A Demand-driven Narrowing Calculus with Overlapping Definitional Trees

Rafael del Vado Vírseda
Dpto. Sistemas Informát. y Prog.
Universidad Complutense de Madrid
Av.Complutense,28040Madrid,Spain
rdelvado@sip.ucm.es

ABSTRACT

We propose a *demand-driven conditional narrowing calculus* in which a variant of definitional trees [2] is used to efficiently control the narrowing strategy. This calculus is sound and strongly complete w.r.t. *Constructor-based ReWriting Logic* (CRWL) semantics [7] for a wide class of constructor-based conditional term rewriting systems. The calculus mantains the optimality properties of the needed narrowing strategy [5]. Moreover, the treatment of *strict equality* as a primitive rather than a defined function symbol, leads to an improved behaviour w.r.t. needed narrowing.

Categories and Subject Descriptors

D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; D.1.6 [**Programming Techniques**]: Logic Programming; D.3.3 [**Programming Languages**]: Language Constructs and Features – *Control structures*; D.3.4 [**Programming Languages**]: Processors – *Optimization*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; G.2.2 [**Discrete Mathematics**]: Graph Theory – *Trees*; I.1.1 [**Algebraic Manipulation**]: Expressions and Their Representation – *Simplification of expressions*; I.2.2 [**Automatic Programming**]: Program transformation.

General Terms

Algorithms, Languages, Performance, Theory.

Keywords

Functional Logic Programming Languages, Rewrite Systems, Narrowing, call-time choice, definitional trees.

PPDP'03, August 27-29, 2003, Uppsala, Sweden.

1. INTRODUCTION

Many recent approaches to the integration of functional and logic programming take *constructor-based conditional rewrite systems* as programs and some *lazy narrowing strategy* or *calculus* as a goal solving mechanism and as operational semantics (see [8] for a detailed survey). On the other hand, *non-deterministic* functions are an essential feature of these integrated languages, since they allow problem solving using programs that are textually shorter, easier to understand and maintain, and more declarative than their deterministic counterparts (see [3], [7] or [13] for several motivating examples).

In this context, efficiency has been one of the major drawbacks of the functional logic programming paradigm, where the introduction of non-deterministic computations often generates huge search spaces with their associated overheads both in terms of time and space. Actually, the adoption of a *demand-driven* narrowing strategy can result in a large reduction of the search space [15]. A demand-driven strategy evaluates an argument of a function only if its value is needed to compute the result. An appropriate notion of *need* is a subtle point in the presence of a non-deterministic choice, e.g., a typical computation step, since the need of an argument might depend on the choice itself. For the case of rewriting, an adequate theory of neededness has been proposed by Huet and Lévy [12] for orthogonal and unconditional rewrite systems and extended by Middeldorp [17] to the more general case of neededness for the computation of root-stable forms. The so-called *needed narrowing* strategy [5] has been designed for *Inductively Sequential Systems* (ISS), a proper subclass of orthogonal rewriting systems which coincides with the class of strongly sequential constructor-based orthogonal TRSs [11] and defines only deterministic functions by means of unconditional rewrite rules.

Both needed narrowing and the demand driven strategy from [15] were originally defined with the help of *definitional trees*, a tool introduced by Antoy [2] for achieving a reduction strategy which avoids unneeded reductions. The needed narrowing strategy is sound and complete for the class ISS, and it enjoys interesting and useful optimality properties, but has the disadvantage that it refers only to a closed version of *strict equality* [5] where the goals are often solved by enumerating infinitely many ground solutions, instead of computing a single more general one.

Example 1. Consider the following ISS, and let $s^k(t)$ be the result of applying the constructor s, k consecutive times, to the term t.

$$X + 0 \rightarrow X$$
$$X + s(Y) \rightarrow s(X + Y)$$

For the goal X + Y == Z, needed narrowing as presented in [5] enumerates infinitely many ground solutions $\{Y \mapsto s(0), X \mapsto s^n(0), Z \mapsto s^{n+1}(0)\}$ (for all $n \geq 0$) instead of computing an 'open' solution $\{Y \mapsto s(0), Z \mapsto s(X)\}$ which subsumes the previous ground ones. The reason of this behaviour is the formal treatment of == as a function defined by rewrite rules. In this example:

$$0 == 0 \rightarrow \text{true} \qquad \text{true}, Z \rightarrow Z$$
$$s(X) == s(Y) \rightarrow X == Y \qquad \square$$

A more recent paper [3] has proposed an extension of needed narrowing called *inductively sequential narrowing*. This strategy is sound and complete for *overlapping inductively sequential systems*, a proper extension of the class of ISSs which permits non-deterministic functions, but conditional rules and an 'open' strict equality are not adopted. The optimality properties of needed narrowing are retained in a weaker sense by inductively sequential narrowing. Even more recently, [10] has proposed a reformulation of needed narrowing as a goal transformation system, as well as an extension of its scope to a class of higher-order inductively sequential TRSs. Higher-order needed narrowing in the sense of [10] is sound and relatively complete, and it enjoys optimality properties similar to those known for the first-order case.

In spite of the good results from the viewpoint of neededness, [5, 2, 10] stick to a simplistic treatment of strict equality as a defined function symbol. Other recent works [14, 18, 7] have developed various *lazy narrowing calculi* based on *goal transformation* systems, where an *open semantics* of strict equality does not require the computation of ground solutions, as illustrated by Example 1. Moreover, [7] allows non-deterministic functions with so-called *call-time choice* semantics, borrowed from [13], and illustrated by the next example:

Example 2. Consider the TRS consisting of the rewrite rules for + given in Example 1 and the following rules:

$$coin \rightarrow 0 \qquad double(X) \rightarrow X + X$$
$$coin \rightarrow s(0)$$

Intuitively, the constant function *coin* represents a non-deterministic choice. In order to evaluate the expression *double(coin)*, *call-time choice* semantics chooses some possible value for the argument expresion *coin*, and then applies the function *double* to that value. Therefore, when using *call-time choice*, *double(coin)* can be reduced to 0 and $s^2(0)$, but not to $s(0)$. □

As argued in [7, 13], the *call-time choice* view of non-determinism is the convenient one for many programming applications. However, the narrowing methods proposed in [5, 2, 14, 18] are unsound w.r.t. *call-time choice* semantics. In [7], the *Constructor-based ReWriting Logic CRWL* is proposed to formalize rewriting with *call-time* non-deterministic choices, and the *Constructor-based Lazy Narrowing Calculus CLNC* is provided as a sound and strongly complete goal solving method. Therefore, CLNC embodies the advantages of open strict equality and *call-time choice* non-determinism.

The aim of this paper is to refine CLNC in order to enrich its advantages with those of needed narrowing. We will combine goal transformation rules similar to those in [14, 18, 7] with the use of definitional trees to guide the selection of narrowing positions, in the vein of [15, 5, 3, 10]. As a result, we will propose a *Demand-driven Narrowing Calculus DNC* which can be proved sound and strongly complete w.r.t. CRWL's semantics (*call-time choice* non-determinism and open strict equality), contracts needed redexes, and maintains the good properties shown for needed narrowing in [5, 3]. All these properties qualify DNC as a convenient and efficiently implementable operational model for lazy FLP languages.

The organization of this paper is as follows: Section 2 contains some technical preliminaries regarding the constructor-based rewriting logic CRWL. Section 3 defines the subclass of rewriting systems used as programs in this work. In Section 4 we give a formal presentation of the calculus DNC. We discuss the soundness, completeness and optimality results in Section 5. Finally, some conclusions and plans for future work are drawn in Section 6.

2. PRELIMINARIES

The reader is assumed to be familiar with the basic notions and notations of term rewriting, as presented e.g. in [6].

We assume a *signature* $\Sigma = DC_\Sigma \cup FS_\Sigma$ where $DC_\Sigma = \bigcup_{n\in\mathbb{N}} DC_\Sigma^n$ is a set of ranked *constructor* symbols and $FS_\Sigma = \bigcup_{n\in\mathbb{N}} FS_\Sigma^n$ is a set of ranked *function* symbols, all of them with associated *arity* and such that $DC_\Sigma \cap FS_\Sigma = \emptyset$. We also assume a countable set $\mathcal{V}$ of *variable* symbols. We write $Term_\Sigma$ for the set of (total) *terms* built over Σ and $\mathcal{V}$ in the usual way, and we distinguish the subset $CTerm_\Sigma$ of (total) constructor terms or (total) *c-terms*, which only make use of DC_Σ and $\mathcal{V}$. The subindex Σ will usually be omitted. Terms intend to represent possibly reducible expressions, while c-terms represent data values, not further reducible. The CRWL semantics also uses the constant symbol $\bot$, that plays the role of the *undefined* value. We define $\Sigma_\bot = \Sigma \cup \{\bot\}$; the sets $Term_\bot$ and $CTerm_\bot$ of (partial) terms and (partial) c-terms respectively, are defined in a natural way, by using the symbol $\bot$ just as a constant for syntatic purposes. Partial c-terms represent the result of partially evaluated expressions; thus, they can be seen as approximations to the value of expressions in the CRWL semantics. As usual notations we will write X,Y,Z for variables, *c*,*d* for constructor symbols, *f*,*g* for functions, *a*,*b*,*e* for terms and *s*,*t* for c-terms. Moreover, *Var(e)* will be used for the set of variables occurring in the term *e*. A term *e* is called *ground* term, if $Var(e) = \emptyset$. A term is called *linear* if it is does not contain multiple occurrences of any variable. A natural *approximation ordering* $\sqsubseteq$ over $Term_\bot$ can be defined as the least partial ordering over $Term_\bot$ satisfying the following properties: $\bot \sqsubseteq e$ for all $e \in Term_\bot$, and $h(e_1,...,e_n) \sqsubseteq h(e_1',...,e_n')$, if $e_i \sqsubseteq e_i'$ for all $i \in \{1,\ldots,n\}$, $h \in DC^n \cup FS^n$.

