A Lazy Narrowing Calculus for Declarative Constraint Programming *

F. J. López-Fraguas
fraguas@sip.ucm.es

M. Rodríguez-Artalejo
mario@sip.ucm.es

Rafael del Vado Vírseda
rdelvado@sip.ucm.es

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain
Avda. Complutense s/n, 28040 Madrid

ABSTRACT

The new generic scheme $CFLP(\mathcal{D})$ has been recently proposed in [24] as a logical and semantic framework for lazy constraint functional logic programming over a parametrically given constraint domain $\mathcal{D}$. In this paper we extend such framework with a suitable operational semantics, which relies on a new *constrained lazy narrowing calculus* for goal solving parameterized by a constraint solver over the given domain $\mathcal{D}$. This new calculus is sound and strongly complete w.r.t. the declarative semantics of $CFLP(\mathcal{D})$ programs, which was formalized in [24] by means of a *Constraint Rewriting Logic* $CRWL(\mathcal{D})$.

Categories and Subject Descriptors

D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; D.1.6 [**Programming Techniques**]: Logic Programming; D.3.2 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*

General Terms

Algorithms, Languages, Performance, Theory

Keywords

Functional logic programming languages, constraint logic programming, rewrite systems, narrowing, constraint solvers

1. INTRODUCTION

The idea of *Constraint Functional Logic Programming* arose around 1990 as an attempt to combine two lines of research in declarative programming, namely *Constraint Logic Programming* and *Functional Logic Programming*.

Constraint logic programming was started by a seminal paper published by J. Jaffar and J.L. Lassez in 1987 [16], where the CLP scheme was first introduced. The aim of the scheme was to define a family of constraint logic programming languages $CLP(\mathcal{D})$ parameterized by a constraint domain $\mathcal{D}$, in such a way that the well established results on the declarative and operational semantics of logic programs [20, 1] could be lifted to all the $CLP(\mathcal{D})$ languages in an elegant and uniform way. The best updated presentation of the classical CLP semantics can be found in [18]. In the course of time, CLP has become a very successful programming paradigm, supporting a clean combination of logic programming and domain-specific methods for constraint satisfaction, simplification and optimization, and leading to practical applications in various fields [32, 17, 28].

On the other hand, functional logic programming refers to a line of research started in the 1980s and aiming at the integration of the best features of functional programming and logic programming. As far as we know, the first attempt to combine functional and logic languages was done by J.A. Robinson and E.E. Sibert when proposing the language LOGLISP [30]. Some other early proposals for the design of functional + logic languages are described in [9]. A good survey of the operational principles and implementation techniques used for the integration of functions into logic programming can be found in [14]. *Narrowing*, a natural combination of rewriting and unification, originally proposed as a theorem proving tool, has been used as a goal solving mechanism in functional logic languages such as Curry [15] and $\mathcal{TOY}$ [22]. Under various more or less restrictive conditions, several narrowing strategies are known to be complete for goal solving [14, 29].

To our best knowledge, the first attempt of combining constraint logic programming and functional logic programming was the $CFLP(\mathcal{D})$ scheme proposed by J. Darlington, Y.K. Guo and H. Pull [8]. The idea behind this approach can be described by the equation $CFLP(\mathcal{D}) = CLP(FP(\mathcal{D}))$, intended to mean that a $CFLP$ language over the constraint domain $\mathcal{D}$ is viewed as a CLP language over an extended constraint domain $FP(\mathcal{D})$ whose constraints include equations between expressions involving user defined functions, to be solved by narrowing.

The $CFLP(\mathcal{D})$ scheme proposed by F.J. López-Fraguas in [21] aimed at providing a declarative semantics such that

*This research has been partially supported by the Spanish National Project MELODIAS (TIC2002-01167).

PPDP'04, August 24–26, 2004, Verona, Italy.

$CLP(\mathcal{D})$ programs could be formally understood as a particular case of $CFLP(\mathcal{D})$ programs. In the classical approach to CLP semantics a constraint domain is viewed as a first order structure $\mathcal{D}$, and constraints are viewed as first order formulas that can be interpreted in $\mathcal{D}$. In [21] programs were built as sets of constrained rewrite rules. In order to support a lazy semantics for the user defined functions, constraint domains $\mathcal{D}$ were formalized as continuous structures, with a Scott domain [13] as carrier, and a continuous interpretation of function and predicate symbols. The resulting semantics had many pleasant properties, but also some limitations. In particular, defined functions had to be first order and deterministic, and the use of patterns in function definitions had to be simulated by means of special constraints.

More recently, yet another $CFLP$ scheme has been proposed in the Phd Thesis of M. Marin [25]. This approach introduces $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{L})$, a family of languages parameterized by a constraint domain $\mathcal{D}$, a strategy $\mathcal{S}$ which defines the cooperation of several constraint solvers over $\mathcal{D}$, and a constraint lazy narrowing calculus $\mathcal{L}$ for solving constraints involving functions defined by user given constrained rewrite rules. This approach relies on solid work on higher-order lazy narrowing calculi and has been implemented on top of Mathematica [26, 27]. Its main limitation from our viewpoint is the lack of declarative semantics.

In a recent work [24] we have proposed a new generic scheme $CFLP(\mathcal{D})$, intended as a logical and semantic framework for lazy Constraint Functional Logic Programming over a parametrically given constraint domain $\mathcal{D}$, which provides a clean and rigorous declarative semantics for $CFLP(\mathcal{D})$ languages as in the $CLP(\mathcal{D})$ scheme but overcomes the limitations of our older $CFLP(\mathcal{D})$ scheme [21]. $CFLP(\mathcal{D})$ programs are presented as sets of constrained rewrite rules that define the behaviour of possibly higher-order and/or nondeterministic lazy functions over $\mathcal{D}$. The main novelties in [24] were a new formalization of constraint domains for $CFLP$, a new notion of interpretation for $CFLP(\mathcal{D})$ programs, and a new *Constraint Rewriting Logic* $CRWL(\mathcal{D})$ parameterized by a constraint domain, which provides a logical characterization of program semantics.

Our aim in this paper is to formalize an operational semantics for the new generic scheme $CFLP(\mathcal{D})$ proposed in [24]. We present a lazy constrained narrowing calculus $CLNC(\mathcal{D})$ for solving goals for $CFLP(\mathcal{D})$ programs, which can be proved sound and strongly complete w.r.t. $CRWL(\mathcal{D})$'s semantics. These properties qualify $CLNC(\mathcal{D})$ as a convenient computation mechanism for declarative constraint programming languages.

The reader of this paper is assumed to have some knowledge on the foundations of logic programming [20, 1] and term rewriting [5]. The rest of the paper is organized as follows. The next section is devoted to summarize the presentation of the $CFLP(\mathcal{D})$ scheme [24] and to formalize the notion of a constraint solver over a given constraint domain. In Section 3 we give a formal presentation of the calculus $CLNC(\mathcal{D})$. We discuss the soundness and completeness results in Section 4. Finally, some conclusions and plans for future work are drawn in Section 5.

2. THE GENERIC SCHEME $CFLP(\mathcal{D})$

In this section we introduce the main features of the $CFLP(\mathcal{D})$ scheme [24], as a basis for the constraint narrowing calculus $CLNC(\mathcal{D})$ presented in the rest of the paper.

2.1 Applicative expressions, patterns and substitutions

We briefly introduce the syntax of applicative expressions and patterns, which is needed for understanding the construction of constraint domains and constraint solvers.

We assume a *universal signature* $\Sigma = \langle DC, FS \rangle$, where $DC = \bigcup_{n \in \mathbb{N}} DC^n$ and $FS = \bigcup_{n \in \mathbb{N}} FS^n$ are families of countably infinite and mutually disjoint sets of *data constructors* resp. *evaluable function symbols*, each one with an associated arity. We write $\Sigma_\perp$ for the result of extending DC^0 with the special symbol $\perp$, intended to denote an undefined data value. As notational conventions, we use $c, d \in DC$, $f, g \in FS$ and $h \in DC \cup FS$, and we define the *arity* of $h \in DC^n \cup FS^n$ as $ar(h) = n$. We also assume that DC^0 includes the three constants $true$, $false$ and $success$, which are useful for representing the results returned by various primitive functions. Next we assume a countably infinite set $\mathcal{V}$ of *variables* $X, Y, \ldots$ and a set $\mathcal{U}$ of urelements $u, v, \ldots$, mutually disjoint and disjoint from $\Sigma_\perp$. Urelements are intended to represent some domain specific set of values, as e.g. the set $\mathbb{R}$ of the real numbers used in the well-known CLP language $CLP(\mathcal{R})$ [19]. *Partial expressions* $e \in Exp_\perp(\mathcal{U})$ have the following syntax:

$$e ::= \perp \mid u \mid X \mid h \mid (e\, e_1)$$

where $u \in \mathcal{U}$, $X \in \mathcal{V}$, $h \in DC \cup FS$. These expressions are usually called *applicative*, because $(e\, e_1)$ stands for the *application* operation (represented as juxtaposition) which applies the function denoted by e to the argument denoted by e_1. Applicative syntax is common in higher order functional languages. The usual first order syntax for expressions can be translated to applicative syntax by means of so-called *curried notation*. For instance, $f(X, g(Y))$ becomes $(f\, X\, (g\, Y))$. Following a usual convention, we assume that application associates to the left, and we use the notation $(e\, \overline{e}_n)$ to abbreviate $(e\, e_1 \ldots e_n)$. The set of variables occurring in e is written $var(e)$. An expression e is called *linear* iff there is no $X \in var(e)$ having more than one occurrence in e. The following classification of expressions is also useful: $(X\, \overline{e}_m)$, with $X \in \mathcal{V}$ and $m \geq 0$, is called a *flexible expression*, while $u \in \mathcal{U}$ and $(h\, \overline{e}_m)$ with $h \in DC \cup FS$ are called *rigid expressions*. Moreover, a rigid expression $(h\, \overline{e}_m)$ is called *active* iff $h \in FS$ and $m \geq ar(h)$, and *passive* otherwise. Intuitively, reducing an expression at the root makes sense only if the expression is active. Some interesting subsets of $Exp_\perp(\mathcal{U})$ are: $GExp_\perp(\mathcal{U})$, the set of the *ground* expressions e such that $var(e) = \emptyset$; $Exp(\mathcal{U})$, the set of the *total* expressions e with no occurrences of $\perp$; $GExp(\mathcal{U})$, the set of the ground and total expressions $GExp_\perp(\mathcal{U}) \cap Exp(\mathcal{U})$. Another important subclass of expressions is the set of *partial patterns* $s, t \in Pat_\perp(\mathcal{U})$, whose syntax is defined as follows:

$$t ::= \perp \mid u \mid X \mid c\, \overline{t}_m \mid f\, \overline{t}_m$$

where $u \in \mathcal{U}$, $X \in \mathcal{V}$, $c \in DC^n$, $m \leq n$, $f \in FS^n$, $m < n$. Note that expressions $(f\, \overline{t}_m)$ with $(f \in FS^n, m \geq n)$ are not allowed as patterns, because they are potentially evaluable using a primitive or user given definition for function f. Patterns of the form $(f\, \overline{t}_m)$ with $f \in FS^n$, $m < n$, have been used in functional logic programming [12] as a convenient representation of higher order values. The subsets $Pat(\mathcal{U})$, $GPat_\perp(\mathcal{U})$, $GPat(\mathcal{U}) \subseteq Pat_\perp(\mathcal{U})$ consisting of the *total*, *ground* and *total and ground* patterns, respectively, are

