|\cdot|\cdot|\!\}, \text{ one rule for each } 1 \leq i \leq n$$

$$(\mathsf{NEQ})^S_3 \ \frac{\Gamma \vdash_{\mathcal{P}} t \notin ms}{\Gamma \vdash_{\mathcal{P}} \{\!|t|ts|\!\} \not\models ms} \qquad (\mathsf{NEQ})_4 \ \frac{\Gamma \vdash_{\mathcal{P}} t == l, \Gamma \vdash_{\mathcal{P}} ts \not\models ls}{\Gamma \vdash_{\mathcal{P}} \{\!|t|ts|\!\} \not\models \{\!|l|ls|\!\}}$$

Some comments are needed in order to clarify the rules above. Firstly, remark that all *CGORC*-rules associated to $\to$ and $==$ are similar to those presented in [4], except for rules $(\mathsf{PR})^i_\to$, $1 \leq i \leq 2$. Rule $(\mathsf{PR})^1_\to$ refers to approximation statements between primitive terms. In this case, we remit ourselves to the primitive level. On the other hand, $(\mathsf{PR})^2_\to$ deals with approximation statements whose left-hand side has $+$ or $*$ as the outermost operation, but contains some symbol in $FS$. In this case, it is enough to look for approximations $t^p_i$ for $e_i$, $1 \leq i \leq 2$, which must be primitive terms, and finally prove the approximation statement $t^p_1 \Diamond t^p_2 \to t^p_3$ (this will be done by means of rule $(\mathsf{PR})^1_\to$).

With respect to the *CGORC*-rules associated to $R^*_{\Sigma_\perp}(DV)$, note that the rules (HIP), (CONJ) and (EX) represent general forms of inference, accepted as valid in the intuitionistic fragment of classical logic; see e.g. [19]. Rule $(\mathsf{PR})_\varphi$ establishes that a primitive constraint $\varphi^p$ which involves only primitive variables ($lib(\varphi^p) \subseteq primvar(\Gamma)$) must be handled at the primitive level.

The *CGORC*-rules for $\in$ and $\notin$ are really very intuitive. Let us see now how to prove a disequality between two expressions $e$ and $r$. Rule $(\mathsf{NEQ})_\rightarrow$ has two tasks: To get approximations to the values represented by $e$ and $r$ (removing all function symbols) and, in presence of multisets, to reorder the elements by applying the *CGORC*-rule (OMUT). Disequality between terms not containing the multiset constructor at head can be checked by detecting disagreement of constructor symbols at head (rule $(\mathsf{NEQ})_1$) or by proving the disequality between some of the arguments (rule $(\mathsf{NEQ})_2$). In order to prove a disequality between two multisets $ms$ and $ms'$ we proceed as done in [3].

To conclude with the discussion, note that a constraint of the form $e \leq r$, where $e \leq r \notin R^*_p(DV)$, must be proved by using the *CGORC*-rule (LEQ), which "evaluates" $e$ and $r$ in order to get a couple of primitive terms, and then proving the corresponding constraint between such primitive terms.

**Semi-extended Data Terms and restricted *CGORC* Derivations:** The calculus *CGORC* that we have just defined will be used in Section 4 to prove soundness of a goal solving mechanism. Completeness of goal solving will rely on a *restricted use* of *CGORC*-provability. In order to introduce this idea, we define the set $T^*_{DC_\perp}(DV)$ of *semi-extended partial data terms* as the subset of $T^*_{\Sigma_\perp}(DV)$ composed of all those extended partial data terms not containing the symbols $+$, $*$, 0 and 1. Similarly, the set $T^*_{DC}(DV) \subseteq T^*_{DC_\perp}(DV)$ of *semi-extended total data terms* is composed of those semi-extended partial data terms not containing the constant symbol $\perp$. Note that a semi-extended primitive term is either a variable or an element from $\mathbb{R}$.

We will use the notation $\Vdash_{\mathcal{P}} \chi$ to indicate that $\chi$ can be proved (from $\Gamma = \emptyset$) using a *restricted CGORC* derivation. By definition, this means a *CGORC* derivation built according to the following limitations:

- ▷ Substitutions used in rule (OR) must map $DV$ to $T^*_{DC_\perp}(DV)$.
- ▷ Rules $(\mathsf{PR})^2_\rightarrow$, (OMUT), (C), $(\mathsf{NEQ})_\rightarrow$, $(\mathsf{MEMB})_\rightarrow$, $(\mathsf{NMEMB})_\rightarrow$ and (LEQ) must use approximation statements $e \rightarrow t$ with $t \in T^*_{DC_\perp}(DV)$.

Trivially, $\Vdash_{\mathcal{P}} \chi$ implies $\vdash_{\mathcal{P}} \chi$. As a consequence of Theorem 3 below, the converse implication will hold also. Semi-extended data terms will be useful also for the construction of Free Term Models in Section 3.

## 3 Model-Theoretic Semantics

In this section we present a model-theoretic semantics, showing also its relation to the rewriting calculus *CGORC* and its restricted use. We will make use of several basic notions from domain theory; see e.g. [17]. As notation, a *poset* $S$ will denote a set partially ordered by $\sqsubseteq$ and with least element $\perp$. We will write $\mathcal{C}(S)$ and $\mathcal{I}(S)$ for the sets of *cones* ($\sqsubseteq$-downward closed) and *ideals* (directed

cones) of $S$, respectively. $\mathcal{I}(S)$ ordered by set inclusion $\subseteq$ is a poset with bottom $\{\bot\}$, called the *ideal completion* of $S$. Mapping each $u \in S$ into the principal ideal $\langle u\rangle = \{v \in S | v \sqsubseteq u\}$ gives an order preserving embedding. It is known (see e.g. [16]) that $\mathcal{I}(S)$ is the least cpo $D$ s.t. $S$ can be embedded into $D$. Due to these results, our semantic constructions below could be reformulated in terms of Scott domains [17]. In particular, totally defined elements $u \in \mathsf{Def}(S)$ correspond to finite and maximal elements $\langle u\rangle$ in the ideal completion.

As in [7, 4], to represent non-deterministic lazy functions we use models with posets as carriers, interpreting function symbols as monotonic mappings from elements to cones. The elements of the poset are viewed as finite approximations of possibly infinite values. For given posets $D$ and $E$, we define the set of all *non-deterministic* and *deterministic* functions from $D$ to $E$, respectively as follows:

$$[D \to_{nd} E] = \{f : D \to \mathcal{C}(E) \mid \forall u, u' \in D : (u \sqsubseteq_D u' \Rightarrow f(u) \subseteq f(u'))\}$$
$$[D \to_{d} E] = \{f \in [D \to_{nd} E] \mid \forall u \in D : f(u) \in \mathcal{I}(E)\}$$

Note that any non-deterministic function $f$ can be extended to a monotonic mapping $f^* : \mathcal{C}(D) \to \mathcal{C}(E)$ defined as $f^*(C) = \bigcup_{c\in C} f(c)$. Abusing of notation, we will identify $f$ with its extension $f^*$.

