Higher-Order and Symbolic Computation manuscript No.
(will be inserted by the editor)

A New Generic Scheme for Functional Logic Programming with Constraints

Francisco J. López Fraguas · Mario Rodríguez Artalejo · Rafael del Vado Vírseda

Received: 31 January 2005 / Accepted: 23 August 2005

Abstract In this paper we propose a new generic scheme $CFLP(\mathscr{D})$, intended as a logical and semantic framework for lazy Constraint Functional Logic Programming over a parametrically given constraint domain $\mathscr{D}$. As in the case of the well known $CLP(\mathscr{D})$ scheme for Constraint Logic Programming, $\mathscr{D}$ is assumed to provide domain specific data values and constraints. $CFLP(\mathscr{D})$ programs are presented as sets of constrained rewrite rules that define the behavior of possibly higher order and/or non-deterministic lazy functions over $\mathscr{D}$. As a main novelty w.r.t. previous related work, we present a Constraint Rewriting Logic $CRWL(\mathscr{D})$ which provides a declarative semantics for $CFLP(\mathscr{D})$ programs. This logic relies on a new formalization of constraint domains and program interpretations, which allows a flexible combination of domain specific data values and user defined data constructors, as well as a functional view of constraints.

Keywords Functional Programming · Logic Programming · Constraints

This research has been partially supported by the Spanish National Projects MELODIAS (TIC2002-01167), MERIT-FORMS (TIN2005-09207-C03-03) and PROMESAS-CAM (S-0505/TIC/0407).

Francisco J. López Fraguas
Departamento de Sistemas Informáticos y Programación. Facultad de Informática de la Universidad Complutense de Madrid. C/ Prof. José García Santesmases, s/n. 28040 Madrid (Spain).
Tel.: +34-91-3947630
Fax: +34-91-3947529
E-mail: fraguas@sip.ucm.es

Mario Rodríguez Artalejo
Departamento de Sistemas Informáticos y Programación. Facultad de Informática de la Universidad Complutense de Madrid. C/ Prof. José García Santesmases, s/n. 28040 Madrid (Spain).
Tel.: +34-91-3947562
Fax: +34-91-3947529
E-mail: mario@sip.ucm.es

Rafael del Vado Vírseda
Departamento de Sistemas Informáticos y Programación. Facultad de Informática de la Universidad Complutense de Madrid. C/ Prof. José García Santesmases, s/n. 28040 Madrid (Spain).
Tel.: +34-91-3947625
Fax: +34-91-3947529
E-mail: rdelvado@sip.ucm.es

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 [54], where the *CLP* scheme was first introduced. The aim of the scheme was to define a family of constraint logic programming languages $CLP(\mathscr{D})$ parameterized by a constraint domain $\mathscr{D}$, in such a way that the well established results on the declarative and operational semantics of logic programs [62,5] could be lifted to all the $CLP(\mathscr{D})$ languages in an elegant and uniform way. The best updated presentation of the classical *CLP* semantics can be found in [56]. 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 [99,100,55,77].

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 [86]. Some other early proposals for the design of functional + logic languages are described in [29]. A more recent survey of the operational principles and implementation techniques used for the integration of functions into logic programming can be found in [46]. *Narrowing*, a natural combination of rewriting and unification, originally proposed as a theorem proving tool [93,61,35,50], has been used as a goal solving mechanism in functional logic languages such as *Curry* [47,48] and *TOY* [69,1]. Under various more or less restrictive conditions, several narrowing strategies are known to be complete for goal solving [31,46,80].

To our best knowledge, the first attempt of combining constraint logic programming and functional logic programming was the $CFLP(\mathscr{D})$ scheme proposed by J. Darlington, Y.K. Guo and H. Pull [27,28]. The idea behind this approach can be roughly described by the equation $CFLP(\mathscr{D}) = CLP(FP(\mathscr{D}))$, intended to mean that a $CFLP$ language over the constraint domain $\mathscr{D}$ is viewed as a *CLP* language over an extended constraint domain $FP(\mathscr{D})$ whose constraints include equations between expressions involving user defined functions, to be solved by narrowing. Other proposals concerning the combination of constraints with functional programming, equational deduction and lambda-calculus appeared around the same time [25,26,58,83,73].

