A Higher-Order Demand-driven Narrowing Calculus with Definitional Trees

Rafael del Vado Vírseda *

Dpto. de Sistemas Informáticos y Computación
Universidad Complutense de Madrid
rdelvado@sip.ucm.es

Abstract. We generalize the *Constructor-based ReWriting Logic* CRWL to the setting of the simply typed λ-calculus, where theories are presented by conditional overlapping fully extended pattern rewrite systems. We claim that this logic is useful for higher-order functional-logic programming, and propose a *Higher-Order Lazy Narrowing calculus* HOLN^{DT} for answering joinability and reducibility queries, in which a variant of *Definitional Trees* is used to efficiently control the demand-driven narrowing strategy. The calculus HOLN^{DT} is shown to be sound and strongly complete with respect to this higher-order conditional rewriting logic.

1 Introduction

The effort to identify suitable computational models for higher-order functional logic programming has grown in recent years. Functional-logic languages with a sound and complete operational semantics are mainly based on *narrowing*, a transformation rule which combines the basic execution mechanisms of functional and logic languages, namely *rewriting* with *unification*. All serious attempts to generalize narrowing for higher-order functional-logic programs (see, for example, [5, 7, 9]) must address issues such as identifying suitable notions of *value* and *equality*, and reducing the huge search space for bindings of higher order variables.

The *Constructor-based ReWriting Logic* CRWL [4] provides a suitable framework for rule-based declarative (functional and logic) programming with non deterministic non-strict functions with *call-time choice* semantics, where programs are *Constructor-Based Conditional Term Rewrite Systems* (CB-CTRSs for short). Since the classical notion of rewriting is not suitable in this setting, a new notion of rewriting is adopted as the basis of *proof calculi* for joinability and reduction statements. An important result is the existence of sound and complete *lazy narrowing calculi* [4, 2, 3] for solving goals in first-order CRWL-theories presented by CB-CTRSs. Moreover, a higher-order extension of CRWL is presented in [5] but using applicative rewrite rules instead of λ-abstractions and higher-order unification.

In this paper, we propose a higher-order rewriting logic for declarative programming with higher-order functions and λ-terms as data structures to obtain more of the expressivity of higher-order functional programming. More precisely, we adopt the framework of simply typed λ-calculus in which terms are in $\beta\eta$-normal form and theories are presented by conditional overlapping inductively

* The author has been partially supported by the Spanish National Projects MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS-CAM (S-0505/TIC/0407).

sequential pattern rewrite systems. These are a subclass of fully-extended conditional pattern rewrite systems whose rules can be arranged in a variant of the classical *definitional trees*, a useful tool introduced by Antoy [1] for achieving a reduction strategy which avoids unneeded steps. Compared to CRWL, the distinctive features of our new higher-order conditional rewriting logic are:

- *values* are terms which do not match the left hand side of any rewrite rule. In the first-order case [4, 2], values are constructor-based terms, but this notion is too weak in a higher-order setting. Still, our notion of value is decidable because matching against a pattern is decidable.
- *equality* is interpreted as joinability to a common value.

We prove the existence of a sound and strongly complete lazy narrowing calculus for higher-order functional-logic programming in such a logic, generalizing and improving previous higher-order narrowing calculi [5, 9] and strategies [7]. Moreover, as the main novelty w.r.t. [7] and following the ideas introduced in [2, 3] for the first-order case, we show that definitional trees can be used to efficiently control the narrowing strategy in the setting of our higher-order rewriting logic.

The paper is structured as follows. In Section 2 we introduce the basic notions and notations of our theoretical framework. In Subsection 2.2 we propose a higher-order conditional rewriting logic characterized by a *proof system* called GHRC, a generalization of the proof system GORC which underlies the constructor-based rewriting logic CRWL of [4]. In Section 3 we propose a higher-order demand-driven narrowing calculus with definitional trees called HOLN$^{\mathrm{DT}}$. Section 4 is devoted to proving the main properties which guarantee the usefulness of HOLN$^{\mathrm{DT}}$ — soundness and strong completeness. Section 5 concludes.

2 Preliminaries

We assume the reader is familiar with the notions and notations pertaining to higher-order narrowing and term rewriting with definitional trees (see, e.g., [7] and [2]). The set of types for the simply typed λ-terms is generated by a set $\mathcal{B}$ of base types (e.g., `nat`, `bool`) and the function type constructor "$\rightarrow$". Simply typed λ-terms are generated in the usual way from a signature $\mathcal{F}$ of function symbols and a countably infinite set $\mathcal{V}$ of variables by successive operations of abstraction and application. We also consider the enhanced signature $\mathcal{F}_\bot = \mathcal{F} \cup \mathtt{Bot}$, where $\mathtt{Bot} = \{\bot_b \mid b \in \mathcal{B}\}$ is a set of distinguished $\mathcal{B}$-typed constants. The constant $\bot_b$ is intended to denote an undefined value of type b. We employ $\bot$ as a generic notation for a constant from `Bot`. In this paper, we assume the following conventions of notation: X, Y, Z, R, H, possibly primed or with subscripts, denote free variables; f, f' denote defined function symbols, g denotes a data constructor or defined function symbol, and a a (free or bound) variable or a constant from $\mathcal{F}$; l, r, s, t, u, possibly primed or with subscript, denote terms; $\pi, \pi', \pi_1, \pi_2, \ldots$ denote terms of base type.

A sequence of syntactic objects $o_1, \ldots, o_n$, where $n \geq 0$, is abbreviated by $\overline{o_n}$. For instance, the simply typed λ-term $\lambda x_1 \ldots \lambda x_k.(\cdots(a\ t_1)\ \cdots\ t_n)$ is abbreviated by $\lambda\overline{x_k}.a(\overline{t_n})$. Substitutions are finite type-preserving mappings from variables to terms, denoted by $\{\overline{X_n \mapsto t_n}\}$, and extend homomorphically from terms to terms. The set of free variables of a term t is denoted by $\mathcal{FV}(t)$.

The long $\beta\eta$-normal form of a term, denoted by $t\downarrow_\beta^{\uparrow\eta}$, is the η-expanded form of the β-normal form of t. It is well-known that $s =_{\alpha\beta\eta} t$ if $s\downarrow_\beta^{\uparrow\eta} =_\alpha t\downarrow_\beta^{\uparrow\eta}$ [8]. Since $\beta\eta$-normal forms are always defined, we will in general assume that terms are in long $\beta\eta$-normal form and are identified modulo α-conversion. For brevity, we may write variables and constants from $\mathcal{F}$ in η-normal form, e.g., X instead of $\lambda\overline{x_k}.X(\overline{x_k})$. We assume that the transformation into long $\beta\eta$-normal form is an implicit operation, e.g., when applying a substitution to a term. With these conventions, every term t has an unique long $\beta\eta$-normal form $\lambda\overline{x_k}.a(\overline{t_n})$, where $a \in \mathcal{F}_\perp \cup \mathcal{V}$ and $a()$ coincides with a. The symbol a is called the **root** of t and is denoted by $hd(t)$. We distinguish between the set $\mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ of **partial** terms (**terms** for short), the set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of **total** terms, and the set $\mathcal{T}(\mathcal{F}_\perp, \mathcal{V})^* = \mathcal{T}(\mathcal{F}_\perp, \mathcal{V}) \setminus \{\lambda\overline{x_k}.\perp \mid k \geq 0\}$ of partially defined terms. $\mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ is a poset with respect to the **approximation ordering** $\sqsubseteq$, defined as the least partial ordering such that:

$$\lambda\overline{x_k}.\perp \sqsubseteq \lambda\overline{x_k}.t \qquad t \sqsubseteq t \qquad \frac{s_1 \sqsubseteq t_1 \ \ldots \ s_n \sqsubseteq t_n}{\lambda\overline{x_k}.a(\overline{s_n}) \sqsubseteq \lambda\overline{x_k}.a(\overline{t_n})}$$

We adopt the convention that the free and bound variables inside a term are kept disjoint, and assume that bound variables with different binders have different names. We define $Dom(\gamma)=\{X\in\mathcal{FV} \mid X\gamma\neq X\}$. Two substitutions γ_1 and γ_2 are equal on a set W of variables, written as $\gamma_1 = \gamma_2$ $[W]$, iff $X\gamma_1=X\gamma_2$ for all $X\in W$, and we write $\gamma_1\leq\gamma_2$ $[W]$ iff there is a substitution σ with $\gamma_2 = \sigma\gamma_1$ $[W]$. The restriction $\gamma\restriction_W$ of a substitution γ to a set W of variables is defined by $X\gamma\restriction_W=X\gamma$ if $X\in W$ and $X\gamma\restriction_W=X$ otherwise. To manipulate terms, we define:

- the set of positions in t: $Pos(\lambda\overline{x_k}.a(\overline{t_n})) = \{1^i \mid 0 \leq i \leq k\} \cup \{1^k.j.q \mid 1 \leq j \leq n,\ q \in Pos(t_j)\}$, where "." denotes sequence concatenation and 1^k is the sequence of 1 repeated k times. The empty sequence is denoted by ϵ. Note that, with this convention, we have $1^0 = \epsilon$.
- the subterm $t|_p$ of t at some position $p \in Pos(t)$:

$$(\lambda\overline{x_k}.a(\overline{t_n}))|_p = \begin{cases} \lambda x_{i+1} \ldots x_k.a(\overline{t_n}) & \text{if } p = 1^i \text{ with } 0 \leq i \leq k, \\ t_i|_q & \text{if } p = 1^k.i.q \text{ and } 1 \leq i \leq n. \end{cases}$$

  A position p is maximal in t if $t|_p$ is of base type. The set of maximal positions in a term t is denoted by $MPos(t)$.
- the sequence of variables abstracted on the path to position $p \in Pos(t)$:

$$seq_{bv}(t,p) = \begin{cases} \epsilon & \text{if } p = \epsilon, \\ x.seq_{bv}(s,q) & \text{if } t = \lambda x.s \text{ and } p = 1.q, \\ seq_{bv}(t_i,q) & \text{if } t = a(\overline{t_n}),\ 0 < i \leq n, \text{ and } p = i.q. \end{cases}$$

  The set of variables abstracted on the path to position $p \in Pos(t)$ is $\mathcal{BV}(t,p) = \{seq_{bv}(t,p)\}$, and the set of variables with bound occurrences in t is $\mathcal{BV}(t)=\bigcup_{p\in Pos(t)} \mathcal{BV}(t,p)$. We also find it convenient to define $t\restriction_p=\lambda\overline{x_k}.(t|_p)$, where $\overline{x_k} = seq_{bv}(t,p)$.