To manipulate terms we give the following definitions. An *occurrence* or *position* is a sequence p of positive integers identifying a subterm in a term. For every term e, the set $O(e)$ of *positions* in e is inductively defined as follows: the empty sequence denoted by ε, identifies e itself. For every term of the form $f(e_1,...,e_n)$, the sequence $i{\cdot}q$, where i is a positive integer not greater than n and q is a position, identifies the subterm of e_i at q. The subterm of e at p is denoted by $e|_p$ and the result of *replacing* $e|_p$ with e' in e is denoted by $e[e']_p$. If p and q are positions, we write $p \preccurlyeq q$ if p is above or is a *prefix* of q, and we write $p \parallel q$ if the positions are *disjoint*, i.e. neither $p \preccurlyeq q$ nor $q \preccurlyeq p$. The expression $p{\cdot}q$ denotes the position resulting from the concatenation of the positions p and q. The set $O(e)$ is partitioned into $\tilde{O}(e)$ and $O_{\mathscr{V}}(e)$ as follows: $\tilde{O}(e) = \{p \in O(e) \mid e|_p \notin \mathscr{V}\}$ and $O_{\mathscr{V}}(e) = \{p \in O(e) \mid e|_p \in \mathscr{V}\}$. If e is a linear term, *pos(X,e)* will be used for the position of the variable X occurring in e. The notation $Var_c(e)$ stands for the set of all variables ocurring in e at some position whose ancestor positions are all occupied by constructors.

The sets of substitutions *CSubst* and $CSubst_\perp$ are defined as applications σ from $\mathscr{V}$ into CTerm and $\text{CTerm}_\perp$ respectively such its *domain* $\text{Dom}(\sigma) = \{X \in \mathscr{V} \mid \sigma(X) \neq X\}$ is finite. We will write θ,σ,μ for substitutions and {} for the *identity* substitution. Substitutions are extended to morphisms on terms by $\sigma(f(e_1,...,e_n)) = f(\sigma(e_1),...,\sigma(e_n))$ for every term $f(e_1,...,e_n)$. The *composition* of two substitutions θ and σ is defined by $(\theta{\circ}\sigma)(X) = \theta(\sigma(X))$ for all $X \in \mathscr{V}$. A substitution σ is *idempotent* iff $\sigma{\circ}\sigma = \sigma$. We frequently write $\{X_1 \mapsto e_1,\ldots,X_n \mapsto e_n\}$ for the substitution that maps X_1 into e_1, …, X_n into e_n. The *restriction* $\sigma\restriction_V$ of a substitution σ to a set $V \subseteq \mathscr{V}$, is defined by $\sigma\restriction_V(X) = \sigma(X)$ if $X \in V$ and $\sigma\restriction_V(X) = X$ if $X \notin V$. A substitution σ is *more general* than σ', denoted by $\sigma \lesssim \sigma'$, if there is a substitution τ with $\sigma' = \tau{\circ}\sigma$. If V is a set of variables, we write $\sigma = \sigma'[V]$ iff $\sigma\restriction_V = \sigma'\restriction_V$, and we write $\sigma \lesssim \sigma'[V]$ iff there is a substitution τ with $\sigma' = \tau{\circ}\sigma[V]$. A term e' is an *instance* of e if there is a substitution σ with $e' = \sigma(e)$. In this case we write $e \preceq e'$. A term e' is a *variant* of e if $e \preceq e'$ and $e' \preceq e$.

Given a signature Σ, a *CRWL-program* $\mathfrak{R}$ is defined as a set of conditional rewrite rules of the form $f(t_1,...,t_n) \to r \Leftarrow a_1 == b_1, ..., a_m == b_m$ ($m \geq 0$) with $f \in FS^n$, $t_1,...,t_n \in \text{CTerm}$, $r,a_i,b_i \in \text{Term}$, $i = 1,...,m$, and $f(t_1,...,t_n)$ is a linear term. The term $f(t_1,...,t_n)$ is called *left hand side* and r is the *right hand side*.

Example 3. For instance, we consider the following CRWL-program defining a non-deterministic function *bits*, where we use the function *coin* from Example 2 and the Prolog syntax for the list constructors (i.e. [ ] denotes the empty list and [·|·] denotes a non-empty list consisting of a first element and a remaining list).

$$bits \to [\,X \mid bits\,] \Leftarrow coin == X$$

bits generates an infinite list with a non-deterministic choice of each element (0 or $s(0)$). □

From a CRWL-program we are interested in deriving sentences of two kinds: *approximation statements* $a \to b$, with $a \in \text{Term}_\perp$ and $b \in \text{CTerm}_\perp$, and *joinability statements* $a == b$, with $a, b \in \text{Term}_\perp$. We formally define a *goal-oriented rewriting calculus* CRWL [7] by means of the following inference rules:

BT *Bottom*: $e \to \perp$ for any $e \in \text{Term}_\perp$.

RR *Restricted Reflexivity*: $X \to X$ for any variable X.

DC *DeComposition:*

$$\frac{e_1 \to t_1 \;\ldots\; e_n \to t_n}{c(e_1,...,e_n) \to c(t_1,...,t_n)}$$

for $c \in DC^n$, $t_i \in \text{CTerm}_\perp$.

OR *Outer Reduction*:

$$\frac{e_1 \to t_1 \;\ldots\; e_n \to t_n \quad \text{C} \quad r \to t}{f(e_1,...,e_n) \to t}$$

if $t \neq \perp$ and $(f(t_1,...,t_n) \to r \Leftarrow \text{C}) \in [\mathfrak{R}]_\perp$ where $[\mathfrak{R}]_\perp = \{\theta(l \to r \Leftarrow \text{C}) \mid (l \to r \Leftarrow \text{C}) \in \mathfrak{R}, \theta \in \text{CSubst}_\perp\}$ is the set of all partial c-instances of rewrite rules in $\mathfrak{R}$.

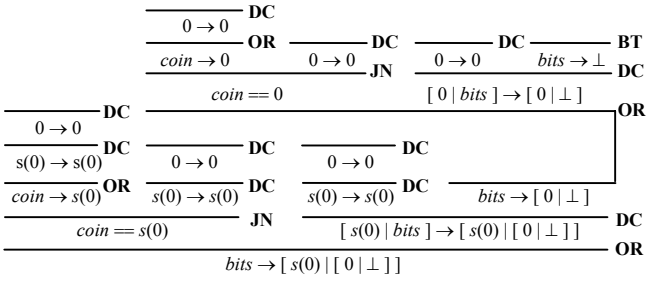
JN *Join*

$$\frac{a \to t \qquad b \to t}{a == b}$$

for total $t \in \text{CTerm}$.

Rule **BT** corresponds to an implicit rewrite rule $X \to \perp$, meaning that ⊥ approximates the value of any expression. Rules **RR**, **DC** and **OR** implicitly encode reflexivity, monotonicity and transitivity of the rewriting relation, while rule **JN** is used to establish the validity of conditions in the case of conditional rewrite rules. The rule **OR** uses *partial data substitutions* θ to instantiate rewrite rules from $\mathfrak{R}$, while classical rewriting would use arbitrary substitutions of *total expressions* for variables. It is because of this difference that CRWL expresses *call-time choice* semantics. Moreover, **BT** and **OR** (in cooperation with the other inference rules) also ensure that CRWL expresses a non-strict semantics for function evaluation.

Example 4. As a concrete example of CRWL-derivability, we show a CRWL-proof for the approximation statement $bits \to [s(0)|[0|\perp]]$, using the CRWL-program of Example 3.



□

In the sequel we will use the notations $\mathfrak{R} \vdash_{\text{CRWL}} \varphi$ to indicate that φ (an approximation statement or a joinability statement) can be deduced from the CRWL-program $\mathfrak{R}$ by finite application of the above rules. In the following lemma we collect some simple facts about provable statements involving c-terms, which are needed for the proofs of the main results in this paper. The proof is straighforward by induction over the structure of c-terms or over the structure of CRWL-proofs (see [7] and [19] for details).