The $CFLP$ scheme proposed by F.J. López-Fraguas in [64,65] tried to provide results on the declarative semantics of $CFLP(\mathscr{D})$ programs closer to those known for *CLP*. In the classical approach to *CLP* semantics a constraint domain is viewed as a first order structure $\mathscr{D}$, and constraints are viewed as first order formulas that can be interpreted in $\mathscr{D}$. In [64,65] programs were built as sets of constrained rewrite rules. In order to support a lazy semantics for the user defined functions, constraint domains $\mathscr{D}$ were formalized as continuous structures, with a Scott domain [91,45] 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 [74]. This work introduces $CFLP(\mathscr{D}, S, L)$, a family of languages parameterized by a constraint domain $\mathscr{D}$, a strategy S which defines the cooperation of several constraint solvers over $\mathscr{D}$, and a constraint lazy narrowing calculus 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 [75,76]. Its main limitation from our viewpoint is the lack of declarative semantics.

Our aim in this paper is to propose a new *CFLP* scheme which provides a clean declarative semantics for $CFLP(\mathscr{D})$ languages, as in the *CLP* scheme, and also overcomes the limitations of the older *CFLP* scheme in [64,65]. The main novelties of the current proposal are a new formalization of constraint domains for *CFLP*, a new notion of interpretation for *CFLP* programs, and a new *Constraint Rewriting Logic* $CRWL(\mathscr{D})$ parameterized by a constraint domain, which provides a logical characterization of program semantics.

$CRWL(\mathscr{D})$ is a natural extension of the Constructor-based ReWriting Logic *CRWL*, originally proposed in [41,42] as a logical framework for first order functional logic programming languages based on lazy and possibly non-deterministic functions, whose semantics cannot be directly described in terms of equational logic. Early work on *CRWL* was inspired by Hussmann's work on nondeterminism in algebraic specifications and programs [51–53]. In comparison to Meseguer's Rewriting Logic [79,78], originally aimed as a unified logic and semantic framework for concurrent languages and systems, *CRWL* shows clear differences in objectives and motivation. A careful comparison of both approaches has been worked out by M. Palomino in [84,85], showing that the semantics of both logics, when viewed as institutions, are formally incomparable.

In the last years, various extensions of *CRWL* have been devised, to account for various features of functional logic languages, such as higher order functions [43], polymorphic types [44], algebraic data constructors [10–12], an ad-hoc treatment of certain kinds of constraints [8,9], and finite failure [70,71]. A survey of previous work on *CRWL* can be found in [87]. A generic extension of *CRWL* with constraint reasoning was missing up to now.

Constraint functional logic programming obviously falls within the wider field of *Multiparadigm Constraint Programming*. Giving a survey of the many interesting research activities in this area lies outside the scope of the present paper. Here we just mention *Concurrent Constraint Programming* [88–90] as a particularly relevant subject which arose from the interplay between concurrent extensions of logic programming languages and the *CLP* scheme, and has inspired the design of various declarative languages [49,103,102]. Our *CFLP* scheme, however, does not deal with concurrency issues.

Operational semantics is another important topic which lies outside the scope of this paper. However, the very short Subsection 3.2 has been included in order to provide a few essential pointers to the literature.