defined in the natural way. Following the spirit of denotational semantics [13], we view $Pat_\bot(\mathcal{U})$ as the set of finite elements of a semantic domain, and we define the *information ordering* $\sqsubseteq$ as the least partial ordering over $Pat_\bot$ satisfying the following properties: $\bot \sqsubseteq t$ for all $t \in Pat_\bot(\mathcal{U})$, and $(h\bar{t}_m) \sqsubseteq (h\bar{t'}_m)$ whenever these two expressions are patterns and $t_i \sqsubseteq t'_i$ for all $1 \leq i \leq m$. In the sequel, $\bar{t}_m \sqsubseteq \bar{t'}_m$ will be understood as meaning that $t_i \sqsubseteq t'_i$ for all $1 \leq i \leq m$. Note that a pattern $t \in Pat_\bot(\mathcal{U})$ is maximal w.r.t. the information ordering iff t is a total pattern, i.e. $t \in Pat(\mathcal{U})$. For some purposes it is useful to extend the information ordering to the set of all partial expressions. This extension is simply defined as the least partial ordering over $Exp_\bot(\mathcal{U})$ which verifies $\bot \sqsubseteq e$ for all $e \in Exp_\bot(\mathcal{U})$, and $(e\, e_1) \sqsubseteq (e'\, e'_1)$ whenever $e \sqsubseteq e'$ and $e_1 \sqsubseteq e'_1$. As usual, we define *substitutions* $\sigma \in Sub_\bot(\mathcal{U})$ as mappings $\sigma : \mathcal{V} \to Pat_\bot(\mathcal{U})$ extended to $\sigma : Exp_\bot(\mathcal{U}) \to Exp_\bot(\mathcal{U})$ in the natural way. Similarly, we consider *total substitutions* $\sigma \in Sub(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to Pat(\mathcal{U})$, *ground substitutions* $\sigma \in GSub_\bot(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to GPat_\bot(\mathcal{U})$, and *ground total substitutions* $\sigma \in GSub(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to GPat(\mathcal{U})$. By convention, we write ε for the identity substitution, $e\sigma$ instead of $\sigma(e)$, and $\sigma\theta$ for the composition of σ and θ, such that $e(\sigma\theta) = (e\sigma)\theta$ for any $e \in Exp_\bot(\mathcal{U})$. We define the *domain* and the *variable range* of a substitution in the usual way, namely: $dom(\sigma) = \{X \in \mathcal{V} \mid \sigma(X) \neq X\}$ and $ran(\sigma) = \bigcup_{X \in dom(\sigma)} var(\sigma(X))$. As usual, a substitution σ such that $dom(\sigma) \cap ran(\sigma) = \emptyset$ is called *idempotent*. For any set of variables $\mathcal{X} \subseteq \mathcal{V}$ we define the *restriction* $\sigma \upharpoonright \mathcal{X}$ as the substitution σ' such that $dom(\sigma') = \mathcal{X}$ and $\sigma'(X) = \sigma(X)$ for all $X \in \mathcal{X}$. We use the notation $\sigma =_\mathcal{X} \theta$ to indicate that $\sigma \upharpoonright \mathcal{X} = \theta \upharpoonright \mathcal{X}$, and we abbreviate $\sigma =_{\mathcal{V} \setminus \mathcal{X}} \theta$ as $\sigma =_{\setminus \mathcal{X}} \theta$. Finally, we consider two different ways of comparing given substitutions $\sigma, \sigma' \in Sub_\bot(\mathcal{U})$. σ is said to be less particular than σ' over $\mathcal{X} \subseteq \mathcal{V}$ (in symbols, $\sigma \leq_\mathcal{X} \sigma'$) iff $\sigma\theta =_\mathcal{X} \sigma'$ for some $\theta \in Sub_\bot(\mathcal{U})$. The notation $\sigma \leq \sigma'$ abbreviates $\sigma \leq_\mathcal{V} \sigma'$. σ is said to bear less information than σ' over $\mathcal{X} \subseteq \mathcal{V}$ (in symbols, $\sigma \sqsubseteq_\mathcal{X} \sigma'$) iff $\sigma(X) \sqsubseteq \sigma'(X)$ for all $X \in \mathcal{X}$. The notation $\sigma \sqsubseteq \sigma'$ abbreviates $\sigma \sqsubseteq_\mathcal{V} \sigma'$.

2.2 Constraints over a given constraint domain

Intuitively, a *constraint domain* is expected to provide a set of specific data elements, along with certain primitive functions and predicates operating upon them. The following definition extends the notion of constraint domain $\mathcal{D}$ introduced in [24] by adding a constraint solver:

Definition 1. **Constraint Domains.**

1. A *constraint signature* is any family $PF = \bigcup_{n \in \mathbb{N}} PF^n$ of primitive function symbols p, each one with an associated arity, such that $PF^n \subseteq FS^n$ for each $n \in \mathbb{N}$.

2. A *constraint domain* of signature PF is any structure $\mathcal{D} = \langle D_\mathcal{U}, \{p^\mathcal{D} \mid p \in PF\}, Solve^\mathcal{D} \rangle$ such that the carrier set $D_\mathcal{U} = GPat_\bot(\mathcal{U})$ coincides with the set of ground patterns for some set of urelements $\mathcal{U}$, the interpretation $p^\mathcal{D}$ of each $p \in PF^n$ satisfies the following requirements:

   (a) $p^\mathcal{D} \subseteq D_\mathcal{U}^n \times D_\mathcal{U}$, which boils down to $p^\mathcal{D} \subseteq D_\mathcal{U}$ in the case $n = 0$. In the sequel we always write $p^\mathcal{D}\, \bar{t}_n \to t$ to indicate that $(\bar{t}_n, t) \in p^\mathcal{D}$. In the case $n = 0$, this notation boils down to $p^\mathcal{D} \to t$.

   (b) $p^\mathcal{D}$ behaves monotonically in its arguments and antimonotonically in its result; i.e., if $p^\mathcal{D}\, \bar{t}_n \to t$, $\bar{t}_n \sqsubseteq \bar{t'}_n$ and $t \sqsupseteq t'$ one has $p^\mathcal{D}\, \bar{t'}_n \to t'$.

   (c) $p^\mathcal{D}$ behaves *radically* in the following sense: whenever $p^\mathcal{D}\, \bar{t}_n \to t$ and $t \neq \bot$, there is some total $t' \in D_\mathcal{U}$ such that $p^\mathcal{D}\, \bar{t}_n \to t'$ and $t' \sqsupseteq t$.

   and $Solve^\mathcal{D}$ is a *constraint solver*, whose expected behaviour will be explained in *Definition 5* below.

Assuming an arbitrarily fixed constraint domain $\mathcal{D}$ built over a certain set of urelements $\mathcal{U}$, we will now define the syntax of constraints. In the sequel, we will write $DF = FS \setminus PF$ for the set of user defined function symbols, and $DF^n = FS^n \setminus PF^n$ for the set of user defined function symbols of arity n. The following definition distinguishes primitive constraints without any active occurrence of defined function symbols, from user defined constraints that can have such occurrences. For the sake of brevity, we sometimes write simply 'constraints' instead of 'user defined constraints'.

Definition 2. **Syntax of Constraints.**

1. *Atomic Primitive Constraints* have the syntactic form $p\, \bar{t}_n \to\!!\, t$, with $p \in PF^n$, $t_i \in Pat_\bot(\mathcal{U})$ for all $1 \leq i \leq n$, and $t \in Pat(\mathcal{U})$. The special constants $\lozenge$ and $\blacklozenge$ are also atomic primitive constraints.

2. *Primitive Constraints* are built from atomic primitive constraints by means of logical conjunction $\wedge$ and existential quantification $\exists$.

3. *Atomic Constraints* have the syntactic form $p\, \bar{e}_n \to\!!\, t$, with $p \in PF^n$, $e_i \in Exp_\bot(\mathcal{U})$ for all $1 \leq i \leq n$, and $t \in Pat(\mathcal{U})$. The special constants $\lozenge$ and $\blacklozenge$ are also atomic constraints.

4. *Constraints* are built from atomic constraints by means of conjunction $\wedge$ and existential quantification $\exists$.

In the sequel we use the notations: $PCon_\bot(\mathcal{D})$ for the set of all the primitive constraints π over $\mathcal{D}$ and $PCon(\mathcal{D})$ for the set of all the total primitive constraints over $\mathcal{D}$, defined as $\{\pi \in PCon_\bot(\mathcal{D}) \mid \pi$ has no occurrences of $\bot\}$. We also write $DCon_\bot(\mathcal{D})$ for the set of all the user defined constraints δ over $\mathcal{D}$, as well as $DCon(\mathcal{D})$ for the subset of $DCon_\bot(\mathcal{D})$ consisting of total constraints. We reserve the capital letters Π resp. C for sets of primitive resp. user defined constraints, often interpreted as conjunctions. The semantics of primitive constraints depends on the notion of solution, presented in the next definition.

Definition 3. **Solutions of Primitive Constraints.**

1. The set of *valuations* resp. *total valuations* over $\mathcal{D}$ is defined as $Val_\bot(\mathcal{D}) = GSub_\bot(\mathcal{U})$ resp. $Val(\mathcal{D}) = GSub(\mathcal{U})$.

2. The set of solutions of $\pi \in PCon_\bot(\mathcal{D})$ is a subset $Sol_\mathcal{D}(\pi) \subseteq Val_\bot(\mathcal{D})$ recursively defined as follows:

   (a) $Sol_\mathcal{D}(\lozenge) = Val_\bot(\mathcal{D})$ and $Sol_\mathcal{D}(\blacklozenge) = \emptyset$.

   (b) $Sol_\mathcal{D}(p\, \bar{t}_n \to\!!\, t) = \{\eta \in Val_\bot(\mathcal{D}) \mid t\eta$ is total and $p^\mathcal{D}\, \bar{t}_n\eta \to t\eta\}$.

   (c) $Sol_\mathcal{D}(\pi_1 \wedge \pi_2) = Sol_\mathcal{D}(\pi_1) \cap Sol_\mathcal{D}(\pi_2)$.

(d) $Sol_{\mathcal{D}}(\exists X.\pi) = \{\eta \in Val_{\perp}(\mathcal{D}) \mid \eta' \in Sol_{\mathcal{D}}(\pi)$ for some $\eta' =_{\setminus\{X\}} \eta\}$.

3. The set of solutions of $\Pi \subseteq PCon_{\perp}(\mathcal{D})$ is defined as $Sol_{\mathcal{D}}(\Pi) = \bigcap_{\pi\in\Pi} Sol_{\mathcal{D}}(\pi)$, corresponding to a logical reading of Π as the conjunction of its members. In particular, $Sol_{\mathcal{D}}(\emptyset) = Val_{\perp}(\mathcal{D})$, corresponding to the logical reading of an empty conjunction as the identically true constraint $\Diamond$.

Using the notion of solution, some useful semantic notions related to primitive constraints are easily introduced:

Definition 4. **Primitive Semantic Notions.** Assuming a finite set $\Pi \subseteq PCon_{\perp}(\mathcal{D})$ of primitive constraints, a primitive constraint $\pi \in PCon_{\perp}(\mathcal{D})$, expressions $e, e' \in Exp_{\perp}(\mathcal{U})$, patterns $\bar{t}_n$, $t \in Pat_{\perp}(\mathcal{U})$, and a primitive function symbol $p \in PF^n$, we define:

1. π is called *satisfiable* in $\mathcal{D}$ (in symbols $Sat_{\mathcal{D}}(\pi)$) iff $Sol_{\mathcal{D}}(\pi) \neq \emptyset$. Otherwise π is called *unsatisfiable* (in symbols $Unsat_{\mathcal{D}}(\pi)$). Analogously for constraint sets.