A **pattern** [10] is a term t for which all subterms $t|_p = X(\overline{t_n})$, with $X \in \mathcal{FV}(t)$ and $p \in MPos(t)$, satisfy the condition that $t_1\downarrow_\eta, \ldots, t_n\downarrow_\eta$ is a sequence

of distinct elements of $\mathcal{BV}(t,p)$. Moreover, if all such subterms of t satisfy the additional condition $\mathcal{BV}(t,p) \setminus \{t_1{\downarrow_\eta}, \ldots, t_n{\downarrow_\eta}\} = \emptyset$, then the pattern t is **fully extended**.

An **equality statement** is a multiset $\{\!\{s,t\}\!\}$, written $s == \mathtt{t}$, where $s,t \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ are terms of the same type.

Definition 1. *A* **Conditional Pattern Rewrite System** *(***CPRS** *for short) is a finite set of conditional rewrite rules of the form $f(\overline{l_n}) \to r \Leftarrow C$, where*

- *$f(\overline{l_n})$ and r are total terms of the same base type,*
- *$f(\overline{l_n})$ is a fully extended linear pattern, and*
- *C is a (possibly empty) finite sequence of equality statements between total terms. In symbols, $C \equiv \overline{s_m == \mathtt{t_m}}$, with $s_i, t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for $i = 1, \ldots, m$.*

The term $f(\overline{l_n})$ is called the left hand side (lhs), r is the right hand side (rhs), and C is the conditional part of the rewrite rule.

A CPRS $\mathcal{R}$ induces a partition of $\mathcal{F}$ into $\mathcal{F}_d$ (*defined symbols*) and $\mathcal{F}_c$ (*constructors*):

$$\mathcal{F}_d = \{f \in \mathcal{F} \mid \exists (f(\overline{l_n}) \to r \Leftarrow C) \in \mathcal{R}\}, \quad \mathcal{F}_c = \mathcal{F} \setminus \mathcal{F}_d.$$

$\mathcal{R}$ is a **Constructor-Based** CPRS (**CB-CPRS** for short) if each rewrite rule $f(\overline{l_n}) \to r \Leftarrow C$ satisfies the additional condition that $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{F}_c, \mathcal{V})$.

Definition 2 (Lifter). *Given a term t, a subset V of $\mathcal{FV}(t)$, and a sequence $\overline{x_k}$ of distinct variables with no occurrences in t, the* **$\overline{x_k}$-lifter of t with respect to** *V is the term $t^{\uparrow \overline{x_k} \restriction V}$, defined recursively as follows:*

$$t^{\uparrow \overline{x_k} \restriction V} = \begin{cases} \lambda \overline{y_l}.(\pi^{\uparrow (\overline{y_l}, \overline{x_k}) \restriction V}) & \text{if } t \equiv \lambda \overline{y_l}.\pi, \\ a\left(t_n^{\uparrow \overline{x_k} \restriction V}\right) & \text{if } t \equiv a(\overline{t_n}) \text{ with } a \notin V, \\ X\left(\overline{x_k}, t_n^{\uparrow \overline{x_k} \restriction V}\right) & \text{if } t \equiv X(\overline{t_n}) \text{ with } X \in V. \end{cases}$$

The **$\overline{x_k}$-lifter of a term** *t is the term $t^{\uparrow \overline{x_k}} = t^{\uparrow \overline{x_k} \restriction \mathcal{FV}(t)}$. We also define $t^{\updownarrow \overline{x_k}} = \lambda \overline{x_k}.(t^{\uparrow \overline{x_k}})$.*

For example, $(\lambda x.Y(f(x, Z(x))))^{\uparrow y,z} = \lambda x.Y(y, z, f(x, Z(y, z, x)))$, whereas $(\lambda x.Y(x, Z(x)))^{\updownarrow y,z} = \lambda y, z, x.Y(y, z, x, Z(y, z, x))$. If $C = \overline{s_m == \mathtt{t_m}}$ is a sequence of equality statements then we write $C^{\updownarrow \overline{x_k}}$ for the sequence $\overline{s_m^{\updownarrow \overline{x_k}} == \mathtt{t_m}^{\updownarrow \overline{x_k}}}$.

For later use, we introduce the notation $\mathcal{R}_f$ for the subset of $\mathcal{R}$ consisting of the rewrite rules whose left-hand sides have head f.

It is well known that unification of patterns is decidable and unitary [10]. Therefore, for every $t \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ and pattern π, there exists at most one matcher between t and π, which we denote by $matcher(t, \pi)$.

A position $p \in MPos(t)$ is **rigid** in t if $hd(t|_q) \in \mathcal{BV}(t,q) \cup \mathcal{F}$ for all $q \leq p$. $p \in MPos(t)$ is **safe** in t if $hd(t|_q) \in \mathcal{BV}(t,q) \cup \mathcal{F}_c$ for all $q \leq p$. t is **flex** if $hd(t) \in \mathcal{FV}(t)$, and **rigid** otherwise. We denote by $Pos_r(t)$ the set of rigid positions of t, and by $Pos_s(t)$ the set of safe positions of t.

Example 1. If $t = \lambda x, y.g(f(x(X(a, y)), \perp))$, where $f \in \mathcal{F}_d$, $a, g \in \mathcal{F}_c$, and y is a bound variable of base type, then $MPos(t) = \{1^i \mid 2 \leq i \leq 6\} \cup \{1^5.2, 1.1.2\}$, $Pos_r(t) = \{1^2, 1^3, 1^4\}$, and $Pos_s(t) = \{1.1\}$. □

2.1 Higher-Order Overlapping Definitional Trees

The following definitions generalize the higher-order definitional trees introduced in [7]. There, neither conditional rewrite rules nor overlapping left hand sides were considered. The definitions also generalize Antoy's ODTs [1] and del Vado Vírseda's ODTs [2] for the first-order case.

Definition 3. *$\mathcal{T}$ is an* **Overlapping Definitional Tree** *(**ODT** for short) with fully extended linear pattern π iff $\mathcal{T}$ can be built in finitely many steps by using the following two construction rules:*

1. *$\mathcal{T} \equiv \underline{rule}(\pi, \{r_1 \Leftarrow C_1, \ldots, r_m \Leftarrow C_m\})$, abbreviated $\underline{rule}(\pi, \{r_i \Leftarrow C_i\}_{1 \leq i \leq m})$, where $\pi \to r_i \Leftarrow C_i$ are called the rewrite rules offered by $\mathcal{T}$.*
2. *$\mathcal{T} \equiv \underline{case}(\pi, p, \{\pi_1 : \mathcal{T}_1, \ldots, \pi_m : \mathcal{T}_m\})$, abbreviated $\underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1 \leq i \leq m})$, where $p \in MPos(\pi)$, $\pi|_p = X(\overline{y_n})$, each π_i is a term of the form $a_i(\overline{X_{q_i}})^{\uparrow \overline{y_n}}$, with $a_i \in \mathcal{F}_c \cup \{\overline{y_n}\}$ such that $a(\overline{X_{q_i}})^{\uparrow \overline{y_n}}$ is of the same type as X, $\overline{X_{q_i}}$ is a sequence of distinct fresh variables, $a_i \neq a_j$ whenever $i \neq j$, and each $\mathcal{T}_i$ is an ODT with linear pattern $\pi\{X \mapsto a(\overline{X_{q_i}})^{\uparrow \overline{y_n}}\}$.*

Let $\mathcal{R}[\mathcal{T}]$ be the set of all rewrite rules in $\mathcal{R}$ offered by the $\underline{rule}$-nodes of the ODT $\mathcal{T}$. A **call pattern** for an n-ary function symbol f is any fully extended linear pattern $f(\overline{t_n})$ such that $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}_c, \mathcal{V})$. Note that, if $\mathcal{T}$ is an ODT with pattern π, then π is a call pattern for some $f \in \mathcal{F}_d$. An **ODT for** $f \in \mathcal{F}_d$ is an ODT whose pattern is a call pattern for f.