### 3.1 Specification of $\not\models$, $\in$ and $\notin$ by Horn Clauses

Let us now define the behaviour of predicate symbols in $PS$ (except for $==$) by means of Horn clauses. Note that all Horn clauses below have a direct correspondence with the rewriting calculus $CGORC$, and they will determine the class of models of a program $\mathcal{P}$. The Horn clauses are the following:

$$H^1_{\in} : x \in \{\!|\, y | ys \,|\!\} \Leftarrow x == y \qquad H^1_{\notin} : x \notin \{\!|\, \,|\!\} \Leftarrow$$
$$H^2_{\in} : x \in \{\!|\, y | ys \,|\!\} \Leftarrow x \in ys \qquad H^2_{\notin} : x \notin \{\!|\, y | ys \,|\!\} \Leftarrow x \not\models y, x \notin ys$$
$$H^1_{\not\models} : c(\bar{x}_n) \not\models d(\bar{y}_m) \Leftarrow \quad \% \; c \not\equiv d,\; c, d \not\equiv \{\!|\, \cdot | \cdot \,|\!\}$$
$$H^2_{\not\models} : c(\bar{x}_n) \not\models c(\bar{y}_n) \Leftarrow x_i \neq y_i \quad \% \; c \not\equiv \{\!|\, \cdot | \cdot \,|\!\}, \text{ one clause for each } 1 \leq i \leq n$$
$$H^3_{\not\models} : \{\!|\, x | xs \,|\!\} \not\models ys \Leftarrow x \notin ys \qquad H^4_{\not\models} : \; xs \not\models \{\!|\, y | ys \,|\!\} \Leftarrow y \notin xs$$
$$H^5_{\not\models} : \{\!|\, x | xs \,|\!\} \not\models \{\!|\, y | ys \,|\!\} \Leftarrow x == y, xs \not\models ys$$
$$H^6_{\not\models} : \{\!|\, x | xs \,|\!\} \not\models \{\!|\, y | ys \,|\!\} \Leftarrow \{\!|\, x | xs \,|\!\} \to \{\!|\, x' | xs' \,|\!\}, \{\!|\, y | ys \,|\!\} \to \{\!|\, y' | ys' \,|\!\}, \{\!|\, x' | xs' \,|\!\} \not\models \{\!|\, y' | ys' \,|\!\}$$

### 3.2 Algebras and Models

We are now prepared to introduce our algebras, combining ideas from [18, 4]. A *polymorphic $\Sigma$-algebra* has the following structure:

$$\langle D^{\mathcal{A}}, T^{\mathcal{A}}, :^{\mathcal{A}}, \{K^{\mathcal{A}}\}_{K\in\{Real\}\cup TC}, \{\Diamond^{\mathcal{A}}\}_{\Diamond\in PO\cup DC}, \{\Diamond^{\mathcal{A}}\}_{\Diamond\in PP\cup PS}, \{f^{\mathcal{A}}\}_{f\in FS}\rangle$$

where:

(1) $D^{\mathcal{A}}$ (data universe) is a poset with partial order $\sqsubseteq^{\mathcal{A}}$ and bottom element $\bot^{\mathcal{A}}$. Furthermore, $D^{\mathcal{A}}$ contains a copy $D^{\mathcal{A}_p}$ of $\mathbb{R}$ given by a bijective mapping $h_p$ from $D^{\mathcal{A}_p}$ to $\mathbb{R}$, and verifies that: If $d \sqsubseteq^{\mathcal{A}} d'$ and $d' \in D^{\mathcal{A}_p}$ (resp. $d \in D^{\mathcal{A}_p}$), then $d \equiv \bot^{\mathcal{A}}$ or $d \equiv d'$ (resp. $d' \equiv d$)

(2) $T^{\mathcal{A}}$ (type universe) is any non-empty set.

(3) $:^{\mathcal{A}} \subseteq D^{\mathcal{A}} \times T^{\mathcal{A}}$ is a binary relation such that for all $\ell \in T^{\mathcal{A}}$, it holds that $\mathcal{E}^{\mathcal{A}}(\ell) = \{d \in D^{\mathcal{A}} \mid d :^{\mathcal{A}} \ell\} \in \mathcal{C}(D^{\mathcal{A}})$ and $\mathcal{E}^{\mathcal{A}}(Real^{\mathcal{A}}) = D^{\mathcal{A}_p} \cup \{\bot^{\mathcal{A}}\}$.
(4) For each $K \in TC^n$, $K^{\mathcal{A}} : (T^{\mathcal{A}})^n \to T^{\mathcal{A}}$ is a function.
(5) $0^{\mathcal{A}} = \{h_p^{-1}(0^{\mathcal{R}}), \bot^{\mathcal{A}}\} = \langle h_p^{-1}(0^{\mathcal{R}})\rangle$ and $1^{\mathcal{A}} = \{h_p^{-1}(1^{\mathcal{R}}), \bot^{\mathcal{A}}\} = \langle h_p^{-1}(1^{\mathcal{R}})\rangle$.
(6) For any $d_1, d_2 \in D^{\mathcal{A}}$, $\Diamond \in \{+, *\}$: $d_1 \Diamond^{\mathcal{A}} d_2 = \{h_p^{-1}(h_p(d_1) + h_p(d_2)), \bot^{\mathcal{A}}\} = \langle h_p^{-1}(h_p(d_1) + h_p(d_2))\rangle$, if $d_1, d_2 \in D^{\mathcal{A}_p}$ and $d_1 \Diamond^{\mathcal{A}} d_2 = \langle \bot^{\mathcal{A}}\rangle$, otherwise.
(7) For each $c \in DC^n$, $c^{\mathcal{A}} \in [(D^{\mathcal{A}})^n \to_d (D^{\mathcal{A}} - D^{\mathcal{A}_p})]$ and satisfies: For all $d_i \in D^{\mathcal{A}}$, $1 \leq i \leq n$, there exists $d \in (D^{\mathcal{A}} - D^{\mathcal{A}_p})$ such that $c^{\mathcal{A}}(d_1, \ldots, d_n) = \langle d\rangle$. Furthermore, if $d_i$ are totally defined, $1 \leq i \leq n$, then $d$ is totally defined.
(8) For all $d_1, d_2 \in D^{\mathcal{A}}$: $(d_1, d_2) \in \leq^{\mathcal{A}}$ iff $d_1, d_2 \in D^{\mathcal{A}_p}$ and $h_p(d_1) \leq h_p(d_2)$.
(9) For all $d_1, d_2 \in D^{\mathcal{A}}$: $(d_1, d_2) \in ==^{\mathcal{A}}$ iff $d_1, d_2$ are totally defined and $d_1 \equiv d_2$.
(10) For all $\Diamond \in \{\not=, \in, \notin\}$, $\Diamond^{\mathcal{A}} \subseteq (D^{\mathcal{A}})^2$ and it is monotonic. Moreover:
- If $d_1, d_2 \in D^{\mathcal{A}_p}$ and $h_p(d_1) \not\equiv h_p(d_2)$ then $(d_1, d_2) \in \not=^{\mathcal{A}}$.
- If $(d_1, d_2) \in \not=^{\mathcal{A}}$ and $d_2 \in D^{\mathcal{A}_p}$ (resp. $d_1 \in D^{\mathcal{A}_p}$), then $d_1 \in D^{\mathcal{A}_p}$ (resp. $d_2 \in D^{\mathcal{A}_p}$) and $h_p(d_1) \not\equiv h_p(d_2)$.
(11) For all $f \in FS^n$, $f^{\mathcal{A}} \in [(D^{\mathcal{A}})^n \to_{nd} D^{\mathcal{A}}]$.