2. π is a *consequence* of Π in $\mathcal{D}$ (in symbols, $\Pi \models_{\mathcal{D}} \pi$) iff $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\pi)$. In particular, $p\,\bar{t}_n \to! t$ is a consequence of Π in $\mathcal{D}$ (in symbols, $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \to! t$) iff $p^{\mathcal{D}}\,\bar{t}_n\eta \to t\eta$ with $t\eta$ total holds for all $\eta \in Sol_{\mathcal{D}}(\Pi)$.

3. $e \sqsubseteq e'$ is a *consequence* of Π in $\mathcal{D}$ (in symbols, $\Pi \models_{\mathcal{D}} e \sqsubseteq e'$) iff $e\eta \sqsubseteq e'\eta$ holds for all $\eta \in Sol_{\mathcal{D}}(\Pi)$.

At this point we can specify the expected behaviour of the constraint solver $Solve^{\mathcal{D}}$ introduced in *Definition 1.* The following definition is inspired in [21, 2, 23].

Definition 5. **Constraint Solvers.**

1. We say that a variable $X \in \mathcal{V}$ is *demanded* by a set of primitive constraints $\Pi \subseteq PCon(\mathcal{D})$ iff $\mu(X) \neq \perp$ holds for every $\mu \in Sol_{\mathcal{D}}(\Pi)$. We write $dvar_{\mathcal{D}}(\Pi)$ for the set of the variables demanded by Π. For practical constraint domains, $dvar_{\mathcal{D}}(\Pi)$ is expected to be computable (see *Appendix A*).

2. A *constraint solver* over a constraint domain $\mathcal{D}$ is a function named $Solve^{\mathcal{D}}$ expecting as parameters a finite set $S \subseteq PCon(\mathcal{D})$ of atomic primitive constraints (called the *constraint store*) and a finite set of variables $\chi \subseteq \mathcal{V}$ (called the set of *protected variables*). The solver is expected to return a finite disjunction $Solve^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$ satisfying the following requirements:

   (a) Each $S_i \subseteq PCon(\mathcal{D})$ is a finite set of atomic primitive constraints such that $Solve^{\mathcal{D}}(S_i, \chi) = S_i \,\square\, \varepsilon$ (i.e. S_i is in χ-*solved form*). Furthermore, $var(S_i) \cap \chi = \emptyset$ or else $dvar_{\mathcal{D}}(S_i) \cap \chi \neq \emptyset$.

   (b) Each $\sigma_i \in Sub(\mathcal{U})$ is an idempotent substitution of total patterns for variables such that $dom(\sigma_i) \cap var(S_i) = \emptyset$ and $\chi \cap (dom(\sigma_i) \cup ran(\sigma_i)) = \emptyset$.

   (c) $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \,\square\, \sigma_i)$, where $\exists_{\setminus S}$ is the existential quantification of all the variables $\overline{U}_i$ in $S_i \,\square\, \sigma_i$ $(1 \leq i \leq k)$ not occurring free in S and $Sol_{\mathcal{D}}(\exists\overline{U}_i.\ S_i \,\square\, \sigma_i) = \{\mu \in Val_{\perp}(\mathcal{D}) \mid$ exists $\mu' \in Val_{\perp}(\mathcal{D})$ such that $\mu' =_{\setminus\overline{U}_i} \mu$, $\mu' \in Sol_{\mathcal{D}}(S_i)$ and $X\mu' \equiv t\mu'$ for each $X \mapsto t \in \sigma_i\}$.

   In the case $k = 0$, $\bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$ is understood as $\blacklozenge$. In this case, $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(\blacklozenge) = \emptyset$ means failure detection.

From an operational viewpoint, a solver offers a choice between k alternatives. No alternative can bind protected variables. Moreover, for each alternative $S_i \,\square\, \sigma_i$, either all the protected variables disappear or some protected variable becomes demanded. More details on the working of solvers will be given in *Section 3.*

Example 1. **The constraint domain $\mathcal{H}_{seq}$.** We consider the constraint domain $\mathcal{H}_{seq}$ built over an empty set of urelements and having the strict equality *seq* as its only primitive, interpreted to behave as follows: $seq^{\mathcal{H}_{seq}}\ t\ t \to true$ for all total $t \in GPat(\emptyset)$; $seq^{\mathcal{H}_{seq}}\ t\ s \to false$ for all $t, s \in GPat_{\perp}(\emptyset)$ such that t, s have no common upper bound w.r.t. the information ordering; $seq^{\mathcal{H}_{seq}}\ t\ s \to \perp$ otherwise. In the sequel, $t == s$ abbreviates $seq\ t\ s \to! true$, $t /= s$ abbreviates $seq\ t\ s \to! false$ and $Tot(t)$ abbreviates $seq\ t\ t \to! true$. A possible constraint solver $Solve^{\mathcal{H}_{seq}}$ can be found in *Appendix A.* The specification of solvers for other useful constraint domains is planned as future work.

2.3 *CFLP*($\mathcal{D}$)-programs

In the sequel we assume an arbitrarily fixed constraint domain $\mathcal{D}$ built over a set of urelements $\mathcal{U}$. In this setting, *CFLP*($\mathcal{D}$)*-Programs* are presented as sets of constrained rewrite rules that define the behaviour of possibly higher order and/or non-deterministic lazy functions over $\mathcal{D}$, called *program rules.* More precisely, a program rule R for $f \in DF^n$ has the form $R : f\,\bar{t}_n \to r \Leftarrow P \,\square\, C$ and is required to satisfy the conditions listed below:

1. The *left-hand side* $f\,\bar{t}_n$ is a linear expression, and for all $1 \leq i \leq n$, $t_i \in Pat(\mathcal{U})$ are total patterns.

2. The *right-hand side* $r \in Exp(\mathcal{U})$ is a total expression.

3. P is a finite sequence of so-called *productions* $e_i \to s_i$ $(1 \leq i \leq k)$ also intended to be interpreted as conjunction, and fulfilling the following *admissibility conditions*:

   (a) For all $1 \leq i \leq k$, $e_i \in Exp(\mathcal{U})$ is a total expression, $s_i \in Pat(\mathcal{U})$ is a total linear pattern, and $var(s_i) \cap var(f\,\bar{t}_n) = \emptyset$.

   (b) It is possible to reorder the productions of P in the form $P \equiv e_1 \to s_1, \ldots, e_k \to s_k$ where $var(e_i) \cap var(s_j) = \emptyset$ for all $1 \leq i \leq j \leq k$.

   (c) For all $1 \leq i < j \leq k$, $var(s_i) \cap var(s_j) = \emptyset$.

4. $C \subseteq DCon(\mathcal{D})$ is a finite set of total constraints, intended to be interpreted as conjunction, and possibly including occurrences of defined function symbols.

A program rule such that P and C are both empty can be abbreviated as $f\,\bar{t}_n \to r$. We note that an equivalent formulation for the admissibility condition 3.(b) can be obtained by defining the *production relation* $X \gg_P Y$ iff there is some $1 \leq i \leq k$ such that $X \in var(e_i)$ and $Y \in var(s_i)$, and requiring that the transitive closure of $\gg_P$ must be irreflexive, or equivalently, a strict partial order.

Example 2. The following *CFLP*($\mathcal{D}$)-program can be used over the constraint domain $\mathcal{H}_{seq}$ presented in *Example 1.*

We use the constructors $0 \in DC^0$, $s \in DC^1$, a constructor for pairs (i.e. (e_1, e_2) denotes the pair of a first element e_1 and a second element e_2) and a Prolog-like syntax for list constructors (i.e. [ ] denotes the empty list and $[X|Xs]$ denotes a non-empty list consisting of a first element X and a remaining list Xs). More examples of $CFLP(\mathcal{D})$-programs can be found in [24].

$$
\begin{array}{llll}
from & N & \to & [N|from(s\ N)] \\
null & [\,] & \to & s\ 0 \\
null & [X|Xs] & \to & 0
\end{array}
$$

$$
\begin{array}{llll}
split & [\,] & \to & ([\,],\,[\,]) \\
split & [X|Xs] & \to & case\ R\ X\ Ys\ Zs \Leftarrow split\ Xs \to (Ys,\,Zs) \\
 & & & \qquad \square\ seq\ X\ s\ 0 \to\!!\ R
\end{array}
$$

$$
\begin{array}{llllll}
case & true & X & Ys & Zs & \to & ([X|Ys],\,Zs) \\
case & false & X & Ys & Zs & \to & (Ys,\,[X|Zs])
\end{array}
$$

2.4 The Constraint Rewriting Logic $CRWL(\mathcal{D})$

The *Constraint Rewriting Logic* $CRWL(\mathcal{D})$, parameterized by a constraint domain $\mathcal{D}$, was introduced in [24] in order to provide a declarative semantic for $CFLP(\mathcal{D})$-programs. In order to define this logic, we must first introduce some preliminary notions about the constrained statements that we intend to derive from a given $CFLP(\mathcal{D})$-program.

Definition 6. **Constrained Statements and $\mathcal{D}$-entailment.** Let $\mathcal{D}$ be any fixed constraint domain over a set of urelemets $\mathcal{U}$. In what follows we assume partial patterns $t, t_i \in Pat_\bot(\mathcal{U})$, partial expressions e, $e_i \in Exp_\bot(\mathcal{U})$, and a finite set $\Pi \subseteq PCon_\bot(\mathcal{D})$ of primitive constraints.

1. We consider two possible kinds of constrained statements (*c-statements*):
   (a) *c-productions* $e \to t \Leftarrow \Pi$. A c-production is called *trivial* iff $t = \bot$ or $Unsat_\mathcal{D}(\Pi)$.
   (b) *c-atoms* $p\,\overline{e}_n \to\!!\ t \Leftarrow \Pi$, with $p \in PF^n$ and t total. A c-atom is called *trivial* iff $Unsat_\mathcal{D}(\Pi)$.
2. Given two c-statements φ and φ', we say that φ $\mathcal{D}$-*entails* φ' (in symbols, $\varphi \succcurlyeq_\mathcal{D} \varphi'$) iff one of the two following cases holds:
   (a) $\varphi = e \to t \Leftarrow \Pi$, $\varphi' = e' \to t' \Leftarrow \Pi'$, and there is some $\sigma \in Sub_\bot(\mathcal{U})$ such that $\Pi' \models_\mathcal{D} \Pi\sigma$, $\Pi' \models_\mathcal{D} e' \sqsupseteq e\sigma$, $\Pi' \models_\mathcal{D} t' \sqsubseteq t\sigma$.
   (b) $\varphi = p\,\overline{e}_n \to\!!\ t \Leftarrow \Pi$, $\varphi' = p\,\overline{e'}_n \to\!!\ t' \Leftarrow \Pi'$, and there is some $\sigma \in Sub_\bot(\mathcal{U})$ such that $\Pi' \models_\mathcal{D} \Pi\sigma$, $\Pi' \models_\mathcal{D} p\,\overline{e'}_n \sqsupseteq (p\,\overline{e}_n)\sigma$, $\Pi' \models_\mathcal{D} t' \sqsupseteq t\sigma$. [1]

The next definition assumes a given $CFLP(\mathcal{D})$-program $\mathcal{P}$ and uses the notation $[\mathcal{P}]_\bot$ for the set $\{R\theta \mid R \in \mathcal{P}, \theta \in Sub_\bot(\mathcal{U})\}$ consisting of all the possible instances of the function defining rules belonging to $\mathcal{P}$. The purpose of the calculus is to infer the semantic validity of arbitrary c-statements from the program rules in $\mathcal{P}$.