Definition 4. *A CB-CPRS $\mathcal{R}$ is a* **Conditional Overlapping Inductively Sequential System** *(**COISS** for short) if every $f \in \mathcal{F}_d$ has an ODT $\mathcal{T}$ with pattern $f(\overline{X_n})$ such that $\mathcal{R}[\mathcal{T}] = \mathcal{R}_f$.*

Example 2. We consider the set of data constructors

$$\mathcal{F}_c = \{0 : \mathtt{nat}, \quad s : \mathtt{nat} \to \mathtt{nat}, \quad \mathtt{true}, \mathtt{false} : \mathtt{bool}\}$$

and the set of defined symbols

$$\mathcal{F}_d = \{f, g : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}, \quad \mathtt{leq} : (\mathtt{nat} \to \mathtt{nat}) \to (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{bool}\}$$

defined by the CB-CPRS

$$\begin{aligned}\mathcal{R} = \{\ & f(X,Y) \to s(0), \quad g(X,Y) \to 0,\\ & \mathtt{leq}(\lambda x.0, F) \to \mathtt{true},\\ & \mathtt{leq}(\lambda x.s(F(x)), \lambda x.0) \to \mathtt{false},\\ & \mathtt{leq}(\lambda x.s(F(x)), \lambda x.s(G(x))) \to \mathtt{leq}(\lambda x.F(x), \lambda x.G(x))\}.\end{aligned}$$

It is easy to check that $\mathcal{R}$ is a COISS. For example, the defined symbol `leq` has the ODT

$$\begin{aligned}\mathcal{T} = \underline{case}(\ & \mathtt{leq}(\lambda x.X(x), \lambda x.Y(x)), 1.1, \{\\ & 0 : \underline{rule}(\mathtt{leq}(\lambda x.0, \lambda x.Y(x)), \{\mathtt{true} \Leftarrow \{\}\}),\\ & s(F(x)) : \underline{case}(\ \mathtt{leq}(\lambda x.s(F(x)), \lambda x.Y(x)), 2.1, \{\\ & \qquad 0 : \underline{rule}(\mathtt{leq}(\lambda x.s(F(x)), \lambda x.0), \{\mathtt{false} \Leftarrow \{\}\}),\\ & \qquad s(G(x)) : \underline{rule}(\ \mathtt{leq}(\lambda x.s(F(x)), \lambda x.s(G(x))),\\ & \qquad\qquad \{\mathtt{leq}(\lambda x.F(x), \lambda x.G(x)) \Leftarrow \{\}\})\})\})\end{aligned}$$

and $\mathcal{R}[\mathcal{T}] = \mathcal{R}_{\mathtt{leq}}$. Note that the variables in the ODT have been written in long $\beta\eta$-normal form, in order to show that the ODT is built properly. □

We find it convenient to define the $\overline{x_k}$-lifter of an ODT as follows:

Definition 5. *The* $\overline{x_k}$**-lifter** $\mathcal{T}^{\uparrow \overline{x_k}}$ *of an ODT* $\mathcal{T}$ *is defined recursively as follows:*

- $\underline{rule}(\pi^{\uparrow \overline{x_k}}, \{r_i^{\uparrow \overline{x_k}} \Leftarrow C_i^{\downarrow \overline{x_k}}\}_{1 \leq i \leq m})$ *if* $\mathcal{T} \equiv \underline{rule}(\pi, \{r_i \Leftarrow C_i\}_{1 \leq i \leq m})$,
- $\underline{case}(\pi^{\uparrow \overline{x_k}}, p, \{\pi_i^{\uparrow \overline{x_k}} : \mathcal{T}_i^{\uparrow \overline{x_k}}\}_{1 \leq i \leq m})$ *if* $\mathcal{T} \equiv \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1 \leq i \leq m})$.

The $\overline{x_k}$*-lifted term* $\pi^{\uparrow \overline{x_k}}$ *is called the* **pattern of** $\mathcal{T}^{\uparrow \overline{x_k}}$ *and is denoted by* $patt(\mathcal{T}^{\uparrow \overline{x_k}})$*, and the position* p *is called the* **choice position** *of* $\mathcal{T}^{\uparrow \overline{x_k}}$ *and is denoted by* $Pos(\mathcal{T}^{\uparrow \overline{x_k}})$.

We note that, if $\mathcal{T} \equiv \underline{case}(\pi^{\uparrow \overline{x_k}}, p, \{\pi_i^{\uparrow \overline{x_k}} : \mathcal{T}_i^{\uparrow \overline{x_k}}\}_{1 \leq i \leq m})$, then the terms $\pi^{\uparrow \overline{x_k}}|_p$, $\pi_i^{\uparrow \overline{x_k}}$ and $patt(\mathcal{T}_i^{\uparrow \overline{x_k}})|_p$ are of the same base type, and $hd(\pi_i^{\uparrow \overline{x_k}}) = hd(patt(\mathcal{T}_i^{\uparrow \overline{x_k}})|_p)$.

From now on, assume we are given a COISS $\mathcal{R}$ together with a predefined association of an ODT $\mathcal{T}_f$, with $\mathcal{R}[\mathcal{T}_f] = \mathcal{R}_f$ to every $f \in \mathcal{F}_d$. We extend the notion $\mathcal{R}[\mathcal{T}]$ to $\overline{x_k}$-lifted ODTs by defining $\mathcal{R}[\mathcal{T}^{\uparrow \overline{x_k}}] = \mathcal{R}[\mathcal{T}]$. If $\mathcal{T}_1, \mathcal{T}_2$ are $\overline{x_k}$-lifted ODTs, then we write $\mathcal{T}_1 \ll \mathcal{T}_2$ if $\mathcal{T}_1$ is a proper subtree of $\mathcal{T}_2$.

Definition 6. *Let* $t \equiv \lambda \overline{x_k}.f(\overline{s_n})$ *and* $p = 1^k.q \in MPos(t)$. p *is a* **demanded position** *of* t *(i.e.,* $p \in dmd(t)$*) if there exists* $\mathcal{T} \ll \mathcal{T}_f^{\uparrow \overline{x_k}}$ *with* $Pos(\mathcal{T}) = q$*, and* $\forall \mathcal{T}'.\ \mathcal{T}' \ll \mathcal{T} \wedge q' = Pos(\mathcal{T}') \Rightarrow hd(f(\overline{s_n})|_{q'}) = hd(patt(\mathcal{T})|_{q'})$.

2.2 A Higher-Order Conditional Rewriting Logic

In this section, we propose a (conditional) higher-order rewriting logic for declarative programming with non-strict and non-deterministic functions with *call time choice* semantics, as a generalization of the first-order rewriting logic CRWL [4]. We propose this logic as the basis of a proof calculus, called GHRC, for reduction and joinability statements to a common value. The GHRC proof calculus provides a declarative semantic for a COISS $\mathcal{R}$, and will be used to prove the soundness and completeness properties of our narrowing calculus with definitional trees HOLN^{DT} in Section 4. First, we need to define the suitable notion of *value* that is used in our setting.

Definition 7. *A* **value** *is a partial term* t *which has the following property:*

$$\forall p \in MPos(t), \forall (\pi \to r \Leftarrow C) \in \mathcal{R} :\ \nexists matcher(t\restriction_p, \pi^{\updownarrow seq_{bv}(t,p)})^1$$

A **total value** *is a value which is a total term. A* **value substitution** *is a substitution which binds variables to values. We write* $Val(\mathcal{F}_\perp, \mathcal{V})$ *(resp.* $Val(\mathcal{F}, \mathcal{V})$*) for the set of values (resp. total values), and* $VSubst(\mathcal{F}_\perp, \mathcal{V})$ *for the set of substitutions which bind variables to values.*

For a given COISS $\mathcal{R}$, we want to derive statements of the following kind:

- *reduction* statements: $s \twoheadrightarrow t$, where $s, t \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ are of the same type.
- *equality* statements: $s == t$, which holds iff reduction statements $s \twoheadrightarrow u$ and $t \twoheadrightarrow u$ can be derived for some total value $u \in Val(\mathcal{F}, \mathcal{V})$.

The provability relation is defined by the following proof system:

[1] Note: in this definition, we assume $\mathcal{FV}(t) \cap \mathcal{FV}(\pi) = \emptyset$.

Definition 8 (GHRC proof calculus).

B *Bottom:* $\lambda\overline{x_k}.\pi \twoheadrightarrow \lambda\overline{x_k}.\bot$

MN *Monotonicity:* $$\frac{\lambda\overline{x_k}.s_1 \twoheadrightarrow \lambda\overline{x_k}.t_1 \ \ldots \ \lambda\overline{x_k}.s_n \twoheadrightarrow \lambda\overline{x_k}.t_n}{\lambda\overline{x_k}.a(\overline{s_n}) \twoheadrightarrow \lambda\overline{x_k}.a(\overline{t_n})}$$

RF *Reflexivity:* $s \twoheadrightarrow s$

OR *Outermost reduction:*

$$\frac{\lambda\overline{x_k}.s_1 \twoheadrightarrow l_1^{\downarrow\overline{x_k}}\theta \ \ldots \ \lambda\overline{x_k}.s_n \twoheadrightarrow l_n^{\downarrow\overline{x_k}}\theta \quad C^{\downarrow\overline{x_k}}\theta \quad r^{\downarrow\overline{x_k}}\theta \twoheadrightarrow u}{\lambda\overline{x_k}.f(\overline{s_n}) \twoheadrightarrow u}$$

for any $u \neq \lambda\overline{x_k}.\bot$, $\theta \in VSubst(\mathcal{F}_\bot, \mathcal{V})$, *and* $(f(\overline{l_n}) \to r \Leftarrow C) \in \mathcal{R}$.

J *Join:* $$\frac{s \twoheadrightarrow u \quad t \twoheadrightarrow u}{s == \mathtt{t}} \quad \text{if } u \in Val(\mathcal{F}, \mathcal{V}).$$

Thus, we interpret equality as joinability to a common total value. Detailed examples of derivations in the form of *proof trees* in this kind of rewriting logics can be found in [4] and [2]. We write $\mathcal{R} \vdash A$ if A is provable with GHRC for $\mathcal{R}$, and $\mathcal{R} \vdash \{\!\{A_1, \ldots, A_n\}\!\}$ if $\mathcal{R} \vdash A_i$ for $i = 1, \ldots, n$. We denote by $\mathcal{PT}(A)$ the set of proof trees for a statement A, $\mathcal{PT}_L(A)$ for the proof trees of $\mathcal{PT}(A)$ which end with the application of inference rule $L \in \{\mathbf{B}, \mathbf{MN}, \mathbf{RF}, \mathbf{OR}, \mathbf{J}\}$, and by $|\mathcal{P}|_{OR}$ the number of applications of **OR** in a proof tree $\mathcal{P}$. We also write $\mathcal{R} \vdash_L A$ if there exists a proof of $\mathcal{R} \vdash A$ which ends with application of rule L, and $\mathcal{R} \nvdash_L A$ if there is no such a proof. In the sequel, we will also consider the distinguished statement `false`, which denotes an unprovable statement.