In the following, $Alg(\Sigma)$ will denote the class of polymorphic $\Sigma$-algebras. Note that, Item (7) ensures that constructors are interpreted as deterministic mappings that preserve finite and maximal elements. Furthermore, note also that all primitive symbols in $\Sigma_p$ are interpreted in any algebra $\mathcal{A}$ according to their standard meaning in $\mathcal{R}$.

A *valuation* in $\mathcal{A}$ has the form $\xi = (\mu, \eta)$, where $\mu : TV \to T^{\mathcal{A}}$ is a *type valuation* and $\eta : DV \to D^{\mathcal{A}}$ is a *data valuation*. $\eta$ is called *totally defined* iff $\eta(x)$ is totally defined, for all $x \in DV$. $Val(\mathcal{A})$ denotes the set of all valuations over $\mathcal{A}$.

For a given $\xi = (\mu, \eta) \in Val(\mathcal{A})$, *type denotations* $[\![\tau]\!]^{\mathcal{A}}\xi = [\![\tau]\!]^{\mathcal{A}}\mu \in T^{\mathcal{A}}$ and *extended partial expression denotations* $[\![e]\!]^{\mathcal{A}}\xi = [\![e]\!]^{\mathcal{A}}\eta \in \mathcal{C}(D^{\mathcal{A}})$ are defined as usual, by considering that all $d \in \mathbb{R}$ is interpreted in $\mathcal{A}$ as $\langle h_p^{-1}(d)\rangle = \{h_p^{-1}(d), \bot^{\mathcal{A}}\}$.

We are particularly interested in those algebras that are well-behaved w.r.t. types. We say that $\mathcal{A} \in Alg(\Sigma)$ is *well-typed* iff for all $h : (\tau_1, \ldots, \tau_n) \to \tau_0 \in DC_\bot \cup FS \cup PO$, we have that $h^{\mathcal{A}}(\mathcal{E}^{\mathcal{A}}([\![\tau_1]\!]^{\mathcal{A}}\mu), \ldots, \mathcal{E}^{\mathcal{A}}([\![\tau_1]\!]^{\mathcal{A}}\mu)) \subseteq \mathcal{E}^{\mathcal{A}}([\![\tau_0]\!]^{\mathcal{A}}\mu)$, for every type valuation $\mu$. Also, for given $\xi = (\mu, \eta) \in Val(\mathcal{A})$, we say that $\xi$ is *well-typed* w.r.t an environment $V$ iff $\eta(x) \in \mathcal{E}^{\mathcal{A}}([\![\tau]\!]^{\mathcal{A}}\mu)$ for every $x : \tau \in V$. Reasoning by structural induction, we can prove that expression denotations behave as expected w.r.t. well-typed algebras and valuations:

**Theorem 1.** *Let $V$ be an environment. Let $\mathcal{A} \in Alg(\Sigma)$ be well-typed and $\xi = (\mu, \eta) \in Val(\mathcal{A})$ well-typed w.r.t. $V$. For all $e \in E^{*\tau}_{\Sigma_\bot}(V)$, $[\![e]\!]^{\mathcal{A}}\eta \subseteq \mathcal{E}^{\mathcal{A}}([\![\tau]\!]^{\mathcal{A}}\mu)$.*

Next we define the notion of *model*. Consider $\mathcal{A} \in Alg(\Sigma)$. Let $\eta$ be a data valuation over $\mathcal{A}$. Then:

▷ $(\mathcal{A}, \eta) \models e_1 \Diamond e_2$, $\Diamond \in PP \cup PS$, iff there exist $d_i \in [\![e_i]\!]^{\mathcal{A}}\eta$, $1 \leq i \leq 2$, such that $(d_1, d_2) \in \Diamond^{\mathcal{A}}$.
▷ $(\mathcal{A}, \eta) \models \varphi_1 \wedge \varphi_2$ iff $(\mathcal{A}, \eta) \models \varphi_i$, $1 \leq i \leq 2$.

- ▷ $(\mathcal{A}, \eta) \models \exists x\varphi$ iff there exists $d \in D^{\mathcal{A}}$ such that $(\mathcal{A}, \eta[x \leftarrow d]) \models \varphi$.
- ▷ $(\mathcal{A}, \eta) \models e \to e'$ iff $[\![e']\!]^{\mathcal{A}}\eta \subseteq [\![e]\!]^{\mathcal{A}}\eta$.
- ▷ $\mathcal{A} \models \{\!\{ x, y|xs \}\!\} \approx \{\!\{ y, x|xs \}\!\}$ iff for any $\eta$, it holds that $[\![\{\!\{ x, y|xs \}\!\}]\!]^{\mathcal{A}}\eta = [\![\{\!\{ y, x|xs \}\!\}]\!]^{\mathcal{A}}\eta$.
- ▷ $\mathcal{A}$ satisfies a Horn clause of the form $A \Leftarrow B_1, \ldots, B_m$ iff for any data valuation $\eta$ such that $(\mathcal{A}, \eta) \models B_i$, $1 \leq i \leq m$, it holds that $(\mathcal{A}, \eta) \models A$.
- ▷ $\mathcal{A}$ satisfies a defining rule $e \to r \Leftarrow \varphi$ iff every data valuation $\eta$ such that $(\mathcal{A}, \eta) \models \varphi$ verifies that $(\mathcal{A}, \eta) \models e \to r$.
- ▷ Let $\mathcal{P} = \langle \Sigma, \mathrm{R} \rangle$ be a program. $\mathcal{A} \models \mathcal{P}$ iff $\mathcal{A}$ satisfies every defining rule in R and (MSET).