Lemma 1 (basic properties of CRWL-deductions). *Let $\mathcal{R}$ be a CRWL-program. For any $e \in Term_\perp$, $t,s \in CTerm_\perp$ and $\theta,\theta' \in CSubst_\perp$, we have:*

a) $\mathcal{R} \mid\!-_{CRWL} t \to s \Leftrightarrow t \sqsupseteq s$. *Furthermore, if $s \in CTerm$ is total, then $t \sqsupseteq s$ can be replaced by $t = s$.*

b) $\mathcal{R} \mid\!-_{CRWL} e \to t,\ t \sqsupseteq s \Rightarrow \mathcal{R} \mid\!-_{CRWL} e \to s$.

c) $\mathcal{R} \mid\!-_{CRWL} \theta(e) \to t,\ \theta \sqsubseteq \theta' \Rightarrow \mathcal{R} \mid\!-_{CRWL} \theta'(e) \to t$, *with the same length and structure in both deductions.*

d) $\mathcal{R} \mid\!-_{CRWL} e \to s \Rightarrow \mathcal{R} \mid\!-_{CRWL} \theta(e) \to \theta(s)$, *if θ is a total substitution.*

e) $\mathcal{R} \mid\!-_{CRWL} e \to s \Leftrightarrow$ *there exists* $t' \in CTerm_\perp$ *such that* $\mathcal{R} \mid\!-_{CRWL} e|_p \to t'$ *and* $\mathcal{R} \mid\!-_{CRWL} e[t']_p \to s$, *provided that* $p \in O(e)$.

3. OVERLAPPING DEFINITIONAL TREE AND COISS

The demand-driven narrowing calculus that we are going to present works for a class of CRWL-programs in which conditional rewrite rules with possibly overlapping left hand sides must be organized in a hierarchical structure called *definitional tree* [2]. More precisely, we choose to reformulate the notion of *overlapping definitional tree* and *overlapping inductively sequential system* introduced in [3], including now conditional rules.

Definition 1 (ODT). *Let $\Sigma = DC \cup FS$ be a signature and $\mathcal{R}$ a CRWL-program over Σ. A* **pattern** *is any linear term of the form $f(t_1,...,t_n)$, where $f \in FS^n$ is a function symbol and t_i are c-terms. $\mathfrak{I}$ is an* **Overlapping Definitional Tree** *(**ODT** for short) with pattern π iff its depth is finite and one of the following cases holds:*

- *$\mathfrak{I} = \underline{rule}(l \to r_1 \Leftarrow C_1 \mid ... \mid r_m \Leftarrow C_m)$, where $l \to r_i \Leftarrow C_i$ for all i in $\{1,...,m\}$ is a variant of a rule in $\mathcal{R}$ such that $l = \pi$.*
- *$\mathfrak{I} = \underline{case}(\pi, X, [\mathfrak{I}_1,...,\mathfrak{I}_k])$, where X is a variable in π, $c_1,...,c_k \in DC$ for some $k > 0$ are pairwise different constructors of $\mathcal{R}$, and for all i in $\{1,...,k\}$, $\mathfrak{I}_i$ is an ODT with pattern $\sigma_i(\pi)$, where $\sigma_i = \{X \mapsto c_i(Y_1,...,Y_{m_i})\}$, m_i is the arity of c_i and $Y_1,...,Y_{m_i}$ are new distinct variables.*

We represent an ODT $\mathfrak{I}$ with pattern π using the notation $\mathfrak{I}_\pi$. An **ODT of a function symbol** $f \in FS^n$ defined by $\mathcal{R}$ is an ODT $\mathfrak{I}$ with pattern $f(X_1,...,X_n)$, where $X_1,...,X_n$ are new distinct variables.

Definition 2 (COISS). *A function f is called* **overlapping inductively sequential** *w.r.t. a CRWL-program $\mathcal{R}$ iff there exists an ODT $\mathfrak{I}_f$ of f such that the collection of all the rewrite rules $l \to r_i \Leftarrow C_i$ $(1 \le i \le m)$ obtained from the different nodes $\underline{rule}(l \to r_1 \Leftarrow C_1 \mid ... \mid r_m \Leftarrow C_m)$ occurring in $\mathfrak{I}_f$ equals, up to variants, the collection of all the rewrite rules in $\mathcal{R}$ whose left hand side has the root symbol f. A CRWL-program $\mathcal{R}$ is called* **Conditional Overlapping Inductively Sequential System** *(**COISS**, for short) iff each function defined by $\mathcal{R}$ is overlapping inductively sequential.*

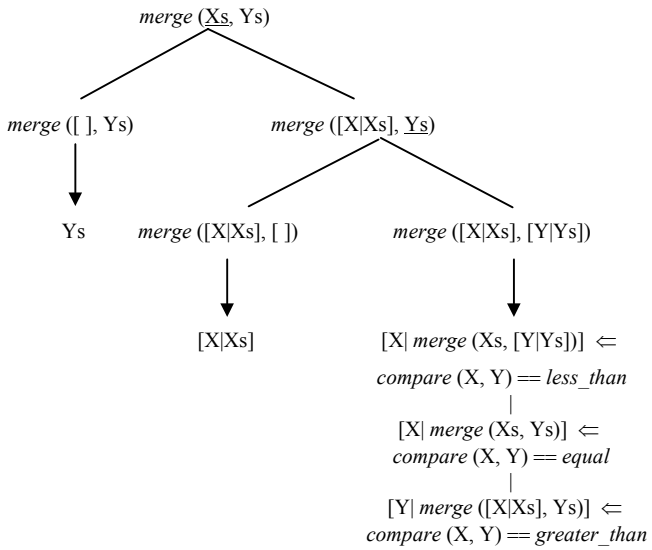
Example 5. The following CRWL-program $\mathfrak{R}$ defines a function *merge* that merges two given lists. We use Prolog's syntax for the list constructors and we omit the rewrite rules needed for defining the function *compare*.

```
merge ([ ], Ys)       → Ys
merge ([X|Xs], [ ])   → [X|Xs]
merge ([X|Xs], [Y|Ys])
      → [X| merge (Xs, [Y|Ys])]  ⇐ compare (X, Y) == less_than
      → [X| merge (Xs, Ys)]      ⇐ compare (X, Y) == equal
      → [Y| merge ([X|Xs], Ys)]  ⇐ compare (X, Y) == greater_than
```

since the defined symbol *merge* has the following ODT, *merge* is overlapping inductively sequential w.r.t. $\mathfrak{R}$.

```
case(merge (Xs, Ys), Xs, [
 rule(merge ([ ]      , Ys) → Ys),
 case(merge ([X|Xs], Ys), Ys, [
   rule(merge ([X|Xs], [ ]) → [X|Xs]),
   rule(merge ([X|Xs], [Y|Ys]) →
      [X| merge (Xs, [Y|Ys])] ⇐ compare (X, Y) == less_than  |
      [X| merge (Xs, Ys)]     ⇐ compare (X, Y) == equal      |
      [Y| merge ([X|Xs], Ys)] ⇐ compare (X, Y) == greater_than)] )])
```

This tree can be illustrated by the following picture:



□

Let $\mathfrak{R}$ be any COISS. For our discussion of a demand-driven narrowing calculus with ODTs over COISSs we are going to use a presentation similar to that of the CLNC calculus (see [7]). So, goals for $\mathfrak{R}$ are essentially finite conjunctions of CRWL-statements, and solutions are c-substitutions such that the goal affected by the substitution becomes CRWL-provable. The precise definition of admissible goal includes a number of technical conditions which will be defined and explained below. The general idea is to ensure the computation of solutions which are correct w.r.t. CRWL's *call-time choice* semantics, while using ODTs in a similar way to [5, 3, 10] to ensure that all the narrowing steps performed during the computation are needed ones.

4. THE DNC CALCULUS

In this section we present a *Demand-driven Narrowing Calculus* (shortly, *DNC*) for goal solving. We give first a precise definition for the class of admissible goals and solutions we are going to work with.

Definition 3 (Admissible goals). *A* **goal** *for a given COISS $\mathcal{R}$ must have the form $G \equiv \exists U.\ S \square P \square E$, where the symbols $\square$ and ',' must be interpreted as conjunction, and:*

- *$EVar(G) =_{def} U$ is the set of so-called* **existential variables** *of the goal G. These are intermediate variables, whose bindings in a solution may be partial c-terms.*
- *$S \equiv X_1 \approx s_1, \ldots, X_n \approx s_n$ is a set of equations, called* **solved part**. *Each s_i must be a total c-term, and each X_i must occur exactly once in the whole goal. This represents an idempotent c-substitution $\sigma_S = \{X_1 \mapsto s_1, \ldots, X_n \mapsto s_n\}$.*
- *$P \equiv t_1 \to R_1, \ldots, t_k \to R_k$ is a multiset of* **productions** *where each R_i is a variable and t_i is a term or a pair of the form $\langle \pi, \mathfrak{I} \rangle$, where π is an instance of the pattern in the root of an ODT $\mathfrak{I}$. Those productions $l \to R$ whose left hand side l is simply a term are called* **suspensions**, *while those whose left hand side is of the form $\langle \pi, \mathfrak{I} \rangle$ are called* **demanded productions**. *$PVar(P) =_{def} \{R_1,\ldots,R_k\}$ is called the set of* **produced variables** *of the goal G. The* **production relation** *between G-variables is defined by $X \gg_P Y$ iff there is some $1 \le i \le k$ such that $X \in Var(t_i)$ and $Y = R_i$ (if t_i is a pair $\langle \pi, \mathfrak{I} \rangle$, $X \in Var(t_i)$ must be interpreted as $X \in Var(\pi)$).*
- *$E \equiv l_1 == r_1, \ldots, l_m == r_m$ is a multiset of* **strict equations**. *$DVar(P \square E)$, called the set of* **demanded variables** *of the goal G, is the least subset of $Var(P \square E)$ which fulfills the following conditions:*
  - *$(l == r) \in E \Rightarrow Var_C(l) \cup Var_C(r) \subseteq DVar(P \square E)$.*
  - *$(\langle \pi, \underline{case}(\pi, X, [\mathfrak{I}_1,\ldots,\mathfrak{I}_q]) \rangle \to R) \in P$ with $t|_{pos(X,\pi)} = Y \in Var(t)$ and $R \in DVar(P \square E) \Rightarrow Y \in DVar(P \square E)$.*