In our study of GHRC it is relevant to tell more about the set from which we pick up the rewrite rule employed in an **OR** step. For this purpose, we assume in the sequel that Ω, Ω' range over sets of rewrite rules, and define the set $\mathcal{PT}^{\Omega}_{OR}(A)$ of proof trees $P \in \mathcal{PT}_{OR}(A)$, which employ a rewrite rule from Ω in the last proof step. We also write $\mathcal{R} \vdash^{\Omega}_{OR} A$ if there exists $P \in \mathcal{PT}^{\Omega}_{OR}(A)$, where Ω ranges over sets of rewrite rules of $\mathcal{R}$. Obviously, $\mathcal{PT}^{\Omega'}_{OR}(A) \subseteq \mathcal{PT}^{\Omega}_{OR}(A)$ whenever $\Omega' \subseteq \Omega$. We also define the set $\mathcal{PT}^{u}_{J}(s == \mathtt{t})$ of proof trees, which are of the form $\frac{\mathcal{P}_1 \ \mathcal{P}_2}{s==\mathtt{t}}(\mathrm{J})$, where $\mathcal{P}_1 \in \mathcal{PT}(s \twoheadrightarrow u)$ and $\mathcal{P}_2 \in \mathcal{PT}(t \twoheadrightarrow u)$.

In the remainder of this subsection we give two results, generalizing useful known properties of CRWL-deductions for the first-order case (see [4, 2] for more details), which characterize the semantics proofs built with GHRC and which are relevant to our further development of a demand-driven narrowing calculus.

Lemma 1 (Approximation Property). *Let* $s \in Val(\mathcal{F}_\bot, \mathcal{V})$. *If* $\mathcal{R} \vdash s \twoheadrightarrow t$ *then* $t \in Val(\mathcal{F}_\bot, \mathcal{V})$, $s \sqsupseteq t$, *and* $\mathcal{R} \nvdash_{\mathbf{OR}} s \twoheadrightarrow t$. *Moreover, if* $t \in Val(\mathcal{F}, \mathcal{V})$ *then* $s = t$.

Lemma 2 (Splitting Property). *If* $\mathcal{P} \in \mathcal{PT}^{\Omega}_{OR}(s \twoheadrightarrow t)$ *then* $|\mathcal{P}|_{OR} > 0$ *and* $hd(s) \in \mathcal{F}_d$. *Moreover, if* $p \in dmd(s)$ *and* $\overline{x_q} = seq_{bv}(s, p)$, *then either*

(i) $hd(s|_p) = g \in \mathcal{F}_d$ *and there exist* $\lambda\overline{x_q}.\pi \in \mathcal{T}(\mathcal{F}_\bot, \mathcal{V})^*$, $\mathcal{P}_1 \in \mathcal{PT}^{\mathcal{R}_g}_{OR}(\lambda\overline{x_q}.(s|_p) \twoheadrightarrow \lambda\overline{x_q}.\pi)$ *and* $\mathcal{P}_2 \in \mathcal{PT}^{\Omega}_{OR}(s[\pi]_p \twoheadrightarrow t)$ *such that* $|\mathcal{P}_1|_{OR} + |\mathcal{P}_2|_{OR} = |\mathcal{P}|_{OR}$, *or*

(ii) $hd(s|_p) \in \mathcal{F}_c \cup \{\overline{x_q}\}$ *and* $\mathcal{P} \in \mathcal{PT}^{\Omega'}_{OR}(s \twoheadrightarrow t)$, *where* $\Omega' = \{(\pi \to r \Leftarrow C) \in \Omega \mid p \in Pos_s(\pi)$ *and* $hd(\pi|_p) = hd(s|_p)\}$.

2.3 Goals and Solutions

Finally, we give a precise definition for the class of *goals* (from a given COISS $\mathcal{R}$) and the set of *solutions* of a goal with which we are going to work.

Definition 9. *An* **atomic goal** *is one of the following:*

- **equation:** *multiset* $\{\!\{s,t\}\!\}$ *of total terms of the same type, written* $s ==^? \mathtt{t}$.
- **annotated equation:** *pair* $\langle s ==^? \mathtt{t}, \mathtt{u}\rangle$ *between an equation* $s ==^? \mathtt{t}$ *and a total fully extended linear pattern* u *of the same type as* s. *We write such an annotated equation as* $s ==^?_{\mathtt{u}} \mathtt{t}$. *Equations are symmetric:* $s ==^? \mathtt{t} \equiv \mathtt{t} ==^? \mathtt{s}$ *and* $s ==^?_{\mathtt{u}} \mathtt{t} \equiv \mathtt{t} ==^?_{\mathtt{u}} \mathtt{s}$.
- **suspension:** *pair* $\langle s, R\rangle \in \mathit{Term}(\mathcal{F},\mathcal{V}) \times \mathcal{FV}$, *written* $s \rightarrowtail^? R$, *where* R *is a variable of the same type as* s.
- **production:** *ternary relation between a term* $\lambda\overline{x_k}.f(\overline{s_n})$ *with* $f \in \mathcal{F}_d$, *an* $\overline{x_k}$*-lifted ODT* $\mathcal{T}$ *for* f, *and a free variable* R *of the same type as* $\lambda\overline{x_k}.f(\overline{s_n})$, *written* $\lambda\overline{x_k}.\langle f(\overline{s_n}), \mathcal{T}\rangle \rightarrowtail^? R$.

or the distinguished symbol `fail`. *A* **goal** *is a multiset* $\{\!\{G_1,\ldots,G_n\}\!\}$ *of atomic goals* G_i.

In the sequel, we assume that w, w' denote either terms or expressions of the form $\lambda\overline{x_k}.\langle\pi,\mathcal{T}\rangle$, where $\mathcal{T}$ is an $\overline{x_k}$-lifted ODT for $hd(\pi)$. Then, $w \rightarrowtail^? R$ denotes either a suspension or a production. Similarly to other demand-driven or lazy narrowing calculi (see, e.g., [4] and [2]), we also need to define a suitable notion of *produced* and *demanded variable* to deal with a higher-order lazy evaluation.

Definition 10. *R is a* **produced variable** *in a goal G if* $\exists(w \rightarrowtail^? R) \in G$. *$H$ is a* **strict variable** *in G if* $\exists(s ==^?_{\mathtt{u}} \mathtt{t}) \in \mathtt{G}$ *and* $H \in \mathcal{FV}(u)$. *We write* $\mathcal{PV}(G)$ *(resp.* $\mathcal{SV}(G)$*) for the set of produced variables (resp. strict variables) in G. The set* $\mathcal{DV}(G)$ *of* **demanded variables** *in G is defined inductively as follows:* $R \in \mathcal{DV}(G)$ *if* $\exists(w \rightarrowtail^? R) \in G$, *and either*

(a) $\exists\ (\lambda\overline{y_k}.\langle\pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R') \in G$ *with* $hd(\pi'|_p) = R$, *or*
(b) $\exists\ (s ==^?_{\mathtt{u}} \mathtt{t}) \in \mathtt{G}$ *with* $hd(s) = R$, *or*
(c) $\exists\ (\lambda\overline{y_k}.R(\overline{t_n}) \rightarrowtail^? R') \in G$ *and* $R' \in \mathcal{DV}(G)$.

The set $\mathcal{SDV}(G)$ *of* **strongly demanded variables** *is defined by strengthening condition (b) of* $\mathcal{DV}(G)$ *to*

(b′) $\exists\ (s ==^?_{\mathtt{u}} \mathtt{t}) \in \mathtt{G}$ *with* $hd(s) = R$ *and* $u{\downarrow_\eta} \notin \mathcal{FV}$.

Note that $\mathcal{SDV}(G) \subseteq \mathcal{DV}(G) \subseteq \mathcal{PV}(G)$. Additionally, any goal G is **admissible** if the following conditions (called *goal invariants*) hold:

1. If $(w \rightarrowtail^? R) \in G$ then R occurs only once in the rhs of a production or suspension,
2. $R \notin \mathcal{FV}(w)$ for all suspension or production of G,
3. If $(\lambda\overline{x_k}.\langle\pi',\mathcal{T}\rangle \rightarrowtail^? R) \in G$ then $R \in \mathcal{DV}(G)$, and either
   (i) $\mathcal{T} \equiv \underline{rule}(\pi, \{r_i \Leftarrow C_i\}_{1\le i\le m})$ and $\lambda\overline{x_k}.\pi$ matches $\lambda\overline{x_k}.\pi'$, or
   (ii) $\mathcal{T} \equiv \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})$ and $1^k.p \in dmd(\lambda\overline{x_k}.\pi')$,

and if $s ==^?_{\mathtt{u}} \mathtt{t} \in \mathtt{G}$ then

4. If $H \in \mathcal{FV}(u)$ then H occurs only once in G,
5. If $u{\downarrow_\eta} \notin \mathcal{FV}$ then $hd(u) \in \{hd(s), hd(t)\}$.

Definition 11. *$\gamma \in Subst(\mathcal{F}_\perp, \mathcal{V})$ is a* **solution** *of a goal G if $\gamma{\restriction_{\mathcal{FV}(G)\setminus\mathcal{PV}(G)}} \in VSubst(\mathcal{F}_\perp, \mathcal{V})$, and for each atomic goal $G_i \in G$ there exists a proof tree $\mathcal{P}_i$ such that*

(a) $\mathcal{P}_i \in \mathcal{PT}(s\gamma == \mathtt{t}\gamma)$ *if* $G_i \equiv s ==^? \mathtt{t}$.
(b) $\mathcal{P}_i \in \mathcal{PT}(s\gamma \twoheadrightarrow R\gamma)$ *if* $G_i \equiv s \rightarrowtail^? R$.
(c) $\mathcal{P}_i \in \mathcal{PT}_J^{u\gamma}(s\gamma == \mathtt{t}\gamma)$ *if* $G_i \equiv s ==^?_{\mathtt{u}} \mathtt{t}$.
(d) $\mathcal{P}_i \in \mathcal{PT}_{OR}^{\mathcal{R}[\mathcal{T}]}(\lambda\overline{x_k}.\pi\gamma \twoheadrightarrow R\gamma)$ *if* $G_i \equiv \lambda\overline{x_k}.\langle \pi, \mathcal{T}\rangle \rightarrowtail^? R$.