The reader is assumed to have some knowledge on the foundations of logic programming [62,5] and term rewriting [30,59,13]. The rest of the paper is organized as follows: Section 2 presents a new formalization of constraint domains $\mathscr{D}$, tailored to the needs of constraint functional logic programming. Section 3 presents $CFLP(\mathscr{D})$ programs and their interpretations, along with results concerning the existence of least program models. Section 4 introduces the constraint rewriting logic $CRWL(\mathscr{D})$, presenting an inference system as well as correctness results w.r.t. the model-theoretic semantics given in the previous section. The short Section 5 summarizes the results of the paper and points to some possible lines of future work. Appendix A illustrates the practical realizability of the *CFLP* paradigm by presenting a small collection $CFLP(\mathscr{D})$ programs written in the concrete syntax of the *TOY* language, for several choices of the constraint domain $\mathscr{D}$. This appendix can be safely skipped by readers who are interested only in the more theoretical stuff dealt with in the main core of the paper. Finally, Appendix B includes some technical proofs that have been moved away from the main text in order to ease reading.

This article is a carefully revised and expanded version of the WRLA'2004 paper [66]. The main novelties w.r.t. the WRLA paper can be summarized as follows: Subsection 2.2 includes a new example, showing that finite domain constraints can be formalized as a constraint domain $\mathscr{FD}$ in our framework. New programming examples using this domain have been inserted in Section 3 and Appendix A. A new Subsection 3.2 has been introduced, providing some essential pointers to the literature on goal solving methods. References to this new subsection have been added at several places, in order to provide better motivation for various notions and results. Subsection 3.4 includes a new theorem which characterizes the relationship between the least weak models and the least strong models of $CFLP(\mathscr{D})$ programs. This new theorem has been used for improving the proofs of the *Correctness Results for Weak Semantics* presented in Subsection 4.2. Finally, Subsection 4.1 includes a new example which provides a detailed illustration of proof trees in $CRWL(\mathscr{D})$.

2 Constraint Domains

As already explained, one main aim in this paper is to overcome the limitations of our older $CFLP(\mathscr{D})$ scheme [64,65]. As a first step in this direction, we propose a new view of constraint domains $\mathscr{D}$ as structures with carrier set $GPat_\bot(\mathscr{U})$, consisting of ground patterns built from the symbols in a universal signature Σ and a set of urelements $\mathscr{U}$. 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 $CLP(\mathscr{R})$ [57], while symbols in Σ are intended to represent data constructors (e.g. the list constructor), domain specific primitive functions (e.g. addition and multiplication over $\mathbb{R}$), and user defined functions. Assuming a unique universal signature rather than various domain-dependent signatures turns out to be convenient for technical reasons.

Another important limitation of our older $CFLP(\mathscr{D})$ scheme [64,65], namely the lack of a type system, can be easily overcome by adopting the approach of [44], which shows how to refine a $CRWL$-based semantics for untyped programs with a polymorphic type system in Damas-Milner's style [81,24]. In this paper, however, we refrain from an explicit treatment of types, except for showing type declarations in some concrete programming examples.

The rest of this section gives a formal presentation of constraint domains. We begin by introducing the syntax of applicative expressions and patterns, which is needed for understanding our construction of constraint domains.

2.1 Applicative Expressions, Patterns and Substitutions

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*, indexed by arities. As we will see later on, evaluable functions can be further classified into domain dependent *primitive functions* and user given *defined functions*. We write $\Sigma_\bot$ for the result of extending DC^0 with the special symbol $\bot$, 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. In a typed language, *success* would represent the single data value in a singleton datatype (such as the unit type in SML or Haskell); while $true$ and $false$ would represent the two data values in a boolean datatype.

Next we assume a countably infinite set $\mathscr{V}$ of *variables* $X,Y,\ldots$ and a set $\mathscr{U}$ of urelements $u,v,\ldots$, mutually disjoint and disjoint from $\Sigma_\perp$. *Partial expressions* $e \in Exp_\perp(\mathscr{U})$ have the following syntax:

$$e ::= X\ (X \in \mathscr{V}) \mid \perp \mid u\ (u \in \mathscr{U}) \mid h\ (h \in DC \cup FS) \mid (e\, e_1)$$

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 $X \in \mathscr{V}$ and $m \geq 0$, is called a *flexible expression*, while $u \in \mathscr{U}$ 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. Any pattern is either a variable or a passive rigid expression. Intuitively, reducing an expression at the root makes sense only if the expression is active. This idea will play a role in the semantics presented in sections 3 and 4.