Definition 7. **Constrained Rewriting Calculus.** We write $\mathcal{P} \vdash_\mathcal{D} \varphi$ to indicate that the c-statement φ can be derived from $\mathcal{P}$ in the constrained rewriting calculus $CRWL(\mathcal{D})$ using the inference rules given in *Figure 1.* Some of these rules depend on the semantic notions given in *Definition 4* and the following semantic notion for productions: $p\,\overline{t}_n \to t$ is a *consequence* of Π in $\mathcal{D}$ (in symbols, $\Pi \models_\mathcal{D} p\,\overline{t}_n \to t$) iff $p^\mathcal{D}\,\overline{t}_n\eta \to t\eta$ holds for all $\eta \in Sol_\mathcal{D}(\Pi)$.

By convention, we agree that no inference rule of the constrained rewriting calculus is applied in case that some textually previous rule can be used. In particular, no rule except **TI** can be used to infer a trivial c-statement, and **SP** is not applied whenever **RR** is applicable. Moreover, we also agree that the premise $P \square C \Leftarrow \Pi$ in rule $\mathbf{DF}_\mathcal{P}$ must be understood as a shorthand for several premises $\alpha \Leftarrow \Pi$, one for each atomic statement α occurring in $P \square C$.

Any derivation in the constrained rewriting calculus can be represented as a *proof tree* whose nodes are labelled by c-statements, where each node has been inferred from its children by means of the inference rules. In the sequel, we will use the following notations:

1. T is called an *easy proof tree* iff T makes no use of the inference rules $\mathbf{DF}_\mathcal{P}$, **PF** and **AC**.
2. $\mid T \mid$ denotes the *restricted size* of the proof tree T, defined as the number of nodes in T which are inferred with some of the rules $\mathbf{DF}_\mathcal{P}$, **PF** or **AC**. Obviously, $\mid T \mid = 0$ iff T is an easy proof tree.
3. $T : \mathcal{P} \vdash_\mathcal{D} \varphi$ indicates that $\mathcal{P} \vdash_\mathcal{D} \varphi$ is witnessed by the proof tree T.

The next result state two useful properties of the constrained rewriting calculus. The (rather technical) proof and other properties of $CRWL(\mathcal{D})$ can be found in [24].

LEMMA 1. *Properties of the $CRWL(\mathcal{D})$ Calculus.*

1. *Approximation Property: For any $e \in Exp_\bot(\mathcal{U})$, $t \in Pat_\bot(\mathcal{U})$: $\Pi \models_\mathcal{D} e \sqsupseteq t$ iff there is some easy proof tree T such that $T : \vdash_\mathcal{D} e \to t \Leftarrow \Pi$ (derivation from empty program).*
2. *Entailment Property: $T : \mathcal{P} \vdash_\mathcal{D} \varphi$ and $\varphi \succcurlyeq_\mathcal{D} \varphi'$ implies $T' : \mathcal{P} \vdash_\mathcal{D} \varphi'$ for some proof tree T' such that $\mid T' \mid \leq \mid T \mid$.*

Correctness results relating $CRWL(\mathcal{D})$-derivability to a suitable model-theoretic semantics are also given in [24]. More precisely, as argued in [24], $CRWL(\mathcal{D})$ is sound and complete w.r.t. strong semantics, and sound and ground complete w.r.t. weak semantics, two different classes of semantics.

3. THE $CLNC(\mathcal{D})$ CALCULUS

This section presents a new *Constrained Lazy Narrowing Calculus* over a parametrically given constraint domain $\mathcal{D}$ (shortly, $CLNC(\mathcal{D})$) for solving $CFLP(\mathcal{D})$-goals, borrowing ideas and techniques from previous lazy narrowing calculi for FLP [11, 12, 31] and $CFLP$ [21, 2, 3] languages. We give first a precise definition for the class of admissible goals, answers and solutions we are going to work with.

Definition 8. A *goal* for a given $CFLP(\mathcal{D})$-program must have the form $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$, where the symbol $\square$ must be interpreted as conjunction, and:

- $evar(G) =_{def} \overline{U}$ is the set of so-called *existential variables* of the goal G. These are intermediate variables, whose bindings in a solution may be partial patterns. $fvar(G) =_{def} var(G) \backslash evar(G)$ is the set of so-called *free variables* of the goal G.

[1] Note that $\Pi' \models_\mathcal{D} t' \sqsubseteq t\sigma$ would be wrong, because $\to\!!$ behaves monotonically both in its arguments and in its result. See *Definition 3.(b).*

TI Trivial Inference $\dfrac{}{\varphi}$ if φ is a trivial c-statement.

RR Restricted Reflexivity $\dfrac{}{t \to t \Leftarrow \Pi}$ if $t \in \mathcal{U} \cup \mathcal{V}$.

SP Simple Production $\dfrac{}{s \to t \Leftarrow \Pi}$ if $s \in Pat_\bot(\mathcal{U})$, $s \in \mathcal{V}$ or $t \in \mathcal{V}$, and $\Pi \models_\mathcal{D} s \sqsupseteq t$.

DC Decomposition $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_m \to t_m \Leftarrow \Pi}{h\overline{e}_m \to h\overline{t}_m \Leftarrow \Pi}$ if $h\overline{e}_m$ is passive.

IR Inner Reduction $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_m \to t_m \Leftarrow \Pi}{h\overline{e}_m \to X \Leftarrow \Pi}$

if $h\overline{e}_m$ is passive but not a pattern, $X \in \mathcal{V}$ and $\Pi \models_\mathcal{D} h\overline{t}_m \sqsupseteq X$.

PF Primitive Function $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to t \Leftarrow \Pi}$

if $p \in PF^n$, $t_i \in Pat_\bot(\mathcal{U})$ for each $1 \le i \le n$, and $\Pi \models_\mathcal{D} p\,\overline{t}_n \to t$.

DF$_\mathcal{P}$ $\mathcal{P}$-Defined Function $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, C \Leftarrow \Pi,\ r \to t \Leftarrow \Pi}{f\,\overline{e}_n \to t \Leftarrow \Pi}$

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, C \Leftarrow \Pi,\ r \to s \Leftarrow \Pi,\ s\,\overline{a}_k \to t \Leftarrow \Pi}{f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi}$$

if $f \in DF^n$ $(k > 0)$, $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, C) \in [\mathcal{P}]_\bot$, $s \in Pat_\bot(\mathcal{U})$. .

AC Atomic Constraint $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to\!!\ t \Leftarrow \Pi}$

if $p \in PF^n$, $t_i \in Pat_\bot(\mathcal{U})$ for each $1 \le i \le n$, and $\Pi \models_\mathcal{D} p\,\overline{t}_n \to\!!\ t$.

Figure 1: Rules for $CRWL(\mathcal{D})$-derivability

- $P \equiv e_1 \to t_1, \ldots, e_n \to t_n$ is a finite conjunction of productions where $e_i \in Exp(\mathcal{U})$ and $t_i \in Pat(\mathcal{U})$ for all $1 \le i \le n$. The set of *produced variables* of G is defined as $pvar(P) =_{def} var(t_1) \cup \ldots \cup var(t_n)$.
- $C \equiv \delta_1, \ldots, \delta_k$ is a finite conjunction of atomic constraints (possibly including occurrences of defined function symbols).
- $S \equiv \pi_1, \ldots, \pi_l$ is a finite conjunction of atomic primitive constraints, called *constraint store*.
- σ is an idempotent substitution called *answer substitution* such that $dom(\sigma) \cap var(P \,\square\, C \,\square\, S) = \emptyset$.

Additionally, any *admissible goal* must satisfy the following admissibility conditions, called *goal invariants*:

LN Each produced variable is produced only once, i.e. the tuple $t_1, \ldots, t_n$ must be *linear*.

EX All the produced variables must be existential, i.e. $pvar(P) \subseteq evar(G)$.

NC The transitive closure of the production relation $\gg_P$ (given in *Subsection 2.3*) must be irreflexive, or equivalently, a strict partial order.

SL No produced variable enters the answer substitution, i.e. $var(\sigma) \cap\ pvar(P) = \emptyset$.

Similarly to [11, 12, 31], $CLNC(\mathcal{D})$ uses a notion of *demanded variable* to deal with lazy evaluation, but now w.r.t. a constraint store. Intuitively, productions $e \to X$ in G, where e is not a pattern, do not propagate the binding $\{X \mapsto e\}$. Instead, evaluation of e must be triggered, *provided that X is demanded in G*. The result will be shared by all the occurrences of X.

Definition 9. Let $G \equiv \exists\overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ be an admissible goal for a given $CFLP(\mathcal{D})$-program and $X \in var(G)$. We say that X is a *demanded variable* in G iff $X \in dvar_\mathcal{D}(S)$ (see *Definition 5*) or there exists some production $(X\overline{a}_k \to t) \in P$ such that, either $t \notin \mathcal{V}$ or else $k > 0$ and t is a demanded variable in G. We write $dvar_\mathcal{D}(G)$ (or more precisely $dvar_\mathcal{D}(P \,\square\, S)$) for the set of demanded variables in the goal G.

DC Decomposition $\exists\overline{U}.\ h\bar{e}_m \to h\bar{t}_m, P \square C \square S \square \sigma \vdash_{DC} \exists\overline{U}.\ \overline{e_m \to t_m}, P \square C \square S \square \sigma$ if $h\bar{e}_m$ is passive.

SP Simple Production $\exists[X], \overline{U}.\ X \to t, P \square C \square S \square \sigma \vdash_{SP_{[1],2}} \exists\overline{U}.\ (P \square C \square S)\sigma_0 \square \sigma[\sigma_0]$ if $t \notin \mathcal{V}, [X \notin \overline{U}]$ and $\sigma_0 = \{X \mapsto t\}$.

$\exists X, \overline{U}.\ t \to X, P \square C \square S \square \sigma \vdash_{SP_3} \exists\overline{U}.\ (P \square C \square S)\sigma_0 \square \sigma$ if $t \in Pat(\mathcal{U})$ and $\sigma_0 = \{X \mapsto t\}$.

IM Imitation $\exists X, \overline{U}.\ h\bar{e}_m \to X, P \square C \square S \square \sigma \vdash_{IM} \exists\overline{X}_m, \overline{U}.\ (\overline{e_m \to X_m}, P \square C \square S)\sigma_0 \square \sigma$

if $h\bar{e}_m \notin Pat(\mathcal{U})$ is passive, $X \in dvar_{\mathcal{D}}(P \square S)$ and $\sigma_0 = \{X \mapsto h\overline{X}_m\}$ with $\overline{X}_m$ new variables such that $h\overline{X}_m \in Pat(\mathcal{U})$.

EL Elimination $\exists X, \overline{U}.\ e \to X, P \square C \square S \square \sigma \vdash_{EL} \exists\overline{U}.\ P \square C \square S \square \sigma$ if $X \notin var(P \square C \square S \square \sigma)$.

PF Primitive Function $\exists\overline{U}.\ p\bar{e}_n \to t, P \square C \square S \square \sigma \vdash_{PF} \exists\overline{X}_q, \overline{U}.\ \overline{e_q \to X_q}, P \square C \square p\bar{t}_n \to! t, S \square \sigma$

if $p \in PF^n$, $t \notin \mathcal{V}$ or $t \in dvar_{\mathcal{D}}(P \square S)$, and $\overline{X}_q$ are new variables ($0 \le q \le n$ is the number of $e_i \notin Pat(\mathcal{U})$) such that $t_i \equiv X_j$ $(0 \le j \le q)$ if $e_i \notin Pat_\perp(\mathcal{U})$ and $t_i \equiv e_i$ otherwise for each $1 \le i \le n$.