The proof tree $\mathcal{P}_i$ is called a **witness** *that γ is a solution of G_i.*

We write $Soln(G)$ for the set of solutions of a goal G, and $Wtn_\gamma(G_i)$ for the set of witnesses that γ is a solution of an atomic goal G_i. The next result is useful to prove properties of our demand-driven narrowing calculus and shows that GHRC semantics does not accept an undefined value for demanded variables.

Lemma 3 (Demand Lemma). *Let $\gamma \in Soln(G)$. Then, $R\gamma \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})^*$ for all $R \in \mathcal{SDV}(G)$.*

Proof. Assume by contrary that there exists $R \in \mathcal{SDV}(G)$ with $hd(R\gamma) = \perp$. Then

1. there exists a (possibly empty) sequence of suspensions $\overline{s_q \twoheadrightarrow R_q}$ such that $R = R_0$ and $hd(s_i) = R_{i-1}$ for all $1 \leq i \leq q$, and
2. either (i) G contains $\lambda\overline{y_m}.\langle \pi', \underline{case}(\pi, p, \{\pi : \mathcal{T}_i\}_{1\leq i\leq m})\rangle \rightarrowtail^? X$ with $hd(\pi'|_p) = R_q$, or (ii) G contains a strict equation $s ==^?_{\mathtt{u}} \mathtt{t}$ with $hd(s) = R_q$ and $u = \lambda\overline{x_k}.a(\overline{H_n(\overline{x_k})})$ with $a \in \mathcal{F} \cup \{\overline{x_k}\}$.

From $\gamma \in Soln(G)$ and Lemma 1 results $hd(R_i\gamma) = \perp$ for all $i \leq q$. If 2.(i) holds then $1^m.p \in dmd(\lambda\overline{y_m}.\pi'\gamma)$ and $\mathcal{R} \vdash_{OR} \lambda\overline{y_m}.\pi'\gamma \twoheadrightarrow X\gamma$. In this case, we get a contradiction with Lemma 2, because $1^m.p \in dmd(\lambda\overline{y_m}.\pi'\gamma)$ and $hd(\pi'\gamma|_p) = \perp$. If 2.(ii) holds then $\mathcal{R} \vdash s\gamma \twoheadrightarrow u\gamma$. Then, $hd(s\gamma){=}\perp$ because $hd(s){=}R_q$ and $hd(R_q\gamma){=}\perp$. Since $\perp \neq a{=}hd(u\gamma)$, we can not have $s\gamma \twoheadrightarrow u\gamma$, contradiction. □

3 A Higher-Order Demand-driven Narrowing Calculus

HOLN$^{\mathrm{DT}}$ is a system of transformation rules designed to solve goals $\{\!\{s_1 ==^? \mathtt{t_1}, \ldots, s_n ==^? \mathtt{t_n}\}\!\}$, which we abbreviate by $\{\!\{\overline{s_n ==^? \mathtt{t_n}}\}\!\}$, where $s_i, t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all $i \in \{1, \ldots, n\}$. It acts on *states* of the form $P \equiv \langle G \mid \mathcal{K}\rangle$, where G is a goal and $\mathcal{K}$ is a set of values. A state P is **admissible** if

(a_1) G is admissible,
(a_2) $\mathcal{K}$ is a set of total pattern terms, and
(a_3) for every $s ==^?_{\mathtt{u}} \mathtt{t} \in \mathtt{G}$ there exists a pattern $t' \in \mathcal{K}$ and a maximal position $p \in MPos(t')$ such that $t' \restriction_p = u$.

We write $\mathtt{Adm}$ for the set of admissible states. The meaning of such a state is

$$[\![\langle G \mid \mathcal{K}\rangle]\!] = \{\gamma \in \mathit{Soln}(G) \mid \mathcal{K}\gamma \text{ is a set of values}\}.$$

We note that $[\![\langle \mathtt{fail} \mid \emptyset\rangle]\!] = [\![\langle G \mid \mathcal{K}\rangle]\!] = \emptyset$ whenever $\mathcal{K}$ is not a set of values. In the sequel, we denote the state $\langle \mathtt{fail} \mid \emptyset\rangle$ by $\mathtt{fail}$ and call it **failure state**. We identify with $\mathtt{fail}$ all pairs $\langle G \mid \mathcal{K}\rangle$ for which $\mathcal{K}$ is not a set of values. Solving a goal amounts to computing **refutations**, i.e., sequences of transformation steps.

Definition 12. *A* **HOLN$^{\mathbf{DT}}$-refutation** *of a goal* $G = \{\!\{\overline{s_n =^{?} \mathtt{t_n}}\}\!\}$ *is a maximal finite sequence of transformation steps*

$$\Pi : P_0 \equiv \langle G \mid \emptyset\rangle \equiv \langle G_0 \mid \mathcal{K}_0\rangle \Rightarrow_{\sigma_1} P_1 \equiv \langle G_1 \mid \mathcal{K}_1\rangle \Rightarrow_{\sigma_2} \ldots \Rightarrow_{\sigma_m} P_m \equiv \langle G_m \mid \mathcal{K}_m\rangle$$

between states $P_0, P_1, \ldots, P_m$*, such that* $P_m \neq \mathtt{fail}$ *is a* **final** *state, i.e., a non failure state which can not be transformed anymore. Each transformation step corresponds to an instance of some transformation rule of* HOLN^{DT}*. We abbreviate* Π *by* $P_0 \Rightarrow^*_\sigma P_m$*, where* $\sigma = \sigma_1 \cdots \sigma_m$*. Given such a goal* G*, the set of* **computed answers** *produced by* HOLN^{DT} *is*

$$\mathcal{A}(G) = \{\sigma\gamma\restriction_{\mathcal{FV}(G)} \mid \langle G \mid \emptyset\rangle \Rightarrow^*_\sigma P \text{ is a HOLN}^{DT}\text{-refutation and } \gamma \in [\![P]\!]\}.$$

3.1 Design and Analysis Considerations

In the sequel, we will describe the HOLN$^{\mathrm{DT}}$ calculus and analyze its main properties. The general idea is to ensure the computation of solutions from goals which are correct with respect to GHRC's semantics, while using definitional trees in a similar way to [2, 3] to ensure that all the narrowing steps performed during the computation are needed ones. Since the design considerations are quite involved and the analysis techniques quite complicated, we consider useful to precede our presentation with a brief outline of our design considerations and techniques.

Typical requirements in the design of such a calculus are **soundness**: every computed answer is a solution, i.e., $\mathcal{A}(G) \subseteq \mathit{Soln}(G)$, and **completeness**: for any $\gamma \in \mathit{Soln}(G)$ there exists $\gamma' \in \mathcal{A}(G)$ such that $\gamma' \leq \gamma\ [\mathcal{FV}(G)]$. Note that the completeness requirement demands the capability to compute a minimal complete set of solutions. It is easy to see that if HOLN$^{\mathrm{DT}}$ is complete then it suffices to enumerate minimal complete set of solutions of the final states. Therefore, an important design issue is to guarantee that minimal complete sets of solutions are easy to read off for the final states. In the design of first-order lazy or demand-driven narrowing calculi, as for example [4] and [2], this is achieved by ensuring that final states have empty goal components; thus the minimal complete set of solutions of a final state consists of the empty substitution $\{\}$. Unfortunately, things are much more complicated in the higher-order case. This problem is inevitably related to the problem of unifying flex terms, which is in general intractable. We adopt an approach similar to Huèt's procedure of higher-order pre-unification: we refrain from solving atomic goals between flex terms as much as possible. As a consequence, our final states will be a class of states whose equations are only between flex terms. We will show that it is possible to guarantee that these final states are meaningful and that it is relatively easy to read off some of their solutions.

Of particular importance is the following additional design issue: the transformation rules of HOLN$^{\mathrm{DT}}$ take into account the structure of the witness trees for the solutions we aim to compute. For example, if we encounter an atomic goal $\lambda\overline{x_k}.f(\overline{s_n}) =^?_{\mathtt{u}} \mathtt{t}$ with $f \in \mathcal{F}_d$, then we keep in mind that we are looking for a γ with a witness tree of the form $\frac{\mathcal{P}_1 \;\; \mathcal{P}_2}{\lambda\overline{x_k}.f(\overline{s_n})\gamma == \mathtt{t}\gamma}(\mathrm{J})$, where $\mathcal{P}_1 \in Wtn_\gamma(\lambda\overline{x_k}.f(\overline{s_n}) \rightarrowtail^? u)$.

To ease the presentation of the goal transformation rules of HOLN$^{\mathrm{DT}}$, we distinguish separately rules concerning the different components of an admissible goal: rules for (annotated) equations, rules for productions, rules for suspensions, and finally, rules for failure detection.

3.2 Transformation Rules for Equations

The goal transformation rules for (annotated) equations support an improved treatment of the strict equality $==^?$ as a built-in primitive symbol, along the lines of [4] and [2], rather than a defined function as in the case of [7]. We distinguish several cases according to the syntactic structure of their arguments.

(ann) **annotation**
$\langle \{\!\{ s ==^? \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow \langle \{\!\{ \mathtt{s} =^?_{\mathtt{H}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \cup \{\mathtt{H}\} \rangle$ where H is a fresh variable of suitable type.