Additionally, an **admissible goal** *must satisfy the following properties, called* **admissibility conditions**:

LIN *Each produced variable is produced only once, i.e. variables $R_1,\ldots,R_k$ must be different.*

EX *All the produced variables must be existential, i.e. $PVar(P) \subseteq EVar(G)$.*

CYC *The transitive closure of the production relation $\gg_P$ must be irreflexive, or equivalently, a strict partial order.*

SOL *The solved part S contains no produced variables.*

DT *For each demanded production $\langle \pi, \mathfrak{I} \rangle \to R$ in P, the variable R is demanded (i.e. $R \in DVar(P \square E)$), and the variables in $\mathfrak{I}$ do not occur in other place of the goal.*

Admissibility conditions are natural in so far they are satisfied by all the goals arising from *initial goals* $G_0 \equiv \exists \varnothing.\ \varnothing \square \varnothing \square E$ by means of the goal transformations presented below. We note that an equivalent formulation for the admissibility condition **CYC** can be obtained by requiring that it is possible to reorganize the productions of P in the form $P \equiv t_1 \to R_1, \ldots, t_k \to R_k$ where $R_i \notin \bigcup_{1 \le j \le i-1} Var(t_j)$ for all $1 \le i \le n$ (if t_j is a pair $\langle \pi, \mathfrak{I} \rangle$, $Var(t_j)$ must be interpreted as $Var(\pi)$). On the other hand, the following distinction between the two possible kinds of productions is useful:

- *Demanded productions $\langle t, \mathfrak{I} \rangle \to R$* are used to compute a value for the demanded variable R. The value will be shared by all occurrences of R in the goal, in order to respect *call-time choice* semantics. Note that t is always an instance of the pattern in the root of $\mathfrak{I}$. Moreover, if $\mathfrak{I} = \underline{case}(\pi, X, [\mathfrak{I}_1,\ldots,\mathfrak{I}_k])$, $p = pos(X,\pi)$ must be a position in t such that all symbols in t at positions above p are data constructors.
- *Suspensions $f(t_1,\ldots,t_n) \to R$* eventually become demanded productions if R becomes demanded, or else disappear if R becomes absent from the rest of the goal. See the goal transformations **SA** and **EL** in Subsection 4.2. Suspensions $X \to R$ and $c(e_1,\ldots,e_n) \to R$ are suitably treated by other goal transformations.

Definition 4 (Solutions). *Let $G \equiv \exists U.\ S \square P \square E$ be an admissible goal for a COISS $\mathcal{R}$, and θ a partial c-substitution. We say that θ is a* **solution** *for G with respect to $\mathcal{R}$ if:*

- *For each $X \notin PVar(P)$, $\theta(X)$ is a total c-term.*
- *For each solved equation $(X_i \approx s_i) \in S$, $\theta(X_i) = \theta(s_i)$.*
- *For each suspension $(t \to R) \in P$, $\mathcal{R} \mid\!-_{CRWL} \theta(t) \to \theta(R)$.*
- *For each demanded production $(\langle \pi, \mathfrak{I} \rangle \to R) \in P$, $\mathcal{R} \mid\!-_{CRWL} \theta(\pi) \to \theta(R)$ (i.e. the definitional tree $\mathfrak{I}$ is only used to control the computation).*
- *For each strict equation $(l == r) \in E$, $\mathcal{R} \mid\!-_{CRWL} \theta(l) == \theta(r)$.*

A **witness** *$\mathcal{M}$ for the fact that θ is a solution of G is defined as a multiset containing the CRWL proofs mentioned above. We write Sol(G) for the set of all solutions for G.*

The next result is useful to prove properties about DNC and shows that CRWL semantics does not accept an undefined value for demanded variables.

Lemma 2 (Demand lemma). *If θ is a solution of an admissible goal $G \equiv \exists U.\ S \square P \square E$ for a COISS $\mathcal{R}$ and $X \in DVar(P \square E)$, then $\theta(X) \neq \bot$.*

Proof. Let $G \equiv \exists U.\ S \square P \square E$ be an admissible goal for a COISS $\mathcal{R}$ and $X \in DVar(P \square E)$. We use induction on the transitive closure $\gg_P^+$ of the production relation, which is a well founded ordering, due to the admissibility condition **CYC**. We consider the two cases for X.

a) $X \in Var_C(l) \cup Var_C(r)$ with $(l == r) \in E$.

We suppose that $X \in Var_C(l)$ (analogous if $X \in Var_C(r)$). There are a finite number $k \geq 0$ of constructor symbols above X in l. Since θ is a solution of G, $\Re \mid\!\!-_{CRWL} \theta(l) == \theta(r)$ and exists $u \in CTerm$ with $\Re \mid\!\!-_{CRWL} \theta(l) \to u$ and $\Re \mid\!\!-_{CRWL} \theta(r) \to u$ using the CRWL-rule **JN**. Since $\Re \mid\!\!-_{CRWL} \theta(l) \to u$, where $\theta \in CSubst_\perp$ and u is a total c-term, it is easy to see that u and $\theta(l)$ must have the same constructors at all the positions in the path from the root to the position of X in l. So, the CRWL-deduction takes the rule **DC** k times on $\theta(l) \to u$ to obtain $\Re \mid\!\!-_{CRWL} \theta(X) \to t$ with $t \in CTerm$. Then $\theta(X) = t$ follows from Lemma 1 a), and this entails $\theta(X) \neq \perp$.

b) $X = t|_{pos(Y,\pi)}$ with $(\langle t, \underline{case}(\pi, Y, [\mathfrak{I}_1,...,\mathfrak{I}_k])\rangle \to R) \in P$ and $R \in DVar(P \square E)$.

In this case, $t = f(t_1,...,t_n)$ with $f \in FS^n$, and since θ is a solution, $\Re \mid\!\!-_{CRWL} f(\theta(t_1),...,\theta(t_n)) \to \theta(R)$. By Definition 3, $X \gg_P R$ and hence $X \gg_P^+ R$. Since $R \in DVar(P \square E)$, the induction hypothesis yields $\theta(R) \neq \perp$. Moreover, $pos(Y,\pi) = i \cdot p$ with $1 \leq i \leq n$ and $p \in O(t_i)$ because $\pi \preceq t$. Therefore, the CRWL-deduction $\Re \mid\!\!-_{CRWL} f(\theta(t_1),...,\theta(t_n)) \to \theta(R)$ must be of the form **OR**:

$$\frac{\theta(t_1) \to t_1' \ \ldots \ \theta(t_i) \to t_i' \ \ldots \ \theta(t_n) \to t_n' \quad C \quad r \to \theta(R)}{f(\theta(t_1),...,\theta(t_n)) \to \theta(R)}$$

where $(f(t_1',...,t_i',...,t_n') \to r \Leftarrow C) \in [\Re]_\perp$. Due to the form of the ODT in the demanded production, t_i' has a constructor symbol c_j $(1 \leq j \leq k)$ in the position p. Moreover, there must be only constructor symbols above c_j in t_i'. So, $t_i' \neq \perp$. Moreover, since $\Re \mid\!\!-_{CRWL} \theta(t_i) \to t_i'$ where $t_i' \in CTerm_\perp$ and $\theta \in CSubst_\perp$, there must be the same constructor symbols and in the same order above $\theta(X)$ in the position p of $\theta(t_i)$ (recall $\pi \preceq t$ with $X = t_i|_p$). It follows that $\Re \mid\!\!-_{CRWL} \theta(t_i) \to t_i'$ applies the CRWL-rule **DC** over $\theta(t_i) \to t_i'$ to yield $\Re \mid\!\!-_{CRWL} \theta(X) \to c_j(...)$. We conclude that $\theta(X) \neq \perp$. □

The aim when using DNC is to transform an initial goal into a *solved goal* of the form $\exists$ U. S □ □ with $S \equiv X_1 \approx s_1,...,X_n \approx s_n$ representing a solution $\sigma_S = \{X_1 \mapsto s_1,...,X_n \mapsto s_n\}$. The notation ■, used in failure rules, represents an inconsistent goal without any solution. The calculus DNC consists of a set of transformation rules for goals. Each transformation takes the form $G \leadsto G'$, specifying one of the possible ways of performing one step of goal solving. Derivations are sequences of $\leadsto$-steps. In addition, to the purpose of applying the rules, we see conditions $a == b$ as symmetric. In the next subsections we give succint explanations justifying the formulation of the different kinds of rules that compose the DNC calculus.

4.1 Rules for Strict Equations

In this first subsection we describe the transformation rules supporting the treatment of == as a built-in symbol with the meaning of *open strict equality*, rather than a defined function symbol. Given a strict equation $s == t$ with s,t terms in a goal, the rules try to specify all the possibilities in the reduction of both terms to the same constructor term (a finite, totally defined value). In this sense, we distinguish the following different cases and rules according to the sintactic structure of s (analogously for t, due to the symmetry of ==).