DF Defined Function $\exists\overline{U}.\ f\bar{e}_n \to t, P \square C \square S \square \sigma \vdash_{DF_1} \exists\overline{Y}, \overline{U}.\ \overline{e_n \to t_n}, r \to t, P', P \square C', C \square S \square \sigma$

$\exists\overline{U}.\ f\bar{e}_n\bar{a}_k \to t, P \square C \square S \square \sigma \vdash_{DF_2} \exists X, \overline{Y}, \overline{U}.\ \overline{e_n \to t_n}, r \to X, X\bar{a}_k \to t, P', P \square C', C \square S \square \sigma$

if $f \in DF^n$ $(k > 0), t \notin \mathcal{V}$ or $t \in dvar_{\mathcal{D}}(P \square S)$ and $R : f\bar{t}_n \to r \Leftarrow P' \square C'$ is a fresh variant of a rule in $\mathcal{P}$, with $\overline{Y} = var(R)$ and X new variables.

FV Functional Variable $\exists[F], \overline{U}.\ F\bar{e}_q \to t, P \square C \square S \square \sigma \vdash_{FV_{[1],2}} \exists\overline{X}_p, \overline{U}.\ (h\overline{X}_p\bar{e}_q \to t, P \square C \square S)\sigma_0 \square \sigma[\sigma_0]$

if $[F \notin pvar(P)], q > 0, t \notin \mathcal{V}$ or $t \in dvar_{\mathcal{D}}(P \square S)$, $\sigma_0 = \{F \mapsto h\overline{X}_p\}$ and $\overline{X}_p$ are new variables such that $h\overline{X}_p \in Pat(\mathcal{U})$.

Figure 2: $CLNC(\mathcal{D})$-rules for constrained lazy narrowing

CS Constraint Solving $\exists\overline{U}.\ P \square C \square S \square \sigma \vdash_{CS\{\chi\}} \exists\overline{Y}_i, \overline{U}.\ (P \square C)\sigma_i \square S_i \square \sigma\sigma_i$

if $\chi = pvar(P)$, S is not χ-solved, $Solve^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \square \sigma_i)$, and $\overline{Y}_i$ are the new variables introduced by the solver in $S_i \square \sigma_i$, for each $1 \le i \le k$.

AC Atomic Constraint $\exists\overline{U}.\ P \square\ p\bar{e}_n \to! t, C \square S \square \sigma \vdash_{AC} \exists\overline{X}_q, \overline{U}.\ \overline{e_q \to X_q}, P \square C \square\ p\bar{t}_n \to! t, S \square \sigma$

if $p \in PF^n$, $p\bar{e}_n \to! t$ is an atomic constraint, $\overline{X}_q$ are new variables ($0 \le q \le n$ is the number of $e_i \notin Pat_\perp(\mathcal{U})$) such that $t_i \equiv X_j$ $(0 \le j \le q)$ if $e_i \notin Pat_\perp(\mathcal{U})$ and $t_i \equiv e_i$ otherwise for each $1 \le i \le n$.

CF Conflict Failure $\exists\overline{U}.\ h\bar{e}_p \to h'\bar{t}_q, P \square C \square S \square \sigma \vdash_{CF} \blacksquare$ if $h\bar{e}_p$ is passive, and $h \ne h'$ or else $p \ne q$.

SF Solving Failure $\exists\overline{U}.\ P \square C \square S \square \sigma \vdash_{SF\{\chi\}} \blacksquare$ if $\chi = pvar(P)$, S is not χ-solved, and $Solve^{\mathcal{D}}(S, \chi) = \blacklozenge$.

Figure 3: $CLNC(\mathcal{D})$-rules for constraint solving and failure detection

An admissible goal $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$ is called a *solved goal* iff P and C are empty and S is in $\emptyset$-solved form in the sense of *Definition 5.* An *initial goal* can be any admissible goal.

Definition 10. An *answer* for an admissible goal $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$ and a given $CFLP(\mathcal{D})$-program $\mathcal{P}$, must have the form $\Pi \square \theta$, where $\Pi \subseteq PCon(\mathcal{D})$ is a finite conjunction of atomic primitive constraints, $\theta \in Sub_\perp(\mathcal{U})$ is an idempotent substitution such that $dom(\theta) \cap var(\Pi) = \emptyset$, and there is some substitution $\theta' =_{\setminus evar(G)} \theta$ fulfilling the following conditions:

- $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\theta' \Leftarrow \Pi$,
- $\Pi \models_{\mathcal{D}} S\theta'$,
- $X\theta' \equiv t\theta'$ for each $X \mapsto t \in \sigma$, abbreviated as $\theta' \in Sol(\sigma)$.

A *witness* $\mathcal{M}$ for the fact that $\Pi \square \theta$ is an answer of G is defined as a multiset containing all the $CRWL(\mathcal{D})$-proofs mentioned above. We write $Ans_{\mathcal{P}}(G)$ for the set of all answers for G. An answer $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ is called *trivial* if $Unsat_{\mathcal{D}}(\Pi)$ and *non-trivial* otherwise.

Definition 11. Let $G \equiv \exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ be an admissible goal for a given $CFLP(\mathcal{D})$-program $\mathcal{P}$. We say that a valuation $\mu \in Val_\perp(\mathcal{D})$ is a *solution* of G if there is some valuation $\mu' =_{\setminus evar(G)} \mu$ satisfying the following conditions:

- $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\mu'$,
- $\models_{\mathcal{D}} S\mu'$ (i.e. $\mu' \in Sol_{\mathcal{D}}(S)$),
- $X\mu' \equiv t\mu'$ for each $X \mapsto t \in \sigma$, abbreviated as $\mu' \in Sol(\sigma)$.

We write $Sol_{\mathcal{P}}(G)$ for the set of all solutions for G. Analogously, we define the set of solutions for an answer $\Pi \,\square\, \theta$ as $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) =_{def} \{\mu \in Val_\perp(\mathcal{D}) \mid \mu \in Sol_{\mathcal{D}}(\Pi) \cap Sol(\theta)\}$.

From *Definition 10* and *Definition 11*, it is easy to prove that the notion of solution is a particular case of the notion of answer for a goal. More formally, if G is an admissible goal and $\mu \in Val_\perp(\mathcal{D})$ then $\mu \in Sol_{\mathcal{P}}(G) \Leftrightarrow \emptyset \,\square\, \mu \in Ans_{\mathcal{P}}(G)$. Another useful relationship between answers and solutions is given in the next proposition.

PROPOSITION 1. *Let $G \equiv \exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ be an admissible goal for a given $CFLP(\mathcal{D})$-program $\mathcal{P}$ and $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$ an answer for G. Then, $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{P}}(G)$. Furthermore, if G is in solved form then $S \,\square\, \sigma \in Ans_{\mathcal{P}}(G)$.*

PROOF. Since $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$, there is some θ' satisfying: (a) $\theta' =_{\setminus evar(G)} \theta$; (b) $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\theta' \Leftarrow \Pi$; (c) $\Pi \models_{\mathcal{D}} S\theta'$; and (d) $\theta' \in Sol(\sigma)$. For proving $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{P}}(G)$, we assume any valuation μ such that (1) $\mu \in Sol_{\mathcal{D}}(\Pi)$ and (2) $\mu \in Sol(\theta)$. Then, $\mu \in Sol_{\mathcal{P}}(G)$ holds because the valuation $\mu' = \theta'\mu$ verifies: (a') $\theta'\mu =_{\setminus evar(G)} \mu$, because of (a) and $\theta\mu = \mu$ (which follows from (2)); (b') $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\theta'\mu$, because of (a) and the *Entailment Property* from *Lemma 1* (note that $(P \,\square\, C)\theta' \Leftarrow \Pi \succcurlyeq_{\mathcal{D}} (P \,\square\, C)\theta'\mu$ follows from (1)); (c') $\theta'\mu \in Sol_{\mathcal{D}}(S)$, or equivalently, $\mu \in Sol_{\mathcal{D}}(S\theta')$, because of (1) and (c); and (d') $\theta'\mu \in Sol(\sigma)$, because of (d). For proving the second part of the proposition, let us assume that P and C are empty. Then, $S \,\square\, \sigma \in Ans_{\mathcal{P}}(G)$ holds because $\sigma' = \sigma$ trivially verifies $\sigma' =_{\setminus evar(G)} \sigma$, and also $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma \Leftarrow S$ (because $P \,\square\, C$ is empty); $S \models_{\mathcal{D}} S\sigma$ (because $S\sigma = S$); and $\sigma \in Sol(\sigma)$ (trivially). $\square$

The calculus $CLNC(\mathcal{D})$ consists of a set of transformation rules for admissible goals. Each transformation takes the form $G \Vdash G'$, specifying one of the possible ways of performing one step of goal solving. We write $G \Vdash_R G'$ to indicate that $G \Vdash G'$ by means of the $CLNC(\mathcal{D})$ transformation rule R. Derivations are sequences of $\Vdash$-steps. As in the case of constrained SLD derivations for $CLP(\mathcal{D})$ programs [18], successful derivations will eventually end with a solved goal. Failing derivations (ending with an obviously inconsistent goal $\blacksquare$) and infinite derivations are also possible. The goal transformation rules concerning productions (see *Figure 2*) are designed with the aim of modelling the behaviour of constrained lazy narrowing with *sharing*, but now involving primitive functions, possibly higher-order defined functions and functional variables. The notation $\overline{e_m \to t_m}$ abbreviates $e_1 \to t_1, \ldots, e_m \to t_m$. Some $CLNC(\mathcal{D})$ rules use the notation "[...]" meaning an optional part of a goal, present only under certain conditions. For example, in the **Simple Production** rules, if we have the condition $X \notin \overline{U}$ (and then $X \notin pvar(P)$ by the admissibility condition **EX**) we have the rule $\exists \overline{U}.\ X \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{SP_1} \exists \overline{U}.\ (P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma\sigma_0$, and the rule $\exists X, \overline{U}.\ X \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{SP_2} \exists \overline{U}.\ (P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma$ otherwise. Analogously for the **Functional Variable** rules. The goal transformation rules concerning constraints (see *Figure 3*) are designed to combine atomic (primitive or user defined) constraints with the action of a constraint solver that fulfill the requirements given in *Definition 5*. Failure rules in *Figure 3* are used for failure detection in constraint solving and failure detection in the syntactic unification of the produced part of the goal.

Important Convention: all the goal transformation rules are applied by viewing P and C as *sets*, rather than *sequences*. For example, rule $\mathbf{DF_1}$ allows to select an arbitrary production $f\overline{e}_n \to t$ occurring in the current goal.

Finally, the section is closed with two examples of goal solving which highlight the main properties of the $CLNC(\mathcal{D})$ calculus. At each goal transformation step, we underline which subgoal is selected.