(on)$_1$ **rigid narrowing**
$\langle \{\!\{ \lambda\overline{x_k}.f(\overline{s_n}) =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_{\{\}} \langle \{\!\{ \lambda\overline{\mathtt{x_k}}.\langle \mathtt{f}(\overline{\mathtt{s_n}}), \mathcal{T}_f^{\uparrow \overline{x_k}} \rangle \rightarrowtail^? \mathtt{R}, \mathtt{R} =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle$
where $f \in \mathcal{F}_d$ and either $hd(t) \notin \mathcal{PV}$ or $u{\downarrow_\eta} \in \mathcal{FV}$.

(ov)$_1$ **flex narrowing**
$\langle \{\!\{ \lambda\overline{x_k}.X(\overline{s_m}) =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \lambda\overline{x_k}.\langle X(\overline{s_m}), \mathcal{T}_f^{\uparrow \overline{x_k}} \rangle \rightarrowtail^? R, R =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\}\sigma \mid (\mathcal{K} \cup \{X\})\sigma \rangle$
where $u{\downarrow_\eta} \notin \mathcal{FV}$, $X \notin \mathcal{PV}$ and $\sigma = \{X \mapsto \lambda\overline{y_m}.f(\overline{X_n(\overline{y_m})})\}$ with $f \in \mathcal{F}_d$.

(sg) **strict guess**
$\langle \{\!\{ \lambda\overline{x_k}.a(\overline{s_n}) =^?_{\mathtt{H}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \lambda\overline{\mathtt{x_k}}.\mathtt{a}(\overline{\mathtt{s_n}}) =^?_{\mathtt{H}\sigma} \mathtt{t}, \mathtt{E} \mid \mathcal{K}\sigma \}\!\} \rangle$
where $a \in \mathcal{F} \cup \{\overline{x_k}\}$, and $\sigma = \{H \mapsto \lambda\overline{x_k}.a(\overline{H_n(\overline{x_k})})\}$.

(d) **decomposition**
$\langle \{\!\{ \lambda\overline{x_k}.v(\overline{s_n}) =^?_{\mathtt{u}} \lambda\overline{\mathtt{x_k}}.\mathtt{v}(\overline{\mathtt{t_n}}), \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \overline{\lambda\overline{\mathtt{x_k}}.\mathtt{s_n} =^?_{\mathtt{H_n}} \lambda\overline{\mathtt{x_k}}.\mathtt{t_n}}, \mathtt{E} \}\!\} \mid \mathcal{K}\sigma \rangle$
where $v \in \{\overline{x_k}\} \cup \mathcal{F}$ and either
- $u \equiv H$ and $\sigma = \{H \mapsto \lambda\overline{x_k}.v(\overline{H_n(\overline{x_k})})\}$, or
- $u \equiv \lambda\overline{x_k}.v(\overline{H_n(\overline{x_k})})$ and $\sigma = \{\}$.

(i)$_1$ **imitation**
$\langle \{\!\{ \lambda\overline{x_k}.X(\overline{s_p}) =^?_{\mathtt{u}} \lambda\overline{\mathtt{x_k}}.\mathtt{f}(\overline{\mathtt{t_n}}), \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \overline{\lambda\overline{\mathtt{x_k}}.\mathtt{X_n}(\overline{\mathtt{s_p}}) =^?_{\mathtt{H_n}} \lambda\overline{\mathtt{x_k}}.\mathtt{t_n}}, \mathtt{E} \}\!\}\sigma \mid (\mathcal{K} \cup \{\mathtt{X}\})\sigma \rangle$
where $X \notin \mathcal{PV}$ and either
- $u \equiv H$ and $\sigma = \{X \mapsto \lambda\overline{y_p}.f(\overline{X_n(\overline{y_p})}), H \mapsto \lambda\overline{x_k}.f(\overline{H_n(\overline{x_k})})\}$, or
- $u \equiv \lambda\overline{x_k}.f(\overline{H_n(\overline{x_k})})$ and $\sigma = \{X \mapsto \lambda\overline{y_p}.f(\overline{X_n(\overline{y_p})})\}$.

(p)$_1$ **projection**
$\langle \{\!\{ \lambda\overline{x_k}.X(\overline{s_p}) =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \lambda\overline{\mathtt{x_k}}.\mathtt{X}(\overline{\mathtt{s_p}}) =^?_{\mathtt{u}} \mathtt{t}, \mathtt{E} \}\!\}\sigma \mid (\mathcal{K} \cup \{\mathtt{X}\})\sigma \rangle$
where $X \notin \mathcal{PV}$, t is rigid, and $\sigma = \{X \mapsto \lambda\overline{y_p}.y_i(\overline{X_n(\overline{y_p})})\}$ is a valid projection binding.

(fs) **flex same**
$\langle \{\!\{ \lambda\overline{x_k}.X(\overline{y_p}) =^?_{\mathtt{H}} \lambda\overline{\mathtt{x_k}}.\mathtt{X}(\overline{\mathtt{y'_p}}), \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \mathtt{E} \}\!\}\sigma \mid (\mathcal{K} \cup \{\mathtt{X}\})\sigma \rangle$
where $X \notin \mathcal{PV}$, $\lambda\overline{x_k}.X(\overline{y_p})$ and $\lambda\overline{x_k}.X(\overline{y'_p})$ are patterns, $\sigma = \{X \mapsto \lambda\overline{y_p}.Z(\overline{z_q}),\ H \mapsto \lambda\overline{x_k}.Z(\overline{z_q})\}$, $\{\overline{z_q}\} = \{y_i \mid y_i = y'_i,\ 1 \leq i \leq n\}$.

(fd) **flex different**
$\langle \{\!\{ \lambda\overline{x_k}.X(\overline{y_p}) =^?_{\mathtt{H}} \lambda\overline{\mathtt{x_k}}.\mathtt{Y}(\overline{\mathtt{y'_q}}), \mathtt{E} \}\!\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \langle \{\!\{ \mathtt{E} \}\!\}\sigma \mid (\mathcal{K} \cup \{\mathtt{X}, \mathtt{Y}\})\sigma \rangle$
where $X, Y \notin \mathcal{PV}$, $\lambda\overline{x_k}.X(\overline{y_p})$ and $\lambda\overline{x_k}.Y(\overline{y'_q})$ are patterns, $X \neq Y$, $\sigma = \{X \mapsto \lambda\overline{y_p}.Z(\overline{z_r}), Y \mapsto \lambda\overline{y'_q}.Z(\overline{z_r}), H \mapsto \lambda\overline{x_k}.Z(\overline{z_r})\}$, and $\{\overline{z_r}\} = \{\overline{y_p}\} \cap \{\overline{y'_q}\}$.

3.3 Transformation Rules for Productions

The goal transformation rules for productions of the form $\lambda\overline{x_k}.\langle f(\overline{s_n}), \mathcal{T} \rangle \rightarrowtail^? R$ encode our higher-order demand-driven narrowing strategy, guided by the ODT $\mathcal{T}$ for the defined function symbol f, in a vein similar to the *needed narrowing strategy* of [7] and [2], and thanks to the *Splitting Property* given in Lemma 2.

(ev) **evaluation**

(1) $\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}}$
$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \mathcal{T}_i\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle$ if $hd(\pi'|_p) = hd(\pi_i)$.

(2) $\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{rule}(\pi, \{r_i \Leftarrow C_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}}$
$\langle \{\!\!\{ \overline{\lambda\overline{y_{k_q}}.s_q \rightarrowtail^? R_q}, C_i^?, \lambda\overline{x_k}.r_i \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K} \cup \{\overline{R_q}\}\rangle$
if $1 \le i \le m$, $matcher(\lambda\overline{x_k}.\pi', \lambda\overline{x_k}.\pi) = \{\overline{R_q \mapsto \lambda\overline{y_{k_q}}.s_q}\}$, and $\{\overline{R_q}\} = \mathcal{FV}(\lambda\overline{x_k}.\pi)$.

$(on)_2$ **rigid narrowing**

$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}}$
$\langle \{\!\!\{ \lambda\overline{x_q}.\langle \pi'|_p, \mathcal{T}_f^{\uparrow\overline{x_q}}\rangle \rightarrowtail^? R',$
$\lambda\overline{x_k}.\langle \pi'[R'(\overline{x_q})]_p, \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle$
if $\pi'|_p = f(\overline{t_n})$ with $f \in \mathcal{F}_d$, $\overline{x_q} = \mathcal{BV}(\lambda\overline{x_k}.\pi', 1^k.p)$.

$(ov)_2$ **flex narrowing**

$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma}$
$\langle \{\!\!\{ \lambda\overline{x_q}.\langle \pi'|_p, \mathcal{T}_f^{\uparrow\overline{x_q}}\rangle \rightarrowtail^? R',$
$\lambda\overline{x_k}.\langle \pi'[R'(\overline{x_q})]_p, \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\}\sigma \mid (\mathcal{K}\cup\{X\})\sigma\rangle$
if $\overline{x_q} = \mathcal{BV}(\lambda\overline{x_k}.\pi', 1^k.p)$, $hd(\pi'|_p) = X \notin \mathcal{PV}$, and $\sigma = \{X \mapsto \lambda\overline{y_r}.f(\overline{X_n(\overline{y_r})})\}$ for some $f \in \mathcal{F}_d$ of suitable type.

$(i)_2$ **imitation**

$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma}$
$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \mathcal{T}_i\rangle \rightarrowtail^? R, E\}\!\!\}\sigma \mid (\mathcal{K}\cup\{X\})\sigma\rangle$
if $hd(\pi'|_p) = X \notin \mathcal{PV}$, $hd(\pi_i) = c \in \mathcal{F}_c$, and $\sigma = \{X \mapsto \lambda\overline{y_r}.c(\overline{X_n(\overline{y_r})})\}$.