Some interesting subsets of $Exp_\perp(\mathscr{U})$ are:

- $GExp_\perp(\mathscr{U})$, the set of the *ground* expressions e such that $var(e) = \emptyset$.
- $Exp(\mathscr{U})$, the set of the *total* expressions e with no occurrences of $\perp$.
- $GExp(\mathscr{U})$, the set of the ground and total expressions $GExp_\perp(\mathscr{U}) \cap Exp(\mathscr{U})$.

Another important subclass of expressions is the set of partial patterns $s,t \in Pat_\perp(\mathscr{U})$, whose syntax is defined as follows:

$$\begin{array}{c} t ::= X\ (X \in \mathscr{V}) \mid \perp \mid u\ (u \in \mathscr{U}) \mid \\ (c\,\overline{t}_m)\ (c \in DC^n, m \leq n) \mid (f\,\overline{t}_m)\ (f \in FS^n, m < n) \end{array}$$

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 $f \in FS^n$, $m < n$, are used in *CRWL* [43,44] as a convenient representation of higher order values. The subsets $Pat(\mathscr{U})$, $GPat_\perp(\mathscr{U})$, $GPat(\mathscr{U}) \subseteq Pat_\perp(\mathscr{U})$ consisting of the *total*, *ground* and *ground and total* patterns, respectively, are defined in the natural way.

Following the spirit of denotational semantics [91,45], we view $Pat_\perp(\mathscr{U})$ as the set of finite elements of a semantic domain, and we define the *information ordering* $\sqsubseteq$ as the least partial ordering over $Pat_\perp$ satisfying the following properties: $\perp \sqsubseteq t$ for all $t \in Pat_\perp(\mathscr{U})$, and $(h\,\overline{t}_m) \sqsubseteq (h\,\overline{t'}_m)$ whenever these two expressions are patterns and $t_i \sqsubseteq t'_i$ for all $1 \leq i \leq m$. In the sequel, $\overline{t}_m \sqsubseteq \overline{t'}_m$ will be understood as meaning that $t_i \sqsubseteq t'_i$ for all $1 \leq i \leq m$. Note that a pattern $t \in Pat_\perp(\mathscr{U})$ is maximal w.r.t. the information ordering iff t is a total pattern, i.e. $t \in Pat(\mathscr{U})$.

Any partially ordered set (shortly, poset), can be converted into a semantic domain by means of a technique called *ideal completion*; see e.g. [82]. Therefore, in the rest of this paper we will use the poset $GPat_\perp(\mathscr{U})$ as an implicit representation of the semantic domain resulting from its ideal completion. This is consistent with the use of Scott domains in the semantics of the older $CFLP(\mathscr{D})$ scheme [64,65].

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 $Exp_\perp(\mathscr{U})$ which verifies $\perp \sqsubseteq e$ for all $e \in Exp_\perp(\mathscr{U})$, and $(e\,e_1) \sqsubseteq (e'\,e'_1)$ whenever $e \sqsubseteq e'$ and $e_1 \sqsubseteq e'_1$.

As usual, we define *substitutions* $\sigma \in Sub_\perp(\mathscr{U})$ as mappings $\sigma : \mathscr{V} \to Pat_\perp(\mathscr{U})$ extended to $\sigma : Exp_\perp(\mathscr{U}) \to Exp_\perp(\mathscr{U})$ in the natural way. Similarly, we consider *total substitutions* $\sigma \in Sub(\mathscr{U})$ given by mappings $\sigma : \mathscr{V} \to Pat(\mathscr{U})$, *ground substitutions* $\sigma \in GSub_\perp(\mathscr{U})$ given by mappings $\sigma : \mathscr{V} \to GPat_\perp(\mathscr{U})$, and *ground total substitutions* $\sigma \in GSub(\mathscr{U})$ given by mappings $\sigma : \mathscr{V} \to GPat(\mathscr{U})$. 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 $e \in Exp_\perp(\mathscr{U})$. We define the *domain* and the *variable range* of a substitution in the usual way, namely:

$$dom(\sigma) = \{X \in \mathscr{V} \mid \sigma(X) \neq X\} \qquad ran(\sigma) = \bigcup_{X \in dom(\sigma)} var(\sigma(X))$$