Example 3. We compute all the answers from the goal $G_0 \equiv \exists Ys.\ from\ Y \to Ys \,\square\, X /= s(null\ \ Ys) \,\square\, \square\, \varepsilon$ using the $CFLP(\mathcal{H}_{seq})$-program of *Example 2* over the constraint domain $\mathcal{H}_{seq}$ of *Example 1*. We have the following $CLNC(\mathcal{H}_{seq})$ derivation starting from G_0:

$$\exists Ys.\ from\ Y \to Ys \,\square\, \underline{X /= s(null\ \ Ys)} \,\square\, \square\, \varepsilon \Vdash_{\mathbf{AC}}$$
$$\exists R, Ys.\ \underline{s(null\ \ Ys) \to R}, from\ Y \to Ys \,\square\, \square$$
$$\underline{X /= R \,\square\, \varepsilon} \Vdash_{\mathbf{IM}\{\mathbf{R} \,\mapsto\, \mathbf{s\ K}\}}$$
$$\exists K, Ys.\ null\ \ Ys \to K, from\ Y \to Ys \,\square\, \square$$
$$\underline{X /= s\ K} \,\square\, \varepsilon \Vdash_{\mathbf{CS}\{\mathbf{K}, Ys\}}$$

Now, the constraint solver (see *Appendix A*) gives two possible alternatives

$$Solve^{\mathcal{H}_{seq}}(\{X /= s\ K\}, \{K, Ys\}) = (\square\{X \mapsto 0\}) \vee (\{M /= K\} \,\square\, \{X \mapsto s\ M\})$$

and there are three possible continuations of the computation

1. $\exists K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys \,\square\, \square\ \square\{X \mapsto 0\} \Vdash_{\mathbf{EL}}$
   $\exists Ys.\ \underline{from\ Y \to Ys} \,\square\, \square\, \square\{X \mapsto 0\} \Vdash_{\mathbf{EL}}$
   $\square\, \square\, \square\{X \mapsto 0\}$
   computed answer: $S_1 \,\square\, \sigma_1 \equiv \square\{X \mapsto 0\}$.

2. $\exists M, K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys \,\square\, \square\ M /= K \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{DF}}$
   $\exists M, K, Ys.\ \underline{Ys \to [\,]}, \underline{s\ 0 \to K}, from\ Y \to Ys \,\square\, \square\ M /= K \,\square\, \{X \mapsto s\ M\} \Vdash^2_{\mathbf{SP}\{Ys \,\mapsto\, [\,], \mathbf{K} \,\mapsto\, \mathbf{s\ 0}\}}$
   $\exists M.\ \underline{from\ Y \to [\,]} \,\square\, \square\, M /= s\ 0 \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{DF}}$
   $\exists N, M.\ Y \to N, \underline{[N|from\ (s\ N)] \to [\,]} \,\square\, \square\ M /= s\ 0 \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{CF}} \blacksquare$

3. $\exists M, K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys \,\square\, \square\ M /= K \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{DF}}$
   $\exists U, Us, M, K, Ys.\ \underline{Ys \to [U|Us]}, \underline{0 \to K}, from\ Y \to Ys \,\square\, \square\ M /= K \,\square\, \{X \mapsto s\ M\} \Vdash^2_{\mathbf{SP}\{Ys \,\mapsto\, [\mathbf{U}|\mathbf{Us}], \mathbf{K} \,\mapsto\, \mathbf{0}\}}$
   $\exists U, Us, M.\ \underline{from\ Y \to [U|Us]} \,\square\, \square\ M /= 0 \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{DF}}$
   $\exists N, U, Us, M.\ \underline{Y \to N}, [N|from\ (s\ N)] \to [U|Us] \,\square\, \square\ M /= 0 \,\square\, \{X \mapsto s\ M\} \Vdash_{\mathbf{SP}\{\mathbf{N} \,\mapsto\, \mathbf{Y}\}}$

$\exists U, Us, M.\ \underline{[Y|from\ (s\ Y)] \to [U|Us]} \square\ \square$
$\quad M /= 0 \square \{X \mapsto s\ M\} \Vdash_{\mathbf{DC}}$
$\exists U, Us, M.\ \underline{Y \to U, from\ (s\ Y) \to Us} \square\ \square$
$\quad M /= 0 \square \{X \mapsto s\ M\} \Vdash^{2}_{\mathbf{EL}}$
$\exists M. \square\ \square\ M /= 0 \square \{X \mapsto s\ M\}$
computed answer: $S_2 \square \sigma_2 \equiv M /= 0 \square \{X \mapsto s\ M\}$.

For this example, is also possible to prove that $\Pi \square \theta \equiv X /= s\ 0\ \square\ \{Y \mapsto s\ Z\}$ is a correct answer of G_0 such that $Sol_{\mathcal{H}_{seq}}(\Pi \square \theta) \subseteq \bigcup_{i=1}^{2} Sol_{\mathcal{H}_{seq}}(S_i \square \sigma_i)$; but no *single* computed answer $S \square \sigma$ verifies $Sol_{\mathcal{H}_{seq}}(\Pi \square \theta) \subseteq Sol_{\mathcal{H}_{seq}}(S \square \sigma)$. We will see in *Theorem 2* that this is true in general.

Example 4. **Splitting a list.** The next example splits a list with only one element using the $CFLP(\mathcal{H}_{seq})$-program *split* given in *Example 2*.

$\square\ \underline{split\ [X] == (Xs, Ys)}\ \square\ \square\ \varepsilon \Vdash_{\mathbf{AC}}$
$\exists R_1.\ \underline{split\ [X] \to R_1}\ \square\ \square\ R_1 == (Xs, Ys)\ \square\ \varepsilon \Vdash^{*}_{\mathbf{DF}}$
$\exists Ys_1, Zs_1, R, R_1.\ \underline{case\ R\ X\ Ys_1\ Zs_1 \to R_1},$
$\quad \underline{split\ [\,] \to (Ys_1, Zs_1)}\ \square\ \square\ seq\ X\ s\ 0 \to!\ R,$
$\quad R_1 == (Xs, Ys)\ \square\ \varepsilon \Vdash_{\mathbf{DF}}$
$\exists Ys_1, Zs_1, R, R_1.\ case\ R\ X\ Ys_1\ Zs_1 \to R_1,$
$\quad \underline{([\,], [\,]) \to (Ys_1, Zs_1)}\ \square\ \square\ seq\ X\ s\ 0 \to!\ R,$
$\quad R_1 == (Xs, Ys)\ \square\ \varepsilon \Vdash^{*}_{\mathbf{DC},\mathbf{SP}\{\mathbf{Ys_1} \mapsto [\,],\ \mathbf{Zs_1} \mapsto [\,]\}}$
$\exists R, R_1.\ case\ R\ X\ [\,]\ [\,] \to R_1\ \square\ \square$
$\quad \underline{seq\ X\ s\ 0 \to!\ R, R_1 == (Xs, Ys)}\ \square\ \varepsilon \Vdash_{\mathbf{CS}\{\mathbf{R_1}\}}$

Now, the constraint solver over $\mathcal{H}_{seq}$ (see *Appendix A*) gives three possible alternatives

$Solve^{\mathcal{H}_{seq}}(\{seq\ X\ s\ 0 \to!\ R, R_1 == (Xs, Ys)\}, \{R_1\}) =$
$\quad (\{R_1 == (Xs, Ys)\} \square \{R \mapsto true, X \mapsto s\ 0\})\ \vee$
$\quad (\{R_1 == (Xs, Ys)\} \square \{R \mapsto false, X \mapsto 0\})\ \vee$
$\quad (\{R_1 == (Xs, Ys), M /= 0\} \square \{R \mapsto false, X \mapsto s\ M\})$

and there are three possible continuations of the computation, each of one with a computed answer associated

1. $\exists R_1.\ \underline{case\ true\ s\ 0\ [\,]\ [\,] \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys)\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{DF}}$
   $\exists R_1.\ \underline{([s\ 0], [\,]) \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys)\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{SP}}$
   $\square\ \square\ \underline{([s\ 0], [\,]) == (Xs, Ys)}\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{CS}\{\}}$
   $\square\ \square\ \square\ \{X \mapsto s\ 0, Xs \mapsto [s\ 0], Ys \mapsto [\,]\}$
   answer: $S_1 \square \sigma_1 \equiv \square \{X \mapsto s\ 0, \mathrm{Xs} \mapsto [s\ 0], \mathrm{Ys} \mapsto [\,]\}$.
2. $\exists R_1.\ \underline{case\ false\ 0\ [\,]\ [\,] \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys)\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{DF}}$
   $\exists R_1.\ \underline{([\,], [0]) \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys)\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{SP}}$
   $\square\ \square\ \underline{([\,], [0]) == (Xs, Ys)}\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{CS}\{\}}$
   $\square\ \square\ \square\ \{X \mapsto 0, Xs \mapsto [\,], Ys \mapsto [0]\}$
   answer: $S_2 \square \sigma_2 \equiv \square \{X \mapsto 0, \mathrm{Xs} \mapsto [\,], \mathrm{Ys} \mapsto [0]\}$.
3. $\exists M, R_1.\ \underline{case\ false\ s\ M\ [\,]\ [\,] \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys), M /= 0\ \square\ \{X \mapsto s\ M\} \Vdash_{\mathbf{DF}}$
   $\exists M, R_1.\ \underline{([\,], [s\ M]) \to R_1}\ \square\ \square$
   $R_1 == (Xs, Ys), M /= 0\ \square\ \{X \mapsto s\ M\} \Vdash_{\mathbf{SP}}$
   $\exists M.\ \square\ \square\ \underline{([\,], [s\ M]) == (Xs, Ys), M /= 0}\ \square$
   $\{X \mapsto s\ M\} \Vdash_{\mathbf{CS}\{\}}$
   $\exists M. \square\ \square\ M /= 0\ \square\ \{X \mapsto s\ M, Xs \mapsto [\,], Ys \mapsto [s\ M]\}$
   answer: $S_3 \square \sigma_3 \equiv M/= 0\ \square\ \{X \mapsto s\ M,\ \mathrm{Xs} \mapsto [\,],\ \mathrm{Ys} \mapsto [s\ M]\}$.

4. PROPERTIES OF $CLNC(\mathcal{D})$

This section presents the main results of the paper, namely soundness and completeness of goal solving in $CLNC(\mathcal{D})$ w.r.t. $CRWL(\mathcal{D})$ semantics. We emphasize the technical difficulty of the *Completeness Theorem 2*, harder to prove than related results for FLP languages [11, 12, 31] and also stronger and more general than previous related results for $CFLP$ languages [21, 2, 3]. As main differences w.r.t. the constrained lazy narrowing calculus for the $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{L})$ scheme [25], we provide a logical semantics for correct answers (*Definition 10*) and a formal notion of constraint solver (*Definition 5*) well suited to that semantics.

Our first result proves correctness of a single transformation step. It says that transformation steps preserve admissibility of goals, fail only in case of unsatisfiable goals and do not introduce new solutions.

LEMMA 2. **Correctness Lemma.**

1. *The transformation steps preserve admissibility of goals: If* $G \Vdash_{CLNC(\mathcal{D})} G'$ *and* G *is admissible, then* G' *is admissible. Moreover,* $fvar(G') \subseteq fvar(G)$.
2. *The transformation steps fail only in case of unsatisfiable goals: If* $G \Vdash_{CLNC(\mathcal{D})} \blacksquare$ *then* $Sol_{\mathcal{P}}(G) = \emptyset$ *(or equivalently,* $Ans_{\mathcal{P}}(G)$ *includes only trivial answers).*
3. *The transformation steps do not introduce new solutions: If* $G \Vdash_{CLNC(\mathcal{D})} G'$ *and* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$ *then* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

The following soundness result follows easily from the *Correctness Lemma*. It ensures that computed answers for a goal G are indeed correct answers of G.

THEOREM 1. **Soundness of** $CLNC(\mathcal{D})$**.**
If G_0 *is an initial goal and* $G_0 \Vdash^{*}_{CLNC(\mathcal{D})} G_n$*,where* $G_n \equiv \exists \overline{U}.\ \square\ \square S \square \sigma$ *is a solved goal, then* $S \square \sigma \in Ans_{\mathcal{P}}(G_0)$.