- If s is $f(s_1,\ldots,s_n)$ with $f \in FS^n$, we introduce in the goal a new demanded production decorating s with an appropriate ODT $\mathfrak{I}_f$ to guide the evaluation in a needed narrowing style (see the goal transformations **RRA**, **DN**, **CS** and **DI** in Subsection 4.2). This is performed by the rule **DPI**.

- If s is $c(s_1,\ldots,s_n)$ with $c \in DC^n$, the term t has also a constructor symbol in the root or is a variable (any other case is collected in the item above). In the first case, the constructor symbol must be c to obtain an adequate descomposition of their corresponding arguments in smaller strict equations (see rule **DC**); otherwise rule **CF** fails. If t is a variable, we can suppose that is not produced (otherwise, we must wait until other rules become applicable producing an appropriate instantiation) and not occurs in s (otherwise, the rule **CY** fails). In this situation, if s represents a data value (i.e. a c-term without produced variables), it is possible to propagate the binding $X \mapsto t$ to the rest of the goal, by means of rule **BD**. Otherwise, it is possible to instantiate t with an outermost imitation of the sintactic structure of s to guide the descomposition of the strict equation, as is performed by the rule **IM**.

- Finally, it may happen that s and t are both variables. If any of them variables is produced, we must wait until other rule becomes applicable producing an appropriate instantiation. Otherwise, we can either eliminate a trivial strict equation using rule **ID**, or propagate a binding by means of rule **BD**.

DPI Demanded Production Introduction

$$\exists U \,.\, S \square P \square f(s_1,...,s_n) == t, E \leadsto$$
$$\exists R, U \,.\, S \square \langle f(s_1,...,s_n), \mathfrak{I}_{f(X1,...,Xn)}\rangle \to R, P \square R == t, E$$

if $f \in FS$, and both R and all variables in $\mathfrak{I}_{f(X1,...,Xn)}$ are new variables.

DC DeComposition

$$\exists U \,.\, S \square P \square c(s_1,...,s_n) == c(t_1,...,t_n), E \leadsto$$
$$\exists U \,.\, S \square P \square s_1 == t_1, \ldots, s_n == t_n, E \quad \text{if } c \in DC.$$

ID IDentity

$$\exists U \,.\, S \square P \square X == X, E \leadsto \exists U \,.\, S \square P \square E \quad \text{if } X \notin PVar(P).$$

BD BinDing

$\exists U \,.\, S \square P \square X == t, E \rightsquigarrow \exists U \,.\, X \approx t \,, \sigma(S \square P \square E)$

if $t \in$ CTerm, $Var(t) \cap PVar(P) = \varnothing$, $X \notin PVar(P)$, $X \notin Var(t)$, and $\sigma = \{X \mapsto t\}$.

IM IMitation

$\exists U \,.\, S \square P \square X == c(t_1,...,t_n), E \rightsquigarrow$

$\exists X_1,...,X_n, U \,.\, X \approx c(X_1,...,X_n) \,, \sigma(S \square P \square X_1 == t_1,...,X_n == t_n, E)$

if $c \in DC$, $c(t_1,...,t_n) \notin$ CTerm or $Var(c(t_1,...,t_n)) \cap PVar(P) \neq \varnothing$, $X \notin PVar(P)$, $X \notin Var_c(c(t_1,...,t_n))$, and $\sigma = \{X \mapsto c(X_1,...,X_n)\}$ with $X_1,...,X_n$ new variables.

4.2 Rules for Productions

The goal transformation rules concerning productions are designed with the aim of modelling the behaviour of lazy needed narrowing with *sharing*. Let us first comment the rules for *suspensions* $t \rightarrow R$. Rule **EL** is applied only if R becomes absent from the rest of the goal and then is unnecessary. In other case, the rule **IB** is applicable if t is a data value (i.e. a c-term, not further reducible), propagating this obtained value to all occurrences of R in the goal. If t is not yet a c-term, only transformations **IIM** or **SA** are applicable, propagating partially the obtained value to all the occurrences of R if t has a constructor symbol in the root, or awakening the suspension if t has a function symbol in the root decorating t with the appropriate definitional tree, respectively. Both cases require that R is a demanded variable, whose value in any solution must be different from $\perp$ because of Lemma 2. If R is not demanded, nothing can be done with the suspension but waiting until goal solving progresses by means of other rules. This will eventually happen, as shown by the completeness results in Section 5.

The goal transformation rules for demanded productions $<t, \mathfrak{I}> \rightarrow R$ encode the needed narrowing strategy guided by ODT $\mathfrak{I}$, in a vein similar to [10]. If $\mathfrak{I}$ is a *rule* ODT, then the transformation **RRA** chooses one of the available rules for rewriting t, introducing appropriate suspensions in the new goal so that lazy evaluation with call-time choice is ensured. If $\mathfrak{I}$ is a *case* tree, one of the transformations **CS**, **DI** or **DN** can be applied, according to the kind of symbol u occurring in t al the case-distinction position p. If u is a constructor c_i, then **CS** selects the appropriate subtree (if possible; otherwise **UC** fails). If u is a non-produced variable Y, then **DI** non-deterministically selects a subtree, generating an appropriate binding for Y. Finally, if u is a demanded function symbol g, **DN** introduces a new demanded production in the new goal, in order to evaluate $t|_p$. In any other case, selection of the subgoal $<t, \mathfrak{I}> \rightarrow R$ must be delayed until a further stage of the computation.

IB Input Binding

$\exists R, U \,.\, S \square t \rightarrow R, P \square E \rightsquigarrow \exists U \,.\, S \square \sigma(P \square E)$

if $t \in$ CTerm, and $\sigma = \{R \mapsto t\}$.

IIM Input IMitation

$\exists R, U \,.\, S \square c(t_1,...,t_n) \rightarrow R, P \square E \rightsquigarrow$

$\exists R_1,...,R_n, U \,.\, S \square \sigma(t_1 \rightarrow R_1, \ldots, t_n \rightarrow R_n, P \square E)$

if $c(t_1,...,t_n) \notin$ CTerm, $R \in DVar(P \square E)$, and $\sigma = \{R \mapsto c(R_1,...,R_n)\}$ with $R_1,...,R_n$ new variables.

SA Suspension Awakening

$\exists R, U \,.\, S \square f(t_1,...,t_n) \rightarrow R, P \square E \rightsquigarrow$

$\exists R, U \,.\, S \square <f(t_1,...,t_n) \,, \mathfrak{I}_{f(X1,...,Xn)}> \rightarrow R, P \square E$

if $f \in FS$, $R \in DVar(P \square E)$, and all variables in $\mathfrak{I}_{f(X1,...,Xn)}$ are new variables.

EL ELimination

$\exists R, U \,.\, S \square t \rightarrow R, P \square E \rightsquigarrow \exists U \,.\, S \square P \square E$ if $R \notin Var(P \square E)$.

RRA Rewrite Rule Application *(don't know choice: $1 \leq i \leq k$)*

$\exists R, U. \, S \square <t, \underline{rule}(l \rightarrow r_1 \Leftarrow C_1 \mid ... \mid r_k \Leftarrow C_k)> \rightarrow R, P \square E \rightsquigarrow$

$\exists \overline{X}, R, U. \, S \square \sigma_f(R_1) \rightarrow R_1,...,\sigma_f(R_m) \rightarrow R_m, \sigma_c(r_i) \rightarrow R, P \square \sigma_c(C_i), E$

where

- $\sigma = \sigma_c \cup \sigma_f$ with $Dom(\sigma) = Var(l)$ and $\sigma(l) = t$.
- $\sigma_c =_{def} \sigma \restriction Dom_c(\sigma)$, where $Dom_c(\sigma) = \{X \in Dom(\sigma) \mid \sigma(X) \in$ CTerm$\}$.
- $\sigma_f =_{def} \sigma \restriction Dom_f(\sigma)$, where $Dom_f(\sigma) = \{X \in Dom(\sigma) \mid \sigma(X) \notin$ CTerm$\} = \{R_1,...,R_m\}$.
- $\overline{X} = Var(l \rightarrow r_i \Leftarrow C_i) \setminus Dom_c(\sigma)$.

CS Case Selection

$\exists R, U \,.\, S \square <t, \underline{case}(\pi, X, [\mathfrak{I}_1,...,\mathfrak{I}_k])> \rightarrow R, P \square E \rightsquigarrow$

$\exists R, U \,.\, S \square <t, \mathfrak{I}_i> \rightarrow R, P \square E$

if $t|_{pos(X,\pi)} = c_i(...)$, with $1 \leq i \leq k$ given by t, where c_i is the constructor symbol associated to $\mathfrak{I}_i$.

DI Demanded Instantiation *(don't know choice: $1 \leq i \leq k$)*

$\exists R, U \,.\, S \square <t, \underline{case}(\pi, X, [\mathfrak{I}_1,...,\mathfrak{I}_k])> \rightarrow R, P \square E \rightsquigarrow$

$\exists Y_1,...,Y_{mi}, R, U \,.\, Y \approx c_i(Y_1,...,Y_{mi}), \sigma(S \square <t, \mathfrak{I}_i> \rightarrow R, P \square E)$

if $t|_{pos(X,\pi)} = Y$, $Y \notin PVar(P)$, $\sigma = \{Y \mapsto c_i(Y_1,...,Y_{mi})\}$ with c_i $(1 \leq i \leq k)$ the constructor symbol associated to $\mathfrak{I}_i$ and $Y_1,...,Y_{mi}$ are new variables.