$(p)_2$ **projection**

$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma}$
$\langle \{\!\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\le i\le m})\rangle \rightarrowtail^? R, E\}\!\!\}\sigma \mid (\mathcal{K}\cup\{X\})\sigma\rangle$
if $hd(\pi'|_p) = X \notin \mathcal{PV}$ and $\sigma = \{X \mapsto \lambda\overline{z_r}.z_i(\overline{X_n(\overline{z_r})})\}$ is a valid projection binding for X.

3.4 Transformation Rules for Suspensions

The goal transformation rules concerning suspensions $s \rightarrowtail^? R$ are designed with the aim of modeling the behavior of lazy narrowing with *sharing*, as in other similar narrowing calculi [4, 2, 3], and thanks to the *Demand Lemma*.

(rm) **unnecessary suspension**

$\langle \{\!\!\{ t \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}} \langle \{\!\!\{ E\}\!\!\} \mid \mathcal{K}\rangle$ $(R \notin \mathcal{FV}(E))$

(rn) **rigid narrowing**

$\langle \{\!\!\{ \lambda\overline{x_k}.f(\overline{t_n}) \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}} \langle \{\!\!\{ \lambda\overline{x_k}.\langle f(\overline{t_n}), \mathcal{T}_f^{\uparrow\overline{x_k}}\rangle \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle$ $(R \in \mathcal{DV})$
if $f \in \mathcal{F}_d$.

$(i)_3$ **imitation**

$\langle \{\!\!\{ \lambda\overline{x_k}.f(\overline{s_n}) \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma} \langle \{\!\!\{ \overline{\lambda\overline{x_k}.s_n \rightarrowtail^? R_n}, E\sigma\}\!\!\} \mid \mathcal{K}\sigma\rangle$ $(R \in \mathcal{DV})$
where $\sigma = \{R \mapsto \lambda\overline{x_k}.f(\overline{R_n(\overline{x_k})})\}$.

(rp) **rigid projection**

$\langle \{\!\!\{ \lambda\overline{x_k}.x_i(\overline{t_n}) \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma} \langle \{\!\!\{ \overline{\lambda\overline{x_k}.t_n \rightarrowtail^? R_n}, E\sigma\}\!\!\} \mid \mathcal{K}\sigma\rangle$ $(R \in \mathcal{DV})$
where $\sigma = \{R \mapsto \lambda\overline{x_k}.x_i(\overline{R_n(\overline{x_k})})\}$.

(fp) **flex projection**

$\langle \{\!\!\{ \lambda\overline{x_k}.X(\overline{s_m}) \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma} \langle \{\!\!\{ \lambda\overline{x_k}.X(\overline{s_m}) \rightarrowtail^? R, E\}\!\!\}\sigma \mid (\mathcal{K}\cup\{X\})\sigma\rangle$ $(R \in \mathcal{SDV})$
where $X \notin \mathcal{PV}$ and $\sigma = \{X \mapsto \lambda\overline{y_m}.y_i(\overline{X_n(\overline{y_m})})\}$.

(fg) **flex guess**

$\langle \{\!\!\{ \lambda\overline{x_k}.X(\overline{s_m}) \rightarrowtail^? R, E\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\sigma} \langle \{\!\!\{ \overline{\lambda\overline{x_k}.X(\overline{s_m}) \rightarrowtail^? R}, E\}\!\!\}\sigma \mid (\mathcal{K}\cup\{X\})\sigma\rangle$ $(R \in \mathcal{SDV})$
where $X \notin \mathcal{PV}$ and $\sigma = \{X \mapsto \lambda\overline{y_m}.g(\overline{X_n(\overline{y_m})}), R \mapsto \lambda\overline{x_k}.g(\overline{R_n(\overline{x_k})})\}$ with $g \in \mathcal{F}$.

3.5 Transformation Rules for Failure Detection

Failure rules are used for failure detection in the syntactic unification of annotated equations and in the case of symbol non-cover by an ODT in productions.

(f_1) **clash 1**

$\langle \{\!\!\{ \lambda\overline{x_k}.v(\overline{s_n}) =^?_{\mathtt{u}} \lambda\overline{\mathtt{x_k}}.\mathtt{v}'(\overline{\mathtt{t_m}}), \mathtt{E}\}\!\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}} \mathtt{fail}$
if $v, v' \in \mathcal{F}_c \cup \{\overline{x_k}\}$ and either (i) $v \neq v'$ or (ii) $hd(u) \notin \mathcal{FV} \cup \{v, v'\}$.

(f_2) **clash 2**
$\langle \{\!\{ \lambda\overline{x_k}.\langle \pi', \underline{case}(\pi, p, \{\pi_i : \mathcal{T}_i\}_{1\leq i\leq m})\rangle \rightarrowtail^? R, E \}\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}} \texttt{fail}$
if $hd(\pi'|_p) \notin \{hd(\pi_i) \mid 1 \leq i \leq m\} \cup \mathcal{F}_d \cup \mathcal{FV}$.

(f_3) **occur check**
$\langle \{\!\{ \lambda\overline{x_k}.s =^?_{\mathtt{u}} \lambda\overline{\mathtt{x_k}}.\mathtt{X}(\overline{\mathtt{y_n}}), \mathtt{E} \}\!\} \mid \mathcal{K}\rangle \Rightarrow_{\{\}} \texttt{fail}$
if $X \notin \mathcal{PV}$, $\lambda\overline{x_k}.X(\overline{y_n})$ is a flex pattern, $hd(\lambda\overline{x_k}.s) \neq X$ and $(\lambda\overline{x_k}.s)|_p = X(\overline{z_n})$, where $\overline{z_n}$ is a sequence of distinct bound variables and $p \in Pos_s(\lambda\overline{x_k}.s)$.

Example 3. The classical higher-order function $foldr$ can be defined as

$$\begin{array}{ll} foldr(g, e, [\,]) & \rightarrow e \\ foldr(g, e, [x|xs]) & \rightarrow g(x, foldr(g, e, xs)) \end{array}$$

where the following result holds: if $f(e) = e'$ and $f(g(x, y)) = g'(x, f(y))$ then $f(foldr(g,\ e, xs)) = foldr(g', e', xs)$. Moreover, the following corollary can be easily obtained: if $f([\,]) = e'$ and $f([x|xs]) = g'(x,\ f(xs))$ then $f(xs) = foldr(g',\ e', xs)$. As application, we can use HOLN$^{\mathrm{DT}}$ over the particular function $f \rightarrow \lambda xs.\ pair(sum(xs),\ length(xs))$, where $pair$ is a data constructor, and

$$\begin{array}{lll} sum([\,]) \rightarrow 0 & length([\,]) \rightarrow 0 & fst(pair(x, y)) \rightarrow x \\ sum([x|xs]) \rightarrow x + sum(xs) & lenght([x|xs]) \rightarrow 1 + length(xs) & snd(pair(x, y)) \rightarrow y \end{array}$$

to compute E' and G' s.t. $f([\,]) == E'$ and $\lambda x, xs.f([x|xs]) == \lambda x, xs.G'(x,\ f(xs))$. We obtain the solutions $\{E' \mapsto pair(0, 0)\}$ and $\{G' \mapsto \lambda u, z.pair(u + fst(z), 1 + snd(z))\}$. For example, we have the following HOLN$^{\mathrm{DT}}$-refutation corresponding to $\{E' \mapsto pair(0, 0)\} \in \mathcal{A}(\{\!\{ f([\,]) ==^? E' \}\!\})$:

$\langle \{\!\{ f([\,]) ==^? \mathtt{E'} \}\!\} \mid \emptyset\rangle \Rightarrow (ann)$
$\langle \{\!\{ f([\,]) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow_{\{\}} (on)_1$
$\langle \{\!\{ \lambda xs.\langle f([\,]), \mathcal{T}_f\rangle \rightarrowtail^? R, R ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow_{\{\}} (ev)(2)$
$\langle \{\!\{ pair(sum([\,]), length([\,])) \rightarrowtail^? R, R ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow_{\{R\ \mapsto\ pair(R_1,R_2)\}} (i)_3$
$\langle \{\!\{ sum([\,]) \rightarrowtail^? R_1, length([\,]) \rightarrowtail^? R_2, pair(R_1, R_2) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow^2_{\{\}} (rn)$
$\langle \{\!\{ \lambda x, xs.\langle sum([\,]), \mathcal{T}_{sum}\rangle \rightarrowtail^? R_1, \lambda x, xs.\langle length([\,]), \mathcal{T}_{length}\rangle \rightarrowtail^? R_2,$
$\quad pair(R_1, R_2) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow^2_{\{\}} (ev)(1)$
$\langle \{\!\{ \langle sum([\,]), \mathcal{T}_{sum([\,])}\rangle \rightarrowtail^? R_1, \langle length([\,]), \mathcal{T}_{length([\,])}\rangle \rightarrowtail^? R_2,$
$\quad pair(R_1, R_2) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow^2_{\{\}} (ev)(2)$
$\langle \{\!\{ 0 \rightarrowtail^? R_1, 0 \rightarrowtail^? R_2, pair(R_1, R_2) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow^2_{\{R_1\mapsto 0, R_2\mapsto 0\}} (i)_3$
$\langle \{\!\{ pair(0, 0) ==^?_{\mathtt{H}} \mathtt{E'} \}\!\} \mid \{\mathtt{H}\}\rangle \Rightarrow_{\{E'\mapsto pair(X,Y), H\mapsto pair(H_1,H_2)\}} (i)_1$
$\langle \{\!\{ X ==^?_{\mathtt{H_1}} \mathtt{0}, \mathtt{Y} ==^?_{\mathtt{H_2}} \mathtt{0} \}\!\} \mid \{\mathtt{pair(H_1, H_2), pair(X, Y)}\}\rangle \Rightarrow^2_{\{X\mapsto 0, Y\mapsto 0, H_1\mapsto 0, H_2\mapsto 0\}} (i)_1$
$\langle \{\!\{\ \}\!\} \mid \{pair(0, 0), pair(0, 0), 0, 0\}\rangle$. □

4 Main Properties

The main properties of the calculus relate the solutions of a goal to the answers computed by HOLN$^{\mathrm{DT}}$. First, we analyze how much local information about $Soln(G)$ is carried by $\mathcal{A}(G)$ to prove correctness of a single HOLN$^{\mathrm{DT}}$-step.