The class $\mathcal{M} \subseteq Alg(\Sigma)$ of polymorphic algebras is composed of those $\mathcal{A} \in Alg(\Sigma)$ such that $\mathcal{A}$ satisfies (MSET) and all Horn clauses in Subsection 3.1, whereas $\mathcal{M}(\mathcal{P}) =_{Def} \{\mathcal{A} \in \mathcal{M} \mid \mathcal{A} \models \mathcal{P}\}$.

**Definition 1.** ***(Logical consequence)*** *Consider $\Gamma \subseteq R^*_{\Sigma_\perp}(DV)$ in solved form and $\chi \equiv e \to t$ or $\chi \in R^*_{\Sigma_\perp}(DV)$, where $e \in E^*_{\Sigma_\perp}(DV)$, $t \in T^*_{\Sigma_\perp}(DV)$. It holds that $\Gamma \models \chi$ iff for all $\mathcal{A} \in \mathcal{M}(\mathcal{P})$, and for all $\eta \in Val(\mathcal{A})$ totally defined, if $(\mathcal{A}, \eta) \models \Gamma$ then $(\mathcal{A}, \eta) \models \chi$.*

The following theorem establishes the soundness of the rewriting calculus *CGORC*. It can be proved by induction on *CGORC* derivations.

**Theorem 2.** ***(Soundness of** CGORC**)*** *Consider $\Gamma \subseteq R^*_{\Sigma_\perp}(DV)$ in solved form, and $\chi \in R^*_{\Sigma_\perp}(DV)$ or $\chi \equiv e \to t$, where $e \in E^*_{\Sigma_\perp}(DV)$, $t \in T^*_{\Sigma_\perp}(DV)$. If $\Gamma \vdash_{\mathcal{P}} \chi$ then $\Gamma \models \chi$.*

### 3.3 Free Term Models

Given a program $\mathcal{P} = \langle \Sigma, \mathrm{R} \rangle$ and an environment $V$, we define the term algebra $MT(V, \mathcal{P})$ as follows:

• Let $X$ be the set of data variables occurring in $V$. The set $T^*_{DC_\perp}(X)/{\approx_{Mset}}$ is the data universe, where $T^*_{DC_\perp}(X)/{\approx_{Mset}} =_{Def} \{[t] \mid t \in T^*_{DC_\perp}(X)\}$. $[t]$ is defined as the set $\{t' \in T^*_{DC_\perp}(X) \mid t \approx_{Mset} t'\}$, where $t \approx_{Mset} t'$ iff $\Vdash_{\mathcal{P}} t \to t'$ and $\Vdash_{\mathcal{P}} t' \to t$.

The bottom element is $[\perp] = \{\perp\}$, and the partial order $\sqsubseteq^{MT(V,\mathcal{P})}$ is defined as: For any $[t], [t'] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$, $[t] \sqsupseteq^{MT(V,\mathcal{P})} [t']$ iff $\Vdash_{\mathcal{P}} t \to t'$.

Note that $T^*_{DC_\perp}(X)/{\approx_{Mset}}$ contains $\{[d] \mid d \in \mathbb{R}\}$ as copy of $\mathbb{R}$ given by the bijective mapping $h_p([d]) = d$, for all $d \in \mathbb{R}$.

• Let $A$ be the set of type variables occurring in $V$.
• The type universe is $Type(A) = \{\tau \in Type_{TC}(TV) \mid tvar(\tau) \subseteq A\}$.
• For all $[t] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$, $\tau \in Type(A)$, $[t] :^{MT(V,\mathcal{P})} \tau$ iff $t \in T^{*\tau}_{DC_\perp}(V)$.
• For all $K \in TC^n \cup \{Real\}$, $\tau_i \in Type(A)$: $K^{MT(V,\mathcal{P})}(\tau_1, \ldots, \tau_n) = K(\tau_1, \ldots, \tau_n)$.
• $0^{MT(V,\mathcal{P})} = \langle [0^{\mathcal{R}}] \rangle$ and $1^{MT(V,\mathcal{P})} = \langle [1^{\mathcal{R}}] \rangle$.
• For all $[t], [t'] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$, $\Diamond \in \{+, *\}$: $[t] \Diamond^{MT(V,\mathcal{P})} [t'] = \langle [t \Diamond t'] \rangle$, if $t, t' \in \mathbb{R}$ and $[t] \Diamond^{MT(V,\mathcal{P})} [t'] = \langle [\perp] \rangle$ otherwise.
• For all $c \in DC^n$, $[t_i] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$: $c^{MT(V,\mathcal{P})}([t_1], \ldots, [t_n]) = \langle [c(t_1, \ldots, t_n)] \rangle$.
• For all $[t_i] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$: $([t_1], [t_2]) \in \leq^{MT(V,\mathcal{P})}$ iff $t_1, t_2 \in \mathbb{R}$ and $t_1 \leq t_2$.
• For all $[t_i] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$: $([t_1], [t_2]) \in ==^{MT(V,\mathcal{P})}$ iff $[t_1] = [t_2]$ and $t_1 \in T^*_{DC}(X)$.

- For all $[t_i] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$, $\Diamond \in \{\not\models, \in, \notin\}$: $([t_1],[t_2]) \in \Diamond^{MT(V,\mathcal{P})}$ iff $\Vdash_{\mathcal{P}} t_1 \Diamond t_2$.
- For all $[t_i] \in T^*_{DC_\perp}(X)/{\approx_{Mset}}$, $1 \leq i \leq n$, $f \in FS^n$:
  $f^{MT(V,\mathcal{P})}([t_1],\ldots,[t_n]) = \{[t] \in T^*_{DC_\perp}(X)/{\approx_{Mset}} \mid \Vdash_{\mathcal{P}} f(t_1,\ldots,t_n) \to t\}$.

Next result ensures that $MT(V,\mathcal{P})$ belongs to the class of algebras $\mathcal{M}(\mathcal{P})$, and that restricted *CGORC*-derivability is sound and complete with respect to our notion of model.