DN Demanded Narrowing

$\exists R, U \,.\, S \square <t, \underline{case}(\pi, X, [\mathfrak{I}_1,...,\mathfrak{I}_k])> \rightarrow R, P \square E \rightsquigarrow$

$\exists R', R, U \,.\, S \square <t|_{pos(X,\pi)}, \mathfrak{I}_{g(Y1,...,Ym)}> \rightarrow R',$

$<t[R']_{pos(X,\pi)} \,, \underline{case}(\pi, X, [\mathfrak{I}_1,...,\mathfrak{I}_k])> \rightarrow R, P \square E$

if $t|_{pos(X,\pi)} = g(...)$ with $g \in FS^m$, and both R' and all variables in $\mathfrak{I}_{g(Y1,...,Ym)}$ are new variables.

4.3 Rules for Failure Detection

In this subsection we collect the failure rules already mentioned above, namely: failure in the unification of strict equations (conflict constructors and occurs check) and failure for constructor non-cover in demanded productions. We note that the failure rule **CY** for occurs check requires the condition $X \in \mathrm{Var_c}(t)$ instead of only $X \in \mathrm{Var}(t)$. The reason is that $X == t$ may have solutions when $X \in \mathrm{Var}(t) \setminus \mathrm{Var_c}(t)$. For instance, $X == f(X)$ certainly has solutions if f is defined by the rewrite rule $f(X) \to X$.

CF ConFlict

$\exists U$. S □ P □ $c(s_1,...,s_n) == d(t_1,...,t_m)$, E $\rightsquigarrow$ ■ if $c,d \in$ DC and $c \neq d$.

CY CYcle

$\exists U$. S □ P □ $X == t$, E $\rightsquigarrow$ ■ if $X \neq t$ and $X \in \mathrm{Var_c}(t)$.

UC Uncovered Case

$\exists R, U$. S □ $\langle t, \underline{case}(\pi,X,[\Im_1,...,\Im_k])\rangle \to R$, P □ E $\rightsquigarrow$ ■

if $t|_{pos(X,\pi)} = c(...)$ and $c \notin \{c_1,...,c_k\}$, where c_i is the constructor symbol associated to $\Im_i$ $(1 \le i \le k)$.

Finally, the section is closed with several examples of goal solving which highlight the main properties of the DNC calculus. At each goal transformation step, we underline which subgoal is selected.

Example 6. We compute the non-ground solution $\{Y \mapsto s(0), Z \mapsto s(X)\}$ from the goal $X + Y == Z$ in Example 1. This illustrates the use of an *open equality* to compute general answers rather than having to enumerate over an infinite set.

□ □ $\underline{X+Y == Z} \rightsquigarrow$ **DPI**

$\exists R$. □ $\underline{\langle X+Y,\Im_{A+B}\rangle \to R}$ □ R == Z $\rightsquigarrow_{\{Y \mapsto s(U)\}}$ **DI**

$\exists U,R$. Y ≈ $s(U)$ □ $\underline{\langle X+s(U),\Im_{A+s(C)}\rangle \to R}$ □ R == Z $\rightsquigarrow$ **RRA**

$\exists U,R$. Y ≈ $s(U)$ □ $\underline{s(X+U) \to R}$ □ R == Z $\rightsquigarrow_{\{R \mapsto s(R')\}}$ **IIM**

$\exists R',U$. Y ≈ $s(U)$ □ $\underline{X+U \to R'}$ □ $s(R')$ == Z $\rightsquigarrow$ **SA**

$\exists R',U$. Y ≈ $s(U)$ □ $\underline{\langle X+U,\Im_{A'+B'}\rangle \to R'}$ □ $s(R')$ == Z $\rightsquigarrow_{\{U \mapsto 0\}}$ **DI**

$\exists R',U$. U ≈ 0, Y ≈ $s(0)$ □ $\underline{\langle X+0,\Im_{A'+0}\rangle \to R'}$ □ $s(R')$ == Z $\rightsquigarrow$ **RRA**

$\exists R',U$. U ≈ 0, Y ≈ $s(0)$ □ $\underline{X \to R'}$ □ $s(R')$ == Z $\rightsquigarrow_{\{R' \mapsto X\}}$ **IB**

$\exists U$. U ≈ 0, Y ≈ $s(0)$ □ □ $s(X)$ == Z $\rightsquigarrow_{\{Z \mapsto s(X)\}}$ **BD**

$\exists U$. Z ≈ $s(X)$, U ≈ 0, Y ≈ $s(0)$ □ □

$\underline{computed\ solution}$: $\sigma = \{Y \mapsto s(0), Z \mapsto s(X)\}$. □

Example 7. We compute the solutions from Example 2. This illustrates the use of productions for ensuring *call-time choice* and the use of ODTs for ensuring needed narrowing steps.

□ □ $\underline{double(coin) == N} \rightsquigarrow$ **DPI**

$\exists R$. □ $\underline{\langle double(coin),\Im_{double(X)}\rangle \to R}$ □ R == N $\rightsquigarrow$ **RRA**

$\exists X,R$. □ $coin \to X$, $\underline{X+X \to R}$ □ R == N $\rightsquigarrow$ **AS**

$\exists X,R$. □ $\underline{coin \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow$ **AS**

$\exists X,R$. □ $\underline{\langle coin,\Im_{coin}\rangle \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow$ **RRA**

$\exists X,R$. □ $\underline{0 \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow_{\{X \mapsto 0\}}$ **IB**

$\exists R$. □ $\underline{\langle 0+0,\Im_{A+B}\rangle \to R}$ □ R == N $\rightsquigarrow$ **CS**

$\exists R$. □ $\underline{\langle 0+0,\Im_{A+0}\rangle \to R}$ □ R == N $\rightsquigarrow$ **RRA**

$\exists R$. □ $\underline{0 \to R}$ □ R == N $\rightsquigarrow_{\{R \to 0\}}$ **IB**

□ □ $\underline{0 == N} \rightsquigarrow_{\{N \mapsto 0\}}$ **BD**

N ≈ 0 □ □ ⇒ $\underline{computed\ solution}$: $\sigma_1 = \{N \mapsto 0\}$.

□ □ $\underline{double(coin) == N} \rightsquigarrow$ **DPI**

$\exists R$. □ $\underline{\langle double(coin),\Im_{double(X)}\rangle \to R}$ □ R == N $\rightsquigarrow$ **RRA**

$\exists X,R$. □ $coin \to X$, $\underline{X+X \to R}$ □ R == N $\rightsquigarrow$ **AS**

$\exists X,R$. □ $\underline{coin \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow$ **AS**

$\exists X,R$. □ $\underline{\langle coin,\Im_{coin}\rangle \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow$ **RRA**

$\exists X,R$. □ $\underline{s(0) \to X}$, $\langle X+X,\Im_{A+B}\rangle \to R$ □ R == N $\rightsquigarrow_{\{X \mapsto s(0)\}}$ **IB**

$\exists R$. □ $\underline{\langle s(0)+s(0),\Im_{A+B}\rangle \to R}$ □ R == N $\rightsquigarrow$ **CS**

$\exists R$. □ $\underline{\langle s(0)+s(0),\Im_{A+s(C)}\rangle \to R}$ □ R == N $\rightsquigarrow$ **RRA**

$\exists R$. □ $\underline{s(s(0)+0) \to R}$ □ R == N $\rightsquigarrow_{\{R \mapsto s(R')\}}$ **IIM**

$\exists R'$. □ $\underline{s(0)+0 \to R'}$ □ $s(R')$ == N $\rightsquigarrow$ **AS**

$\exists R'$. □ $\underline{\langle s(0)+0,\Im_{A'+B'}\rangle \to R'}$ □ $s(R')$ == N $\rightsquigarrow$ **CS**

$\exists R'$. □ $\underline{\langle s(0)+0,\Im_{B'+0}\rangle \to R'}$ □ $s(R')$ == N $\rightsquigarrow$ **RRA**

$\exists R'$. □ $\underline{s(0) \to R'}$ □ $s(R')$ == N $\rightsquigarrow_{\{R' \mapsto s(0)\}}$ **IB**

□ □ $\underline{s(s(0)) == N} \rightsquigarrow_{\{N \mapsto s(s(0))\}}$ **BD**

N ≈ $s(s(0))$ □ □ ⇒ $\underline{computed\ solution}$: $\sigma_2 = \{N \mapsto s(s(0))\}$.

We note that both computations are in essence needed narrowing derivations modulo non-deterministic choices between overlapping rules, as in inductively sequential narrowing [3]. □

Example 8. We compute the solutions from the goal *head* (*bits*) == Z, where we use the non-deterministic function *bits* from Example 3 and the rule *head* ([X|Xs]) → X. We use again Prolog's syntax for the list constructors. This example illustrates the use of productions to achieve the effect of a *demand-driven* evaluation.