PROOF. From *Proposition 1* we get $S \square \sigma \in Ans_{\mathcal{P}}(G_n)$. Now, if we repeatedly backwards apply item 3. of the *Correctness Lemma*, we obtain $S \square \sigma \in Ans_{\mathcal{P}}(G_0)$. $\square$

Completeness of $CLNC(\mathcal{D})$ is based on the following idea: whenever $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ and G is not yet solved, there are finitely many local choices for a first computation step $G \Vdash G_j$ $(1 \leq j \leq l)$ so that the new goals G_j are "closer to be solved" and "cover all the solutions of $\Pi \square \theta$". This idea is made precise in the next lemma, which relies on a sophisticated well-founded progress ordering. A similar technique was used in [11, 12, 31] to prove completeness of lazy narrowing calculi for FLP languages. In the present $CFLP(\mathcal{D})$ setting, the solver $Solve^{\mathcal{D}}$ must be taken into account. As a consequence, the number l of local choices can be greater that 1 in general, and the progress ordering is more complicated than those used in [11, 12, 31].

LEMMA 3. **Progress Lemma.**
Assume an admissible goal G *not in solved form, and a witnessed non-trivial answer* $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$*. Then:*

1. *There is some* $CLNC(\mathcal{D})$ *transformation rule applicable to* G.

2. *For any $CLNC(\mathcal{D})$ rule R applicable to G, there exist l goals G_j with witnessed non-trivial answers $\mathcal{M}_j$: $\Pi_j \square \theta_j \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$ such that:*

   - $G \vdash_R G_j$ *for each* $1 \leq j \leq l$,
   - $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_j \square \theta_j)$,
   - $(G, \mathcal{M}) \rhd (G_j, \mathcal{M}_j)$ *for each* $1 \leq j \leq l$, *where* $\rhd$ *is the well-founded progress ordering defined in* Appendix B.

PROOF. (1) If $G \equiv \exists \overline{U}.\ P \square C \square S \square \sigma$ is an admissible goal not in solved form, then P or C 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 answer, due to item 2. in the *Correctness Lemma*. Assume that **AC** is not applicable. Then, C must be empty and the goal has the form $G \equiv \exists \overline{U}.\ P \square \square S \square \sigma$ with P not empty. Now assume that **DC**, **SP**, **IM**, **PF**, **DF** and **FV** are not applicable. Then it must be the case that all the production in P are of the form $h\overline{e}_m \rightarrow X$ or $f\overline{e}_n\overline{a}_k \rightarrow X$ $(k \geq 0)$ or $p\overline{e}_n \rightarrow X$ or $F\overline{a}_k \rightarrow X$ $(k > 0)$ where $h\overline{e}_m$ is a rigid and passive expression but not a pattern and in all cases X is a produced but not demanded variable, in particular $X \notin dvar_{\mathcal{D}}(S)$. Consider the set χ of such $X's$, that is, $\chi = pvar(G)$. If the rule **CS** is not applicable, S must be in χ-solved form. But then, due to the fact that $\chi \cap dvar_{\mathcal{D}}(S) = \emptyset$ and the requirement (a) of constraint solvers in *Definition 5*, we conclude $\chi \cap var(S) = \emptyset$. Choose now some $X \in \chi$ minimal in the $\gg_P^+$ relation (such minimal elements do exists, due to the finite number of variables occurring in G and the property **NC** of admissible goals). Such X cannot appear neither in any other approximation statement in P nor in the substitution σ of the goal by the admissibility condition **SL** and then verifies $X \notin var(P \square C \square S \square \sigma)$. Therefore, the rule **EL** can be applied to the production where X appears. (2) This can be proved by case analysis, using the *Table 1* given in *Appendix B*, which shows the behaviour of the different $CLNC(\mathcal{D})$ transformations w.r.t. the five components of the lexicographic progress ordering. Details are omitted here due to lack of space. □

Reiterated application of the previous lemma leads to the desired completeness result:

THEOREM 2. **Completeness of $CLNC(\mathcal{D})$.** *Let G_0 an initial admissible goal and $\Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ non-trivial. Then there exist a finite number of derivations ending in solved goals $G_0 \vdash^* G_i$ $(1 \leq i \leq k)$ such that $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$.*

PROOF. By repeated application of the *Progress Lemma 3*, we can build a finitely branching tree $\mathcal{T}$ with root $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ such that, each node $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$ associated to a goal G not in solved form, has children $\mathcal{M}_j : \Pi_j \square \theta_j \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$. Since $\rhd$ is a well-founded ordering, there are not infinite paths in $\mathcal{T}$. Because of *König's Lemma*, $\mathcal{T}$ is a finite tree with k leaves associated to solved goals G_i $(1 \leq i \leq k)$ such that $G_0 \vdash^* G_i$ for each $1 \leq i \leq k$. We prove $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$ by induction on the depth p of $\mathcal{T}$.

- *Base case* $(p = 0)$. $\mathcal{T}$ has only the root node $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$, where G_0 is a goal in solved form. In this case, $k = 1$, $G_1 \equiv G_0$ and directly $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq Sol_{\mathcal{P}}(G_0)$ using *Proposition* 1.

- *Inductive case* $(p > 0)$. $\mathcal{T}$ has a root node $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ and subtrees $\mathcal{T}_j$ $(1 \leq j \leq l)$, each of them with root $\mathcal{M}_j : \Pi_j \square \theta_j \in Ans_{\mathcal{P}}(G_j)$ and leaves associated to solved goals $G_{j,i}$ $(1 \leq i \leq k_j)$. We prove $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{j=1}^{l} \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. By applying the *Progress Lemma*, we have $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G_0}.\ \Pi_j \square \theta_j)$. Assume that $\overline{W}_j$ are the existentially quantified variables in $\exists_{\setminus G_0}.\ \Pi_j \square \theta_j$. By *induction hypothesis* for each $1 \leq j \leq l$ (the depth of each subtree $\mathcal{T}_j$ is $p_j < p$), $Sol_{\mathcal{D}}(\Pi_j \square \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. Moreover, since $\overline{W}_j$ are not free variables in $G_{j,i}$ for all $1 \leq i \leq k_j$, we also have $Sol_{\mathcal{D}}(\exists \overline{W}_j.\ \Pi_j \square \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$, and the result follows easily from both inclusions.

□

From the proof of *Theorem 2* we see that completeness is *strong* in the sense that the local choice of the goal transformation rule applied at each step can be a *don't care* choice. Moreover, *Example 3* shows that the number k of computed answers needed to cover the solutions of the given answer $\Pi \square \theta$ must be allowed to be greater than 1 in general. A similar situation occurs in Maher's completeness theorem for $CLP(\mathcal{D})$ [18], although the underlying semantics and proof techniques are quite different in that context. In our setting, item 2. in *Definition 5* (concerning the behaviour of constraint solvers) is responsible for the *finite* number k of computed answers in the completeness theorem.

5. CONCLUSIONS AND FUTURE WORK

We have presented a new constrained lazy narrowing calculus $CLNC(\mathcal{D})$ parameterized by a constraint domain $\mathcal{D}$, intended as a formal specification of a goal solving procedure for constraint functional logic programs in a recently proposed $CFLP(\mathcal{D})$ scheme [24]. $CLNC(\mathcal{D})$ relies on a new formal notion of constraint solver. It is sound and strongly complete w.r.t. the declarative semantics given in [24].

In the near future, we plan to investigate both improvements and applications of the $CFLP(\mathcal{D})$ scheme. Planned improvements include enriching the scheme with algebraic data constructors in the vein of [4] and the optimization of $CLNC(\mathcal{D})$ by means of definitional trees, extending the approach of [31]. Planned applications will focus on practical instances of the $CFLP(\mathcal{D})$ scheme, supporting arithmetic constraints over the real numbers and finite domain constraints. In particular, we plan to formalize the work started in [10] as an instance of the $CFLP(\mathcal{D})$ scheme, and to investigate practical constraint solving methods and applications of the resulting language.

Last but not least, we plan to extend the work on declarative debugging of functional logic programs started in [6, 7] to $CFLP(\mathcal{D})$-programs, considering the diagnosis of both wrong answers and missing answers, and implementing the resulting debugging methods for some practical instances of the $CFLP(\mathcal{D})$ scheme.

6. REFERENCES

[1] K.R. Apt. *Logic Programming*. In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, Chapter 10, Elsevier and The MIT Press, pp. 493–574, 1990.

[2] P. Arenas-Sánchez, A. Gil-Luezas and F.J. López-Fraguas. *Combining Lazy Narrowing with Disequality Constraints*. Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'94), Springer LNCS 844, pp. 385–399, 1994.

[3] P. Arenas-Sánchez, F.J. López-Fraguas and M. Rodríguez-Artalejo. *Functional plus Logic Programming with Built-in and Symbolic Constraints*. Proc. Int. Conf. on Principles and Practice of Declarative Programming (PPDP'99), Paris, Springer LNCS 1702, pp. 152–169, 1999.

[4] P. Arenas-Sánchez and M. Rodríguez-Artalejo. *A general framework for lazy functional logic programming with algebraic polymorphic types*. Theory and Practice of Logic Programming 1(2), pp. 185–245, 2001.

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

[6] R. Caballero and M. Rodríguez-Artalejo. *A Declarative Debugging System for Lazy Functional Logic Programs*. Electronic Notes in Theoretical Computer Science 64, 63 pages, 2002.

[7] R. Caballero and M. Rodríguez-Artalejo. $\mathcal{DDT}$: *A Declarative Debugging Tool for Functional Logic Languages*. To appear in Proc. of the 7th International Symposium on Functional and Logic Programming (FLOPS'2004), Springer LNCS.

[8] J. Darlington, Y.K. Guo and H. Pull. *A New Perspective on the Integration of Functional and Logic Languages*. Proc. of the Int. Conf. on Fifth Generation Computer Systems (FGCS'92), IOS Press, pp. 682–693, 1992.

[9] D. DeGroot and G. Lindstrom (eds.). *Logic Programming: Functions, Relations and Equations*. Prentice-Hall, Englewood Cliffs, 1986.

[10] A.J. Fernández, M.T. Hortalá-González and F. Sáenz Pérez. *Solving Combinatorial Problems with a Constraint Functional Logic Language*. Proc. 5th International Symposium on Principles and Practice of Declarative Languages (PADL'2003), Springer LNCS 2562, pp. 320–338, 2003.

[11] 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*. Journal of Logic Programming 40(1), pp. 47–87, 1999.

[12] J.C. González-Moreno, M.T. Hortalá-González and M. Rodríguez-Artalejo. *Polymorphic Types in Functional Logic Programming*. FLOPS'99 special issue of the Journal of Functional and Logic Programming, 2001. `http://danae.uni-muenster.de/lehre/kuchen/JFLP`.

[13] C.A. Gunter and D. Scott. *Semantic Domains*. in J.van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Elsevier and The MIT Press, Vol. B, Chapter 6, pp. 633–674, 1990.

[14] M. Hanus. *The Integration of Functions into Logic Programming: From Theory to Practice*. Journal of Logic Programming 19&20, pp. 583–628, 1994.

[15] M. Hanus (ed.), *Curry: an Integrated Functional Logic Language*, Version 0.8, April 15, 2003. `http://www-i2.informatik.uni-kiel.de/~curry/`.

[16] J. Jaffar and J.L. Lassez. *Constraint Logic Programming*. In Proc. ACM Symp. on Principles of Programming Languages (POPL'87), ACM Press, pp. 111–119, 1987.

[17] J. Jaffar and M.J. Maher. *Constraint Logic Programming: A Survey*. The Journal of Logic Programming 19&20, pp. 503–581, 1994.

[18] J. Jaffar, M.J. Maher, K. Marriott and P.J. Stuckey. *The Semantics of Constraint Logic Programs*. Journal of Logic Programming, 37 (1-3) pp. 1–46, 1998.

[19] J. Jaffar, S. Michaylov, P.J. Stuckey and R.H.C. Yap. *The CLP($\mathcal{R}$) Language and System*. ACM Transactions on Programming Languages and Systems, 14 (3) pp. 339–395, 1992.