**Theorem 3.** ***(Adequateness of*** $MT(V,\mathcal{P})$***)*** *Let $\mathcal{P} = \langle \Sigma, \mathrm{R}\rangle$ be a program and $V$ an environment. It holds:*

*(1) $MT(V,\mathcal{P}) \in \mathcal{M}(\mathcal{P})$ and if $\mathcal{P}$ is well-typed then $MT(V,\mathcal{P})$ is well-typed.*

*(2) Consider $\chi \equiv e \to t$ or $\chi \in R^*_{\Sigma_\perp}(X)$, where $e \in E^*_{\Sigma_\perp}(X)$, $t \in T^*_{DC_\perp}(X)$. Then the following statements are equivalent:*

*(2.1) $\Vdash_{\mathcal{P}} \chi$.*

*(2.2) $(\mathcal{A},\eta) \models \chi$, for all $\mathcal{A} \in \mathcal{M}(\mathcal{P})$ and for all $\eta$ totally defined.*

*(2.3) $(MT(V,\mathcal{P}),[id]) \models \chi$, where id is the identity substitution over $X$.*

Due to Theorem 2, *unrestricted CGORC*-derivability is sound w.r.t. models in the class $\mathcal{M}(\mathcal{P})$. Nevertheless, completeness in the sense of Theorem 3 holds only for *restricted CGORC*-derivability. For instance, it is true that $(MT(V,\mathcal{P}),[id]) \models x+1 \to x+1$, but there is no *CGORC* proof for $\vdash_{\mathcal{P}} x+1 \to x+1$. The point is that $x+1$ *is not a semi-extended data term.*

We can also give a characterization of $MT(V,\mathcal{P})$ as a *free object* in the category of all models of $\mathcal{P}$. This relies on a notion of morphism similar to that from [4], extended to deal with constraints in a natural way. See [2] for details.

## 4 Operational Semantics

This section presents a *Lazy Narrowing Calculus* (*LNC* for short), which is a goal solving procedure that combines lazy narrowing with unification modulo (MSET) and constraint solving. In order to ensure the completeness of *LNC* (Theorem 4), the process of solving a goal is divided in two main phases, as done in [8, 5]. A *derivation* for a goal $G$ (composed of constraints) is a finite sequence of $\hookrightarrow_{\mathcal{P}}$-steps (named $\hookrightarrow_{\mathcal{P}}$-derivation) followed by a finite sequence of $\hookrightarrow_{DV}$-steps (named a $\hookrightarrow_{DV}$-derivation). The $\hookrightarrow_{\mathcal{P}}$-derivation transforms $G$ into a quasi-solved goal $G'$ containing only approximation statements of the form $t \to x$ and constraints in solved form, while the $\hookrightarrow_{DV}$-derivation processes variables, thereby transforming $G'$ into a solved goal which represents a solution for $G$. The whole process preserves well-typing.

Formally, an *admissible goal* $G$ for a program $\mathcal{P}$ has the structure $G \equiv \exists \bar{u} \cdot S \square P \square R$, where:

- $S \equiv x_1 = t_1, \ldots, x_n = t_n$ is a system of equations in *solved form*, i.e., $x_i$ occurs exactly once in $G$. Note that $S$ represents the substitution $\delta_S \in DSub(T_\Sigma(DV))$ defined as $\delta_S(x_i) = t_i$, $1 \leq i \leq n$ and $\delta_S(z) = z$ otherwise.
- $P \equiv e_1 \to s_1, \ldots, e_k \to s_k$ is a multiset of approximation statements.
- $R \subseteq R_\Sigma(DV)$ is a multiset of constraints, and must fulfill several technical requirements similar to those presented in [7, 5] in order to achieve soundness and completeness, along with the new condition:
  (PR) For all $e \Diamond t \in S \cup P$, $\Diamond \in \{=, \to\}$, $t$ does not contain to 0, 1, + and $*$.

$G \equiv \exists \bar{u} \cdot S \Box P \Box R$ is *well-typed* iff there exists an environment $V$ such that for all $e \diamondsuit e' \in S \cup P$, $\diamondsuit \in \{\rightarrow, =\}$, there exists $\tau \in Type_{TC}(TV)$ such that $e, e' \in E^{\tau}_{\Sigma}(V)$, and for all $\varphi \in R$, $\varphi$ is well-typed w.r.t. $V$.

### 4.1 Demanded and Pending Variables

Similarly to [7, 5], *LNC* uses a notion of *demanded variable* to deal with lazy evaluation. Intuitively, approximation statements $e \rightarrow x$ in $G$, where $e$ contains some function symbol, do not propagate the binding $x/e$. Instead, evaluation of $e$ must be triggered, *provided that $x$ is demanded*. The result will be shared by all the occurrences of $x$.

A variable $x$ of $G \equiv \exists \bar{u} \cdot S \Box P \Box R$ is *demanded* iff $P$ contains a sequence of approximation statements of the form: $t_0 \rightarrow x_1, t_1 \rightarrow x_2, \ldots, t_n \rightarrow x_{n+1}$, where $t_i \in T_{\Sigma}(DV)$, $0 \leq i \leq n$, $x \in dvar(t_0)$, $x_i \in DV$, $1 \leq i \leq n+1$, $x_i \in dvar(t_i)$, $1 \leq i \leq n$, and one of the following conditions holds:

- $x_{n+1} \diamondsuit e \in R$ or $e \diamondsuit x_{n+1} \in R$, where $\diamondsuit \in \{==, \not=, \in\}$.
- $e \notin x_{n+1} \in R$ or $x_{n+1} \notin \{\!|\, e | es \,|\!\} \in R$.
- $x_{n+1} \in lib(\varphi^p)$, where $\varphi^p \in R_p(DV) \cap R$.

Demanded variables $x$ are forced by solutions to take values different from $\bot$. This justifies why $x$ is not considered as demanded in constraints $x \notin es$, where $es$ has not the form $\{\!|\, r | rs \,|\!\}$.

We also need a notion of *pending variable* to detect situations where goal solving must proceed by trying different imitation bindings. Formally, given a goal $G \equiv \exists \bar{u} \cdot S \Box P \Box R$, we say that $x_n$ is pending in $G$ iff $P$ contains a sequence of the form $e \rightarrow x_0, t_1 \rightarrow x_1, t_2 \rightarrow x_2, \ldots, t_n \rightarrow x_n$, where $e \in E_{\Sigma}(DV) - T_{\Sigma}(DV)$, $t_i \in T_{\Sigma}(DV)$, $1 \leq i \leq n$, $x_i \in DV$, $0 \leq i \leq n$ and $x_i \in dvar(t_{i+1})$, $0 \leq i \leq n-1$. The set of pending variables of $G$ will be noted as $\mathsf{pend}(G)$ in the sequel.