□ □ $\underline{head\ (bits) == Z} \rightsquigarrow$ **DPI**

$\exists R$. □ $\underline{\langle head\ (bits),\Im_{head}\rangle \to R}$ □ R == Z $\rightsquigarrow$ **DN**

$\exists R',R$. □ $\underline{\langle bits,\Im_{bits}\rangle \to R'}$, $\langle head\ (R'),\Im_{head}\rangle \to R$ □ R == Z $\rightsquigarrow$ **RRA**

$\exists X,R',R$. □ $\underline{[X \mid bits] \to R'}$, $\langle head\ (R'),\Im_{head}\rangle \to R$

□ $coin == X$, R == Z $\rightsquigarrow_{\{R' \mapsto [Y|Ys]\}}$ **IIM**

$\exists X,Y,Ys,R$. □ X → Y, $bits \to Ys$, $\underline{\langle head\ ([Y|Ys]),\Im_{head}\rangle \to R}$

□ $coin == X$, R == Z $\rightsquigarrow$ **RRA**

$\exists X,Y,Ys,R$. □ $\underline{X \to Y}$, $bits \to Ys$, $\underline{Y \to R}$

□ $coin == X$, R == Z $\rightsquigarrow_{\{Y \mapsto X\} \circ \{R \mapsto Y\}}$ **IB×2**

$\exists X,Ys$. □ $\underline{bits \to Ys}$ □ $coin == X$, X == Z $\rightsquigarrow$ **EL**

$\exists X\,.\,\square\,\square\ coin == X,\ \underline{X == Z} \rightsquigarrow_{\{X \mapsto Z\}}$ **BD**

$\exists X\,.\,X \approx Z\ \square\ \square\ \underline{coin == Z} \rightsquigarrow$ **DPI**

$\exists R'',X\,.\,X \approx Z\ \square\ \underline{<coin,\mathfrak{I}_{coin}> \rightarrow R''}\ \square\ R'' == Z \rightsquigarrow$ **RRA**

$\exists R'',X\,.\,X \approx Z\ \square\ \underline{0\ (\text{or}\ s(0)) \rightarrow R''}\ \square\ R'' == Z \rightsquigarrow_{\{R'' \mapsto 0\ (\text{or}\ s(0))\}}$ **IB**

$\exists X\,.\,X \approx Z\ \square\ \square\ \underline{0\ (\text{or}\ s(0)) == Z} \rightsquigarrow_{\{Z \mapsto 0\ (\text{or}\ s(0))\}}$ **BD**

$\exists X\,.\,Z \approx 0\ (\text{or}\ s(0)),\ X \approx 0\ (\text{or}\ s(0)))\ \square\ \square$

computed solutions: $\sigma_1 = \{Z \mapsto 0\}$ and $\sigma_2 = \{Z \mapsto s(0)\}$ □

5. PROPERTIES OF DNC

To prove soundness and completeness of DNC w.r.t. CRWL semantics we use techniques similar to those used for the CLNC calculus in [7]. Some proofs omitted here can be found in [20, 19]. The first result proves correctness of a single DNC step. It says that transformation steps preserve admissibility of goals, fail only in case of unsatisfiable goals and do not introduce new solutions.

Lemma 3 (Correctness Lemma). *Assume any admissible goal G for a given COISS $\mathcal{R}$. Then:*

a) If $G \rightsquigarrow \blacksquare$, then $Sol(G) = \emptyset$.

b) Otherwise, if $G \rightsquigarrow G'$, then G' is an admissible goal. Moreover, if $\theta' \in Sol(G')$ then there exists $\theta \in Sol(G)$ with $\theta = \theta'\ [\mathcal{V} - (EVar(G) \cup EVar(G'))]$.

The following soundness result follows easily from Lemma 3. It ensures that computed answers for a goal G are indeed solutions of G.

Theorem 1 (Soundness of DNC). *If G_0 is an initial goal and $G_0 \rightsquigarrow G_1 \rightsquigarrow \ldots \rightsquigarrow G_n$, where $G_n \equiv \exists U.\ S \square \square$, then $\sigma_S \in Sol(G_0)$.*

Proof. If we repeatedly backwards apply c) of Lemma 3, we obtain $\theta \in Sol(G_0)$ such that $\theta = \sigma_S\ [\mathcal{V} - \bigcup_{0 \le i \le n} EVar(G_i)]$. By noting that $EVar(G_0) = \emptyset$ and $Var(G_0) \cap \bigcup_{0 \le i \le n} EVar(G_i) = \emptyset$, we conclude $\theta = \sigma_S\ [Var(G_0)]$. But then, since σ_S is a total c-substitution, $\sigma_S \in Sol(G_0)$. □

Completeness of DNC is proved with the help of a well-founded ordering $\triangleright$ for admissible goals and a result that guarantees that DNC transformations can be chosen to make progress towards the computation of a given solution. A similar technique is used in [7] to prove completeness of CLNC, but the well-founded ordering there is a simpler one.

Definition 5 (Well-founded ordering for admissible goals). *Let $\mathcal{R}$ be a COISS, $G \equiv \exists U.\ S \square P \square E$ an admissible goal for $\mathcal{R}$ and $\mathcal{M}$ a witness for $\theta \in Sol(G)$. We define:*

a) A multiset of pairs of natural numbers $\mathcal{W}(G,\theta,\mathcal{M}) =_{def} \{||\Pi||_{G,\theta} \mid \Pi \in \mathcal{M}\}$, called **weight of the witness**, *where for each CRWL-proof $\Pi \in \mathcal{M}$, the pair of numbers $||\Pi||_{G,\theta} \in \mathbb{N} \times \mathbb{N}$ is:*

- *if $\Pi \equiv \mathcal{R} \mid\!\!-_{CRWL} \theta(t) \rightarrow \theta(R)$ for $(<t,\mathfrak{T}> \rightarrow R) \in P$, then $||\Pi||_{G,\theta} =_{def} (\,|\Pi|_{OR}\,,\,|\Pi|\,)$.*
- *if $\Pi \equiv \mathcal{R} \mid\!\!-_{CRWL} \theta(t) \rightarrow \theta(R)$ for $(t \rightarrow R) \in P$, then $||\Pi||_{G,\theta} =_{def} (\,|\Pi|_{OR}\,,\,1 + |\Pi|\,)$.*
- *if $\Pi \equiv \mathcal{R} \mid\!\!-_{CRWL} \theta(t) == \theta(s)$ for $(t == s) \in E$, then $||\Pi||_{G,\theta} =_{def} (\,|\Pi|_{OR}\,,\,|\Pi|\,)$.*

In all the cases, $|\Pi|_{OR}$ is the number of deduction steps in Π which use the CRWL deduction rule ***OR*** *and $|\Pi|$ is the total number of deduction steps in Π.*

b) A multiset of natural numbers $\mathcal{D}(G) =_{def} \{\,|\mathfrak{T}| \mid (<t,\mathfrak{T}> \rightarrow R) \in P\,\}$, called **weight of the definitional trees** *of the goal G, where each $|\mathfrak{T}| \in \mathbb{N}$ is the total number of nodes in the tree $\mathfrak{T}$.*

Over triples $(G,\theta,\mathcal{M})$ such that G is an admissible goal for $\mathcal{R}$ and $\mathcal{M}$ is a witness of $\theta \in Sol(G)$, we define a well-founded ordering $(G,\theta,\mathcal{M}) \triangleright (G',\theta',\mathcal{M}') \Leftrightarrow_{def} (\mathcal{W}(G,\theta,\mathcal{M})\,,\,\mathcal{D}(G)) >_{lex} (\mathcal{W}(G',\theta',\mathcal{M}')\,,\,\mathcal{D}(G'))$, where $>_{lex}$ is the lexicographic product of $>_{mul1}$ and $>_{mul2}$, $>_{mul1}$ is the multiset order for multisets over $\mathbb{N} \times \mathbb{N}$ induced by the lexicographic product of $>_{\mathbb{N}}$ and $>_{\mathbb{N}}$, and $>_{mul2}$ is the multiset order for multisets over $\mathbb{N}$ induced by $>_{\mathbb{N}}$. See [6] for definitions of these notions.

Lemma 4 (Progress Lemma). *Assume an admissible goal G for some COISS, which is not in solved form, and a given solution $\theta \in Sol(G)$ with witness $\mathcal{M}$. Then:*

a) There is some DNC transformation applicable to G.

b) Whatever applicable DNC transformation is chosen, there is some transformation step $G \rightsquigarrow G'$ and some solution $\theta' \in Sol(G')$ with a witness $\mathcal{M}'$, such that $\theta = \theta'\ [\mathcal{V} - (EVar(G) \cup EVar(G'))]$ and $(G,\theta,\mathcal{M}) \triangleright (G',\theta',\mathcal{M}')$.