Lemma 4 (Local Soundness). *If $P \in \mathtt{Adm}$ and $P \Rightarrow_\sigma P'$ is a HOLN^{DT}-step then $P' \in \mathtt{Adm}$ and $\{\sigma\gamma \mid \gamma \in [\![P']\!]\} \subseteq [\![P]\!]$. Moreover, if $P \in \mathtt{Adm}$ satisfies the preconditions of a transformation rule for failure detection, then $[\![P]\!] = \emptyset$.*

This property corresponds to "narrowing" the set of possible computed answers by each transformation step. Now, the *soundness* is quite easy to achieve from Lemma 4: we must check that each inference step in a derivation is locally sound.

Theorem 1 (Soundness). *Let* $\Pi : \langle G \mid \emptyset \rangle \Rightarrow^*_\sigma P'$ *be a* HOLN^{DT}*-derivation. Then,* $\sigma\gamma \in Soln(G)$ *whenever* $\gamma \in [\![P']\!]$.

Proof. Let $\gamma \in [\![P']\!]$. We prove by induction on the length of Π that $\sigma\gamma \in Soln(G)$. If $|\Pi| = 0$ then $\sigma = \{\}$ and $\sigma\gamma = \gamma \in [\![P']\!] = [\![\langle G \mid \emptyset \rangle]\!] = Soln(G)$. If $|\Pi| > 0$ then we can write $\Pi : \langle G \mid \emptyset \rangle \Rightarrow^*_{\sigma_1} P_1 \Rightarrow_{\sigma_2} P'$. By Lemma 4, we know that $\sigma_2\gamma \in [\![P_1]\!]$. We can now apply the IH to the shorter derivation $\langle G \mid \emptyset \rangle \Rightarrow^*_{\sigma_1} P$ and learn that $\sigma\gamma = \sigma_1\sigma_2\gamma \in [\![\langle G \mid \emptyset \rangle]\!] = Soln(G)$. □

Completeness is much more difficult to ensure: we must verify that *any* solution γ of a given goal G will be eventually *approximated*, i.e., that we will eventually reach a representation $S_{fin} = \langle G' \mid \mathcal{K}' \rangle$ of a computed answer γ' such that $\gamma' \leq \gamma\ [\mathcal{FV}(G)]$. This approximation process must take into account all the possible shapes of elementary goals, and make sure that *progress* can be made towards reaching S_{fin}. We achieve this by looking at the syntactic structures of atomic goals, the solution γ which we want to approximate, and the witness tree that γ is a solution of the given goal, and show how these grouped structures or triples (called *configurations* of a set of admissible configurations `Cfg`) can be looked up for computing a representation S_{fin} of an approximation of γ by means of a *well-founded ordering* $\succ$ over `Cfg`.

Lemma 5 (Progress). *There exists a poset* $(\texttt{Cfg}, \succ)$ *with* $\succ$ *well-founded, and a surjection* $\Im : \texttt{Cfg} \to \texttt{Adm}$*, such that, if* $P = \langle G \mid \mathcal{K} \rangle \in \texttt{Adm}$ *is a non-final state,* W *is a finite set of variables, and* $\gamma \in [\![P]\!]$ *with* $Dom(\gamma) \subseteq \mathcal{V}$*, then there exist* $P' = \langle G' \mid \mathcal{K}' \rangle \in \texttt{Adm}$*,* $\gamma' \in [\![P']\!]$*, and a* HOLN^{DT}*-step* $P \Rightarrow_\sigma P'$*, with* $\Im(P) \succ \Im(P')$ *and* $\gamma = \sigma\gamma'\ [W]$.

We are ready now to state our completeness result by application of Lemma 5.

Theorem 2 (Strong Completeness). *Let* $G = \{\!\{\overline{s_n =^? \mathtt{t_n}}\}\!\}$*. Then,* $\mathcal{A}(G) = \{\gamma\restriction_{\mathcal{FV}(G)} \mid \gamma \in Soln(G)\}$.

Proof. Since $\mathcal{A}(G) \subseteq Soln(G)$ by soundness (Theorem 1), we must only show that $Soln(G) \subseteq \mathcal{A}(G)$. Let $\gamma \in Soln(G)$, $P = \langle G \mid \emptyset \rangle$, and $W_0 = \mathcal{FV}(G) \cup Dom(\gamma)$. First, we prove that, for every given admissible state P, any finite set of variables W, and $\gamma \in [\![P]\!]$, there exists a HOLN$^{\mathrm{DT}}$-refutation $\Phi : P \Rightarrow^*_\sigma P'$ such that $\gamma = \sigma\gamma'\ [W]$ for some $\gamma' \in [\![P']\!]$. The proof is by induction with respect to the well-founded ordering $\succ$ introduced in Lemma 5.

If P is final then we can choose $\Phi : P \Rightarrow^0_{\{\}} P$ and $\gamma' = \gamma$. Otherwise, we can apply Lemma 5 to determine $P_1 = \langle G_1 \mid \mathcal{K}_1 \rangle$, $\gamma_1 \in [\![P_1]\!]$, and a HOLN$^{\mathrm{DT}}$-step $\varphi : P \Rightarrow_{\sigma_1} P_1$ with $\Im(P) \succ \Im(P_1)$ and $\gamma = \sigma_1\gamma_1\ [W]$. Let $W' = W \cup \mathcal{FV}(\{X\sigma_1 \mid X \in W\})$. By IH for $\Im(P_1)$, there exists a HOLN$^{\mathrm{DT}}$-refutation $\Phi' : P_1 \Rightarrow^*_{\sigma'} P'$ such that $\gamma_1 = \sigma'\gamma'\restriction_{W'}$ for some $\gamma' \in [\![P']\!]$. Let $\sigma = \sigma_1\sigma'$ and Φ the HOLN$^{\mathrm{DT}}$-refutation obtained by prepending φ to Φ'. Then, $\Phi : P \Rightarrow^*_\sigma P'$ and $\sigma\gamma'$ is a computed

answer. Also, $\gamma\restriction_W = \sigma_1\gamma_1\restriction_W = \sigma_1\sigma'\gamma'\restriction_W = \sigma\gamma'\restriction_W$, and this concludes our preliminary proof. In particular, if $\gamma \in \mathit{Soln}(G)$ then $\gamma \in [\![P]\!]$ where $P = \langle G \mid \emptyset \rangle$. According to our preliminary result, there exists a HOLN$^{\mathrm{DT}}$-refutation $\Phi : P \Rightarrow^*_\sigma P'$ such that $\gamma = \sigma\gamma'\ [\mathcal{FV}(G)]$ for some $\gamma' \in [\![P']\!]$. Thus, $\sigma\gamma'\restriction_{\mathcal{FV}(G)} \in \mathcal{A}(G)$, $\sigma\gamma'\restriction_{\mathcal{FV}(G)} = \gamma\restriction_{\mathcal{FV}(G)}$, and $\gamma \in \mathcal{A}(G)$. □

Our proof reveals that HOLN$^{\mathrm{DT}}$ is *strongly complete*, i.e., completeness does not depend on the choice of the selectable atomic goal in the current state.

5 Conclusion and Future Work

We have presented a generalization of the first-order Constructor-based ReWriting Logic CRWL [4] to the more expressive setting of the simply typed λ-calculus, in order to define a higher-order demand-driven narrowing calculus HOLN$^{\mathrm{DT}}$ for higher-order functional-logic programming. We have proved that HOLN$^{\mathrm{DT}}$ conserves the good properties of the needed narrowing strategy [7], while being sound and strongly complete w.r.t. our higher-order conditional rewriting logic. Higher-order overlapping definitional trees are used to efficiently control the narrowing strategy for answering joinability and reducibility queries, extending and generalizing the first-order case [2], which guarantee the usefulness of HOLN$^{\mathrm{DT}}$.

Because of these results, we plan to implement the HOLN$^{\mathrm{DT}}$ calculus in *Mathematica* [6], a good framework for working with rewrite systems and narrowing. We hope that the implemented calculus serves as a functional-logic programming languages interpreter, allowing researchers in this field write higher-order functional-logic programs combined with powerful *Mathematica* rewrite rules.

References

1. S. Antoy. Optimal non-deterministic functional logic computations. In *ALP'97*, pages 16–30, 1997.
2. R. del Vado Vírseda. A demand-driven narrowing calculus with overlapping definitional trees. In *PPDP*, pages 253–263, 2003.
3. R. del Vado Vírseda. Declarative constraint programming with definitional trees. In *FroCos*, pages 184–199, 2005.
4. J. C. González-Moreno, M. T. Hortalá-González, F. J. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *J. Log. Program.*, 40(1):47–87, 1999.
5. J. C. González-Moreno, M. T. Hortalá-González, and M. Rodríguez-Artalejo. A higher order rewriting logic for functional logic programming. In *ICLP*, pages 153–167, 1997.
6. M. Hamada and T. Ida. Implementation of lazy narrowing calculi in mathematica. Technical report, RISC, Johannes Kepler University, Austria, 1997.
7. M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. *J. Funct. Program.*, 9(1):33–75, 1999.
8. J. R. Hindley and J. P. Seldin. *Introduction to Combinatorics and λ-Calculus.* Cambridge University Press, 1986.
9. T. Ida, M. Marin, and T. Suzuki. Higher-order lazy narrowing calculus: A solver for higher-order equations. In *EUROCAST*, pages 479–493, 2001.
10. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.