As an example, consider a goal of the form $\exists \bar{u} \cdot S \Box e \rightarrow x \Box x \notin xs$, where $e$ contains defined function symbols and possibly denotes an infinite value. Variable $x$ is not demanded and the constraint $x \notin xs$ is solved, but the whole goal is not solved. Since $x$ is pending, the *LNC* calculus will allow goal solving to progress by trying two possible imitation bindings for $xs$: binding $xs$ to $\{\!|\ |\!\}$ will lead immediately to a solved goal, while binding $xs$ to $\{\!|\, y | ys \,|\!\}$ will produce a new goal including a constraint $x \notin \{\!|\, y | ys \,|\!\}$, which can be further processed.

### 4.2 Goals in Solved Form, Answers and Solutions

An admissible goal $G$ is *quasi-solved* iff $G$ has the form $\exists \bar{u} \cdot S \Box P \Box R$, where $R$ contains only constraints in solved form or equalities $x == y$, $x, y \in DV$, and $P$ contains only approximation statements of the form $t \rightarrow x$, where $t \in T_{\Sigma}(DV) - DV$ and $x$ is not a demanded variable, or $t \in DV$. Note that it is important to require $x$ to be not demanded in such approximation statements. This is needed in order to preserve quasi-solved goals when applying variable elimination rules. For instance, if we allow the goal $G \equiv \Box A \rightarrow x \Box x \not= A$, where $A$ is a constant symbol (note that $x$ is demanded), then when propagating the binding $x/A$, the resulting goal $G' \equiv \Box \Box A \not= A$ would not be quasi-solved.

An admissible goal $G$ is in *solved form* iff $G$ has the form $\exists \bar{u} \cdot S \Box \Box R$, where $R$ contains only constraints in solved form, and for all $x == y \in R$, $x, y \in DV$,

it holds that $x == y \in PP(G)$, where $PP(G)$ is defined similarly to $PP(\Gamma)$ but now considering that also equality constraints $x == y$ contribute to $PP(G)$.

W. r. t. primitive constraints, *LNC* is not going to perform any explicit transformation. Without loss of generality, we can suppose that all *LNC*-irreducible goals suffer a process of simplification similar to that in $CLP(\mathcal{R})$. This is possible because all the primitive constraints within a goal $G$ in solved form will be isolated in the primitive part $PP(G)$.

A *correct answer* for $G \equiv \exists \bar{u} \cdot S \Box P \Box R$ is a pair $(\delta, \Gamma)$ such that $\delta \in DSub(T^*_{DC}(DV))$, $\Gamma \subseteq R_{\Sigma_\perp}(DV)$ is finite and in solved form and there exists $\bar{t} \in T^*_{DC_\perp}(DV)$ such that $\delta' = \delta[\bar{u}/\bar{t}]$ (called an *existential extension* of $\delta$ in the sequel) verifies that:

- ▷ There exists a primitive valuation $\eta^p$ over $\mathcal{R}$ such that $(\mathcal{R}, \eta^p) \models_{\mathcal{R}} PP(\Gamma)$.
- ▷ For every equation $x = s \in S$: $x\delta' \equiv s\delta' \in T^*_{DC}(DV)$.
- ▷ For all $\chi \in P \cup R$, it holds that $\Gamma \vdash_{\mathcal{P}} \chi\delta'$. The multiset containing all *CGORC*-proofs for all the elements in $(P \cup R)\delta'$ is named a witness $\mathcal{M}$ for $G$ and $(\delta, \Gamma)$.

A *solution* for $G$ is a correct answer $(\delta, \Gamma)$ for $G$ such that $\Gamma \equiv \emptyset$. Note that the condition (PR) in admissible goals ensures that every $e \to t \in G$ verifies that $t$ does not contain the symbols $+$, $*$, 0 and 1. But, Theorems 2 and 3 ensure that $\vdash_{\mathcal{P}}$ and $\Vdash_{\mathcal{P}}$ are equivalent in presence of approximation statements of the form $e \to t$, where $e \in E^*_{\Sigma_\perp}(DV)$, $t \in T^*_{DC_\perp}(DV)$ and constraints $\varphi \in R^*_{\Sigma_\perp}(DV)$. Thus, given any solution $\delta \in DSub(T^*_{DC}(DV))$ for $G$ and $\chi \in P \cup R$, to prove $\vdash_{\mathcal{P}} \chi\delta'$ is equivalent to prove that $\Vdash_{\mathcal{P}} \chi\delta'$, where $\delta'$ is an existential extension of $\delta$. This means that the notion of solution for a goal can be established in terms of *restricted CGORC*-derivations.

On the contrary, the notion of correct answer needs the full power of *unrestricted CGORC*-derivations. For instance, $(\{\,\}, \{x == y\})$ happens to be a *LNC* computed answer for the goal $G \equiv c(x{+}1) == c(y{+}1)$. According to the Soundness Theorem 4 below, this computed answer must be correct, and there must be some *unrestricted CGORC*-derivation proving $\{x == y\} \vdash_{\mathcal{P}} c(x{+}1) == c(y{+}1)$.

### 4.3 *LNC* Transformation Rules. Soundness and Completeness

Due to lack of space, we will not present completely all the transformation rules for *LNC* but we will give the main ideas behind the transformation rules. A complete description of *LNC* can be found in [2].

All transformation rules in *LNC* have been designed in order to get a completeness result (Theorem 4) whose proof is based on the following idea: Given an admissible goal $G$ different from *FAIL* and not solved, and a solution $\delta$ for $G$, it is possible to find a transformation rule $T$ such that when we apply $T$ to $G$, "something" decreases while preserving (possibly modulo (MSET)) the solution $\delta$. Here, *FAIL* represents an irreducible and inconsistent goal.

In presence of non quasi-solved goals, "something" refers to witness, and $\delta$ is totally preserved. By decreasing we refer to the following multiset ordering: Let $\mathcal{M} = \{\!\!\{ \Pi_1, \ldots, \Pi_n \}\!\!\}$ and $\mathcal{M}' = \{\!\!\{ \Pi'_1, \ldots, \Pi'_m \}\!\!\}$ be multisets of $\Vdash_{\mathcal{P}}$-proofs, then $\mathcal{M}$ is smaller than $\mathcal{M}'$ iff $\{\!\!\{ |\Pi_1|, \ldots, |\Pi_n| \}\!\!\} \prec \{\!\!\{ |\Pi'_1|, \ldots, |\Pi'_m| \}\!\!\}$, where $|\Pi|$ is

the size (i.e., the number of inference steps without considering the applications of the rule $(\mathsf{PR})_\varphi$) of $\Pi$, and $\prec$ is the multiset extension of the usual ordering over the natural numbers.