Proof (a). If $G \equiv \exists U.\ S \square P \square E$ is not a solved form, then P or E is not empty. We will proceed by assuming gradually that no rule, except one (namely **EL**), is applicable to G, and then we will conclude that this remaining rule **EL** must be applicable. Note that failure rules cannot be applicable because otherwise G would have no solution, due to Lemma 3 a). Assume that **DPI**, **DC**, **ID**, **BD** and **IM** are not applicable. Then, all the joinability statements in E must be of one of the following two forms: $X == Y$ or $X == c(s_1,\ldots,s_n)$, with an occurrence of a variable R which is both produced and demanded. All the productions in P must be of one of the following forms: $t \rightarrow R$ or $<t, \mathfrak{T}> \rightarrow R$. Now assume that **IB** and **RRA**, **CS**, **DI**, **DN** are not applicable. Then it must be the case that all the productions in P are of the form $t \rightarrow R$ with $t \notin CTerm$ or $<t, \underline{case}(\pi, Z, [\mathfrak{T}_1,\ldots,\mathfrak{T}_k])> \rightarrow R$ with $t|_{pos(Z,\pi)}$ a produced (and demanded) variable R'. Now assume that **IIM** and **SA** are not applicable. Then, there are no demanded productions in P (otherwise, we could find a demanded production of the form mentioned above and a suspension $t' \rightarrow R'$. Since $t' \notin CTerm$, must be $d(t_1,\ldots,t_m)$ with $d \in DC$, and in this case **IIM** must be

applicable, or $f(t_1,...,t_m)$ with $f \in$ FS, and in this case **SA** must be applicable. Contradiction in both cases). So, all the productions in P must be suspensions of the form $t \to R$ where $t \notin$ CTerm and R is not a demanded variable. At this point E must be empty, because we had previously concluded that any strict equation in E must demand some produced variable. Finally, let R be minimal in the $\gg_P^+$ relation (such minimal elements do exist, due to the finite number of variables occurring in G and the property **CYC** of admissible goals). Such R cannot appear in any other approximation statement in P, and therefore **EL** can be applied to the condition $t \to R$ where R appears.

Proof (b). See [20, 19]. □

By reiterated application of Lemma 4, the following completeness result is easy to prove. The proof reveals that DNC is *strongly complete*, i.e. completeness does not depend on the choice of the transformation rule (among all the applicable rules) in the current goal.

Theorem 2 (Completeness of DNC). *Let $\mathcal{R}$ a COISS, G an initial goal and $\theta \in Sol(G)$. Then there exists a solved form $\exists U.\ S \square \square$ such that $G \leadsto^* \exists U.\ S \square \square$ and $\sigma_S \lesssim \theta\,[Var(G)]$.*

Proof. Thanks to Lemma 4 it is possible to construct a derivation $G \equiv G_0 \leadsto G_1 \leadsto G_2 \leadsto ...$ for which there exist $\theta_0 = \theta, \theta_1, \theta_2, ...$ and $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, ...$ such that $\theta_i = \theta_{i-1}\,[\mathcal{V} - (EVar(G_{i-1}) \cup EVar(G_i))]$, $\mathcal{M}_i$ is a witness of $\theta_i \in Sol(G_i)$ and $(G_i,\theta_i,\mathcal{M}_i) \rhd (G_{i-1},\theta_{i-1},\mathcal{M}_{i-1})$. Since $\rhd$ is well founded, such a derivation must be finite, ending with a solved form $G_n \equiv \exists\ U.\ S \square \square$. Since $EVar(G_0) = \varnothing$ and $Var(G_0) \cap EVar(G_i)$ for all $i = 1, ..., n$ it is easy to see that $\theta_n = \theta\ [Var(G)]$. Now, if $X \in Var(G)$ and there is an equation $X \approx t$ in S, we can use the facts $\theta_n = \theta\ [Var(G)]$ and $\theta_n \in Sol(S)$ to obtain $\theta(X) = \theta_n(X) = \theta_n(t) = \theta_n(\sigma_S(X))$. It follows that $\theta = \theta_n \circ \sigma_S\ [Var(G)]$, and thus $\sigma_S \lesssim \theta\ [Var(G)]$. □

Let us now briefly analyze the behaviour of DNC computations from the viewpoint of needed narrowing. Obviously, the goal transformation rules for demanded productions which deal with ODTs already present in the goal (namely, **RRA**, **CS**, **DI** and **DN**) lead to performing needed narrowing steps. On the other hand, the goal transformation **DPI** and **SA** are the only way to introduce demanded productions $\langle \pi, \Im \rangle \to R$ into the goal. In both cases, R is a demanded variable in the new goal. From the definition of demanded variables within Definition 3, it is easily seen that any demanded variable always occurs at some position of the goal which is needed in the sense of needed narrowing (viewing == as a function symbol, so that the conceptual framework from [5, 3] can be applied). Therefore, DNC derivations actually encode needed narrowing derivations in the sense of [5, 3], with an improved treatment of == as open strict equality and *call-time* non-deterministic choice.

Regarding the range of applicability of DNC, the restriction to the class of COISSs is not a serious one. Using techniques known from [4, 15, 16] we have proved in [20] that any CRWL-program $\mathcal{R}$ can be effectively transformed into a COISS $\mathcal{R}'$ using new auxiliary function symbols, such that for all $e \in Term_\perp$ and $t \in CTerm_\perp$ in the original signature of $\mathcal{R}$ one has $\mathcal{R} \vdash_{CRWL} e \to t \Leftrightarrow \mathcal{R}' \vdash_{CRWL} e \to t$. The combination of this transformation techniques with the DNC calculus offers a sound and complete narrowing procedure for the whole class of the left-linear constructor-based conditional rewrite systems.

6. CONCLUSIONS AND FUTURE WORK

We have presented a demand-driven narrowing calculus DNC for COISS, a wide class of constructor-based conditional TRSs. We have proved that DNC conserves the good properties of needed narrowing [5, 3] while being sound and strongly complete w.r.t. CRWL semantics [7], which ensures a *call-time choice* treatment of non-determinism and an open interpretation of strict equality.

Because of these results, we believe that DNC contributes to bridge the gap between formally defined lazy narrowing calculi and existing languages such as Curry [9] and $\mathcal{TOY}$ [1]. As future work, we plan to extend DNC and COISS in other to include more expressive features, such as higher-order rewrite rules and constraints.

7. ACKNOWLEDGEMENTS

The author is thankful to Mario Rodríguez-Artalejo and Francisco J. López-Fraguas for their collaboration, comments and contributions during the first stages of the development of this work and for the help in preparing the final version of this paper.

8. REFERENCES

[1] M. Abengózar-Carneros et al. $\mathcal{TOY}$: a multiparadigm declarative language, version 2.0. Technical report, Dep. SIP, UCM Madrid, January 2001.

[2] S. Antoy. Definitional trees. In *Proc. Int. Conf. on Algebraic and Logic Programming (ALP'92)*, volume 632 of Springer LNCS, pages 143-157, 1992.

[3] S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. of ALP'97*, pages 16-30. Springer LNCS 1298, 1997.

[4] S. Antoy. Constructor-based conditional narrowing. In *Proc. PPDP'01, ACM Press*, pp. 199-206, 2001.

[5] S. Antoy, R. Echahed, M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776-822, 2000.

[6] F. Baader, T. Nipkow. Term Rewriting and All That. *Cambridge University Press*, 1998.

[7] J. C. González-Moreno, F.J.López-Fraguas, M.T. Hortalá-González, M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *The Journal of Logic Programming*, 40:47-87, 1999.

[8] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583-628, 1994.

[9] M. Hanus (ed.). Curry: An Integrated Functional Logic Language. Version 0.7.1, June 2000. Available at http://www.informatik.uni-kiel.de/curry/report.html.

[10] M. Hanus, C. Prehofer. Higher-Order Narrowing with Definitional Trees. *Journal of Functional Programming*, 9(1):33-75, 1999.

[11] M. Hanus, S. Lucas, A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters (Elsevier)*, vol. 67, nº 1, pp. 1-8, 1998.

[12] G. Huet, J. J. Lévy. Computations in orthogonal term rewriting systems I, II. In J. L. Lassez and G. Plotkin, editors, Computational Logic: Essays in Honour of J. Alan Robinson, pages 395-414 and 415-443. *The MIT Press*, 1991.

[13] H. Hussmann. Nondeterministic Algebraic Specification and non confluent term rewriting. *Journal of Logic Programming*, 12:237-255, 1992.

[14] T. Ida, K. Nakahara. Leftmost outside-in narrowing calculi. *Journal of Functional Programming*, 7(2):129-161, 1997.

[15] R. Loogen, F. J. López-Fraguas, M. Rodríguez-Artalejo. A demand driven computation strategy for lazy narrowing. In *Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'93)*, volume 714 of Springer LNCS pages 184-200, 1993.

[16] F. J. López-Fraguas, J. Sánchez-Hernández. Functional logic programming with failure: A set-oriented view. In *Proc. LPAR'01*, pages 455 – 469, Springer LNAI 2250, 2001.

[17] A. Middeldorp. Call by Need Computations to Root-Stable Form. In *Proc. POPL'1997, ACM Press*, 1997.

[18] A. Middeldorp, S. Okui. A deterministic lazy narrowing calculus. *Journal of Symbolic Computation*, 25 (6), pp. 733-757, 1998.

[19] R. del Vado Vírseda. Estrategias de Estrechamiento Perezoso. A Master's Thesis Directed by Dr. M. Rodríguez-Artalejo. Dpto. Sistemas Informáticos y Programación. Facultad de CC. Matemáticas, U.C.M. September 2002.

[20] R. del Vado Vírseda. A Demand Narrowing with Overlapping Definitional Trees. Technical Report SIP 132/03, UCM Madrid, 2003.