[20] J.W. Lloyd. *Foundations of Logic Programming*. 2nd. ed., Springer Verlag, 1987.

[21] F.J. López-Fraguas. *A General Scheme for Constraint Functional Logic Programming*. Proc. Int. Conf. on Algebraic and Logic Programming (ALP'92), Springer LNCS 632, pp. 213–227, 1992.

[22] F.J.López Fraguas, J. Sánchez Hernández. $\mathcal{TOY}$: *A Multiparadigm Declarative System*. Proc. RTA'99, Springer LNCS 1631, pp 244–247, 1999.

[23] F.J. López-Fraguas and J. Sánchez-Hernández. *A Proof Theoretic Approach to Failure in Functional Logic Programming*. Theory and Practice of Logic Programming 4(1), pp. 41–74, 2004.

[24] F.J. López-Fraguas, M. Rodríguez-Artalejo and R. del Vado-Vírseda. *Constraint Functional Logic Programming Revisited*. To appear in Proc. of the 5th International Workshop on Rewriting Logic and its Applications (WRLA'2004), Electronic Notes in Theoretical Computer Science, 2004.

[25] M. Marin. *Functional Logic Programming with Distributed Constraint Solving*. Ph. D. Thesis, Johannes Kepler Universität Linz, 2000.

[26] M. Marin, T. Ida and W. Schreiner. *CFLP: a Mathematica Implementation of a Distributed Constraint Solving System*. In Third International Mathematical Symposium (IMS'99), Hagenberg, Austria, August 23–25, 10 pages, 1999.

[27] M. Marin, T. Ida and T. Suzuki. *Cooperative Constraint Functional Logic Programming*. In International Symposium on Principles of Software Evolution (IPSE'2000), pp. 223–230, November 1–2, 2000.

[28] K. Marriott and P.J. Stuckey. *Programming with Constraints, An Introduction*. The MIT Press, 1998.

[29] A. Middeldorp and E. Hamoen. *Completeness Results for Basic Narrowing*. Applicable Algebra in Engineering, Communications and Computing 5, pp. 213–253, 1994.

[30] J.A. Robinson ans E.E. Sibert. *LOGLISP: Motivation, Design and Implementation*. In K.L. Clark and S.A. Tärnlund (eds.), *Logic Programming*, Academic Press, pp. 299–313, 1982.

[31] R. del Vado-Vírseda. *A Demand-driven Narrowing Calculus with Overlapping Definitional Trees*. Proc. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'03), ACM Press, pp. 213–227, 2003.

[32] P. Van Hentenryck. *Constraint logic programming*. The Knowledge Engineering Review, Vol. 6:3, pp. 151–194, 1991.

APPENDIX

A. A CONSTRAINT SOLVER OVER $\mathcal{H}_{SEQ}$

Definition 12. **The Constraint Solver** $Solve^{\mathcal{H}_{seq}}$**.** We assume the constraint domain $\mathcal{H}_{seq}$ described in *Example 1.* Let $\chi \subseteq \mathcal{V}$ be a set of protected variables and $S \subseteq PCon(\mathcal{H}_{seq})$ a conjunction of atomic primitive constraints over $\mathcal{H}_{seq}$ of the form $S \equiv seq\ t_1\ s_1 \rightarrow!\ r_1, \ldots, seq\ t_n\ s_n \rightarrow! r_n$, where $t_i, s_i \in Pat(\emptyset)$ and $r_i \in \{true, false\} \cup \mathcal{V}$. We define a *constraint solver* for equality and disequality constraints as follows: $Solve^{\mathcal{H}_{seq}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i) \Leftrightarrow_{def} S \,\square\, \varepsilon \rightsquigarrow_\chi^* \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i) \not\rightsquigarrow_\chi$, where the relation $\rightsquigarrow_\chi$ defined below denotes one constraint solver step, and $\varphi \not\rightsquigarrow_\chi$ express that φ is not reducible by $\rightsquigarrow_\chi$.

The following rule system specifies the behaviour of the relation $\rightsquigarrow_\chi$ between constraint disjunctions of the form $\bigvee_i(S_i \,\square\, \sigma_i)$, such that each $S_i \subseteq PCon(\mathcal{H}_{seq})$, $\sigma_i \in Sub(\emptyset)$, $\chi \cap var(S_i) = \emptyset$ and $var(S_i) \cap dom(\sigma_i) = \emptyset$. When applying the rules for $\rightsquigarrow_\chi$, we ignore the order of S and we view $==$ and $/=$ as symmetric.

General rules for $\rightsquigarrow_\chi$

R0 $\ldots \vee S_i \,\square\, \sigma_i \vee \ldots \rightsquigarrow_\chi \ldots \vee \bigvee_j(S'_j \,\square\, \sigma'_j) \vee \ldots$
if $S_i \,\square\, \sigma_i \rightsquigarrow_\chi \bigvee_j(S'_j \,\square\, \sigma'_j)$.
R1 $seq\ t\ s \rightarrow! R, S \,\square\, \sigma \rightsquigarrow_\chi (t == s, S\theta_1 \,\square\, \sigma\theta_1) \vee (t /= s, S\theta_2 \,\square\, \sigma\theta_2)$
if $R \notin \chi$, $\theta_1 = \{R \mapsto true\}$ and $\theta_2 = \{R \mapsto false\}$.

Rules for strict equality

R2 $h\bar{t}_n == h\bar{s}_n, S \,\square\, \sigma \rightsquigarrow_\chi t_1 == s_1, \ldots, t_n == s_n, S \,\square\, \sigma$
R3 $X == t, S \,\square\, \sigma \rightsquigarrow_\chi Tot(t), S\theta \,\square\, \sigma\theta$
if $X \notin \chi \cup var(t)$, $var(t) \cap \chi = \emptyset$ and $\theta = \{X \mapsto t\}$.

Rules for strict disequality

R4 $h\bar{t}_n /= h\bar{s}_n, S \,\square\, \sigma \rightsquigarrow_\chi \bigvee_{i=1}^{n}(t_i /= s_i, S \,\square\, \sigma)$
R5 $h\bar{t}_n /= h'\bar{s}_m, S \,\square\, \sigma \rightsquigarrow_\chi S \,\square\, \sigma$ if $h \neq h'$ or $m \neq n$.
R6 $X /= h\bar{t}_n, S \,\square\, \sigma \rightsquigarrow_\chi (\bigvee_i(S\theta_i \,\square\, \sigma\theta_i)) \vee (\bigvee_{k=1}^{n}(U_k /= t_k\theta, S\theta \,\square\, \sigma\theta))$
if $X \notin \chi$, $var(\bar{t}_n) \cap \chi \neq \emptyset$, $\theta_i = \{X \mapsto h_i\overline{Y}_{in_i}\}$ for each $h_i \neq h$ and $\theta = \{X \mapsto h\overline{U}_n\}$ with $\overline{Y}_{in_i}, \overline{U}_n$ new variables.

(In practice, **R6** *is used by limiting the choice of* $h_i\overline{Y}_{in_i}$ *to patterns of the same type as* X*. Therefore, the number* n *of choices is finite.)*

Failure rules

R7 $h\bar{t}_n == h'\bar{s}_m, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$ if $h \neq h'$ or $m \neq n$.
R8 $X == t, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$ if $X \not\equiv t$ and $X \in var(t)$.
R9 $X /= X, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$

Calculation of demanded variables in $\mathcal{H}_{seq}$

The following rules serve to compute the set $dvar_{\mathcal{H}_{seq}}(S)$ of demanded variables by a satisfiable set of primitive constraints $S \subseteq PCon(\mathcal{H}_{seq})$.

$dvar(seq\ t\ s \rightarrow! R, S) = \{R\} \cup (dvar_{\mathcal{H}_{seq}}(t == s, S\theta_1) \cap dvar(t /= s, S\theta_2))$
if R is a variable, $\theta_1 = \{R \mapsto true\}$ and $\theta_2 = \{R \mapsto false\}$.

$dvar(h\bar{t}_n == h\bar{s}_n, S) = dvar(t_1 == s_1, \ldots, t_n == s_n, S)$

$dvar(X == t, S) = \{X\} \cup var(t) \cup dvar(S\theta)$
where $\theta = \{X \mapsto t\}$.

$dvar(h\bar{t}_n /= h'\bar{s}_m, S) = dvar(S)$, if $h \neq h'$ or $n \neq m$.

$dvar(h\bar{t}_n /= h\bar{s}_n, S) = dvar(t_1 /= s_1, S) \cap \ldots \cap dvar(t_n /= s_n, S)$

$dvar(X /= Y, S) = \{X, Y\} \cup dvar(S)$
$dvar(X /= t, S) = \{X\} \cup dvar(S)$, if t is not a variable.

It is assumed that the last two rules are not applied if one of the previous rule is applicable. Otherwise, the computation of $dvar(S)$ does not work properly with these rules.

B. PROGRESS ORDERING

Definition 13. **Well-founded progress ordering.** Let $\mathcal{P}$ be a $CLNC(\mathcal{D})$-program, $G \equiv \exists\overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ an admissible goal for $\mathcal{P}$ and $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$ a witnessed answer. We define the following sizes associated to G and $\mathcal{M}$:

- The *restricted size of the witness* $\mathcal{M} = \{\!\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}\!\}$ (represented by $|\ \mathcal{M}\ |$) is the multiset of natural numbers $\{\!\{|\ \mathcal{T}_1\ |, \ldots, |\ \mathcal{T}_n\ |\}\!\}$, where $|\ \mathcal{T}_i\ |$ denotes the restricted size of each proof tree $\mathcal{T}_i$ $(1 \leq i \leq n)$, as defined in *Subsection 2.4.*
- The size $|\ G\ |_1$ is the number of occurrences in G of expressions $F\bar{e}_k$ with F a variable and $k > 0$.
- The size $|\ G\ |_2$ is the number of occurrences in G of rigid and passive expressions $h\bar{e}_n$ that are not patterns.
- The size $|\ G\ |_3$ is the total syntactic size of the right hand sides of productions in G.
- The *restricted size of the constraint store* (represented by $|\ S\ |$) is evaluated to 1 if S is in solved form and 0 in other case.

Over pairs $(G, \mathcal{M})$ we define a *well-founded progress ordering*

$$(G, \mathcal{M}) \rhd (G', \mathcal{M}') \Leftrightarrow_{def} (|\ \mathcal{M}\ |, |\ G\ |_1, |\ G\ |_2, |\ G\ |_3, |\ S\ |) \succ_{lex} (|\ \mathcal{M}'\ |, |\ G'\ |_1, |\ G'\ |_2, |\ G'\ |_3, |\ S'\ |)$$

where $\succ_{lex}$ is the lexicographic product of $\succ_{mul} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}}$, $\succ_{mul}$ is the multiset order for multisets over $\mathbb{N}$, and $>_{\mathbb{N}}$ is the usual ordering over $\mathbb{N}$. See [5] for definitions of these notions.

The following table shows the behaviour of the different $CLNC(\mathcal{D})$ transformations w.r.t. the five components of the lexicographic progress ordering.

RULE	$\|\ \mathcal{M}\ \|$	$\|\ G\ \|_1$	$\|\ G\ \|_2$	$\|\ G\ \|_3$	$\|\ S\ \|$
DC	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
SP	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
IM	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$		
EL	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
PF	$\succ_{\mathbf{mul}}$				
DF	$\succ_{\mathbf{mul}}$				
FV	$\succeq_{mul}$	$>_{\mathbb{N}}$			
CS	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$
AC	$\succ_{\mathbf{mul}}$				

Table 1: progress ordering