In presence of quasi-solved goals, "something" refers to $G$, and $\delta$ is preserved modulo (MSET). Now, by decreasing we refer to the following lexicographic ordering: Given any two quasi-solved goals $G \equiv \exists\bar{u}\cdot S\Box P\Box R$ and $G' \equiv \exists\bar{u}'\cdot S'\Box P'\Box R'$, we say that $G$ is smaller than $G'$ iff $(n_1, m_1) < (n_2, m_2)$, where $n_1$ (resp. $n_2$) is the number of approximation statements in $G$ (resp. in $G'$), whereas $m_1$ (resp. $m_2$) is the number of constraints of the form $x == y$, such that $x == y \notin PP(G)$ (resp. $x == y \notin PP(G')$).

Now, in order to design $LNC$, it is enough to analyze the different kinds of approximation statements and constraints in an admissible goal $G$ which is not yet solved, looking for some transformation which allows to ensure what we have commented above. This is done by analizing the possible structure of a goal $G$. In the analysis below, $\delta$ will denote a solution for $G$ and $\delta'$ will be an existential extension of $\delta$. With respect to approximation statements, let us analyze some of the possibilities. If $G$ is of the form:

▷ $G \equiv \exists\bar{u}\cdot S\Box c(\bar{e}_n) \to d(\bar{t}_m), P\Box R$, where $c \in DC^n$, $d \in DC^m$. The definition of solution establishes that $c(\bar{e}_n)\delta' \to d(\bar{t}_n)\delta'$ must be $\Vdash_{\mathcal{P}}$-provable. Such a proof must use as first inference rule (DC) or (OMUT). According to these two $CGORC$-rules, we have, respectively, the following two $\hookrightarrow_{\mathcal{P}}$-rules:

$\mathsf{Dec}_{\to}$: $\exists\bar{u}\cdot S\Box c(\bar{e}_n) \to c(\bar{t}_n), P\Box R \hookrightarrow_{\mathcal{P}} \exists\bar{u}\cdot S\Box e_1 \to t_1, \ldots, e_n \to t_n, P\Box R$

$\mathsf{Mut}_{\to}$: $\exists\bar{u}\cdot S\Box\{\!|\, e|es \,|\!\} \to s, P\Box R \hookrightarrow_{\mathcal{P}}$
$\exists\bar{u}, x, y, xs\cdot S\Box e \to x, es \to \{\!|\, y|xs \,|\!\}, \{\!|\, y, x|xs \,|\!\} \to s, P\Box R$

where $x, y, xs$ are fresh variables.

▷ $G \equiv \exists\bar{u}\cdot S\Box f(\bar{e}_n) \to t, P\Box R$, where $f \in FS^n$. Now, the only possibility of proving $\Vdash_{\mathcal{P}} f(\bar{e}_n)\delta' \to t\delta'$ is using as last inference rule (OR) but ensuring that $t\delta' \not\equiv \bot$. Hence, the $\hookrightarrow_{\mathcal{P}}$-transformation rule will be:

$\mathsf{Rule}_{\to}$: $\exists\bar{u}\cdot S\Box f(\bar{e}_n) \to t, P\Box R \hookrightarrow_{\mathcal{P}} \exists\bar{u}, \bar{x}\cdot S\Box e_1 \to t_1, \ldots, e_n \to t_n, r \to t, P\Box\varphi, R$

If $t \notin DV$ or $t$ is a demanded variable (which ensures that $t\delta' \not\equiv \bot$), where $f(\bar{t}_n) \to r \Leftarrow \varphi$ is a fresh variant of a rule in R with variables $\bar{x}$.

The rest of cases can be reasoned similarly; see [2]. With respect to constraints, let us analyze several cases:

▷ $G \equiv \exists\bar{u}S\Box P\Box e \in es, R$, where $es \notin DV$. Then, if $es$ contains a function symbol, according to $(\mathsf{MEMB})_{\to}$, we will have the rule:

$\mathsf{Memb}_{\to}$: $\exists\bar{u}\cdot S\Box P\Box e \in es, R \hookrightarrow_{\mathcal{P}} \exists\bar{u}, x, xs\cdot S\Box es \to \{\!|\, x|xs \,|\!\}, P\Box e \in \{\!|\, x|xs \,|\!\}, R.$

where $x, xs$ are fresh variables.

Otherwise ($es$ have the form $\{\!|\, t|ts \,|\!\}$), then we will have two new rules, according to the $CGORC$-rules $(\mathsf{MEMB})_1$ and $(\mathsf{MEMB})_2$ respectively, that can be designed similarly to $\mathsf{Mem}_{\to}$.

▷ $G \equiv \exists\bar{u}\cdot S\Box P\Box e \notin xs, R$, where $xs \in DV$. If $e$ contains some function symbol in $FS$ or $e \in T_\Sigma(DV)$ but $e$ contains some variable of the set $\mathsf{pend}(G)$, then let us analyze all possible forms of the proof $\Vdash_{\mathcal{P}} e\delta' \notin xs\delta'$. If such a proof has used $(\mathsf{NMEMB})_1$, then we have the following $LNC$-transformation rule:

Nmemb III: $\exists \bar{u} \cdot S \Box P \Box e \notin xs, R \hookrightarrow_{\mathcal{P}} \exists \bar{u} \cdot xs = \{\!\!\{\ \}\!\!\}(S \Box P \Box R)[xs/\{\!\!\{\ \}\!\!\}]$

Otherwise, the proof $\Vdash_{\mathcal{P}} e\delta' \notin xs\delta'$ has used $(\mathsf{NMEMB})_2$, and the corresponding $LNC$-rule will be defined.

The rest of rules for solving constraints can be designed similarly. Something similar happens with $\hookrightarrow_{DV}$-rules. In presence of approximation statements of the form $t \rightarrow x$, where $t$ is either a non-variable primitive term or a variable belonging to $primvar(G)$, we generate a new goal transforming $t \rightarrow x$ into $t == x$. Otherwise, $t \rightarrow x$ disappear, propagating the binding $x/t$.

The soundness and completeness theorem for $LNC$, whose proof can be found in [2], is given below. As notation, $\delta' =_{Mset} \delta$ means that $\delta'(x) \approx_{Mset} \delta(x)$, for all $x \in DV$.

**Theorem 4.** ***(Soundness and completeness)***
*Soundness: Let $G$ be an initial goal, $G'$ a quasi-solved goal and $G'' \equiv \exists \bar{u} \cdot S \Box \Box RR$ a goal in solved form such that $G \hookrightarrow^*_{\mathcal{P}} G' \hookrightarrow^*_{DV} G''$. Then $(\delta_S, RR)$ is a correct answer for $G$.*
*Completeness: Let $\mathcal{P} = \langle \Sigma, \mathrm{R} \rangle$ be a program, $G$ an initial goal and $\delta$ a solution for $G$. Then there exist a quasi-solved goal $G'$ and a goal $G''$ in solved form such that $G \hookrightarrow^*_{\mathcal{P}} G' \hookrightarrow^*_{DV} G'' \equiv \exists \bar{u} \cdot S \Box \Box RR$, and $G''$ has a solution $\sigma$ verifying that $\sigma =_{Mset} \delta$. Furthermore, if $\mathcal{P}$ and $G$ are well-typed then $G\delta_S$ is well-typed.*

Note that our completeness result is restricted to solutions instead to correct answers. For instance, consider the program rules $f \rightarrow Zero$ and $g(y, ys) \rightarrow True \Leftarrow y \notin ys$, and the initial goal $G \equiv \Box\Box g(Suc(f), zs) == True$. It is easy to check that $(\{\ \}, \underbrace{\{Suc(Zero) \notin zs\}}_{\Gamma})$ is a correct answer for $G$. However, $LNC$ can not compute the correct answer $(\{\ \}, \{Suc(Zero) \notin zs\})$. Instead, $LNC$ enumerates infinite solutions $zs = \{\!\!\{\ \}\!\!\}$, $zs = \{\!\!\{ Zero \}\!\!\}$, ..., i.e., the correct answer is covered by an infinite number of solutions.

## 5 Conclusions

Starting from the framework of [4, 5], limited to the case where all the data constructors except the multiset constructor are free, we have added built-in constraints over the domain of real numbers and symbolic constraints over constructed terms. The resulting language (SETA) has a firm mathematical foundation, with proof-theoretic and model-theoretic semantics, as well as a sound and complete goal solving mechanism. Moreover, SETA seems to have a potential wide range of applications, including parsing of visual languages, which are worth of further investigation.

Our narrowing calculus is not intended to serve, "as it is", as a concrete computational model. An actual implementation should avoid an indiscriminate use of the multiset equation (MSET), which can obviously lead to useless infinite computations. A first attempt to build such an implementation has been presented in our previous paper [3], where neither mathematical foundations nor built-in arithmetic constraints were considered. More work on implementation methods is still needed, also in regard to efficient narrowing strategies such as *demand driven*, also called *needed* narrowing [13, 1], whose generalization to the case of *non-free* data constructors does not seem obvious.

## References


1. Antoy, R., Echahed, R., Hanus, M.: *A Needed Narrowing Strategy.* Proc. POPL'94, ACM Press, pp. 268–279.
2. Arenas-Sánchez P., López-Fraguas F.J., Rodríguez-Artalejo M.: *Functional plus Logic Programming with Built-in and Symbolic Constraints.* Technical Report SIP-85/98, UCM. Available at ftp://147.96.25.167/pub/seta.ps.gz
3. Arenas-Sánchez P., López-Fraguas F.J., Rodríguez-Artalejo M.: *Embedding Multiset Constraints into a Lazy Functional Logic Language.* Proc. PLILP'98, Springer LNCS 1490, pp. 429–444, 1998.
4. Arenas-Sánchez P., Rodríguez-Artalejo M.: *A Semantic Framework for Functional Logic Programming with Algebraic Polymorphic Types.* Proc. TAPSOFT'97, Springer LNCS 1214, pp. 453–464, 1997.
5. Arenas-Sánchez P., Rodríguez-Artalejo M.: *A Lazy Narrowing Calculus for Functional Logic Programming with Algebraic Polymorphic Types.* Proc. ILPS'97, the MIT Press, pp. 53–69, 1997.
6. Banâtre, J.P. and Le Métayer, D.: *The Gamma model and its discipline of programming.* Science of Computer Programming 15, pp. 55–77, 1990.
7. González-Moreno J.C., Hortalá-González T., López-Fraguas F.J, Rodríguez-Artalejo M.: *An Approach to Declarative Programming Based on a Rewriting Logic.* Journal of Logic Programming, Vol. 4, $n^0$. 1, pp. 47–87, 1999.
8. Hanus, M.: *Lazy Narrowing with Simplification.* Journal of Computer Languages, Vol. 23, $n^0$. 2–4, pp. 61–85, 1997.
9. Hanus M.: *The Integration of Functions into Logic Programming. A Survey.* Journal of Logic Programming Special issue "Ten Years of Logic Programming" (19 & 20):583–628, 1994.
10. Helm R., Marriot K.: *Declarative Specification and Semantics for Visual Languages.* Journal of Visual Languages and Computing, 2:211–331, 1991.
11. Jaffar J., Maher M.J.: *Constraint Logic Programming: A Survey.* Journal of Logic Programming 19–20, pp. 503–582, 1994.
12. Jaffar J., Michaylov S., Stuckey P.J., Yap R.H.C.: *The CLP($\mathcal{R}$) Language and System.* ACM Transactions on Programming Languages and Systems, Vol. 14, No. 3, pp. 339–395, Julio 1992.
13. Loogen R., López Fraguas F.J., Rodríguez Artalejo M.: *A Demand Driven Computation Strategy for Lazy Narrowing.* Proc. PLILP'93, Springer LNCS 714, pp. 184–200, 1993.
14. Marriott, K.: *Constraint multiset grammars.* In Proc. IEEE Symposium on Visual Languages, IEEE Computer Society Press, pp. 118–125, 1994.
15. Martí-Oliet N., Meseguer J.: *Action and Change in Rewriting Logic.* In R. Pareschi & B. Fronhöfer (eds.). Theoretical Approaches to Dynamic Worlds in Computer Science and Artificial Intelligence. Cambridge M.P., 1995.
16. Möller B.: *On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types.* Acta Informatica 22, pp. 537–578, 1985.
17. Scott D.S.: *Domains for Denotational Semantics.* Proc. ICALP'82. Springer LNCS 140, pp. 567–613, 1982.
18. Smolka G.: *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, 1989.
19. Van Dalen D.: *Logic and Structure.* Berlin, Heidelberg, New York. Springer Verlag, 1980.