As usual, a substitution σ such that $dom(\sigma) \cap ran(\sigma) = \emptyset$ is called *idempotent*. For any set of variables $\mathscr{X} \subseteq \mathscr{V}$ we define the *restriction* $\sigma \upharpoonright \mathscr{X}$ as the substitution σ' such that $dom(\sigma') = \mathscr{X}$ and $\sigma'(X) = \sigma(X)$ for all $X \in \mathscr{X}$. We use the notation $\sigma =_\mathscr{X} \theta$ to indicate that $\sigma \upharpoonright \mathscr{X} = \theta \upharpoonright \mathscr{X}$, and we abbreviate $\sigma =_{\mathscr{V} \setminus \mathscr{X}} \theta$ as $\sigma =_{\setminus \mathscr{X}} \theta$. Finally, we consider two different ways of comparing given substitutions $\sigma, \sigma' \in Sub_\perp(\mathscr{U})$:

- σ is said to be less particular than σ' over $\mathscr{X} \subseteq \mathscr{V}$ (in symbols, $\sigma \leq_\mathscr{X} \sigma'$) iff $\sigma\theta =_\mathscr{X} \sigma'$ for some $\theta \in Sub_\perp(\mathscr{U})$. The notation $\sigma \leq \sigma'$ abbreviates $\sigma \leq_\mathscr{V} \sigma'$.
- σ is said to bear less information than σ' over $\mathscr{X} \subseteq \mathscr{V}$ (in symbols, $\sigma \sqsubseteq_\mathscr{X} \sigma'$) iff $\sigma(X) \sqsubseteq \sigma'(X)$ for all $X \in \mathscr{X}$. The notation $\sigma \sqsubseteq \sigma'$ abbreviates $\sigma \sqsubseteq_\mathscr{V} \sigma'$.

2.2 A New Formalization of Constraint Domains

Intuitively, a constraint domain is expected to provide a set of specific data elements, along with certain primitive functions and predicates operating upon them. Primitive predicates can be viewed as primitive functions returning boolean values. Therefore, we just consider primitive functions, and we formalize the notion of constraint domain as follows:

Definition 1 Constraint Domains.

1. A *constraint signature* is any family $\Gamma = \bigcup_{n \in \mathbb{N}} PF^n_\Gamma$ of primitive function symbols p indexed by arities, such that $PF^n_\Gamma \subseteq FS^n$ for each $n \in \mathbb{N}$. We will usually write PF^n in place of PF^n_Γ, leaving Γ implicit.
2. A *constraint domain* of signature Γ is any structure
$$\mathscr{D} = \langle D_\mathscr{U}, \{p^\mathscr{D} \mid p \in PF\}\rangle$$
such that the carrier set $D_\mathscr{U} = GPat_\perp(\mathscr{U})$ coincides with the set of ground patterns for some set of urelements $\mathscr{U}$, and the interpretation $p^\mathscr{D}$ of each $p \in PF^n$ satisfies the following requirements:
   (a) $p^\mathscr{D} \subseteq D^n_\mathscr{U} \times D_\mathscr{U}$, which boils down to $p^\mathscr{D} \subseteq D_\mathscr{U}$ in the case $n = 0$. In the sequel we always write $p^\mathscr{D}\,\bar{t}_n \to t$ to indicate that $(\bar{t}_n, t) \in p^\mathscr{D}$. In the case $n = 0$, this notation boils down to $p^\mathscr{D} \to t$.
   (b) $p^\mathscr{D}$ behaves monotonically in its arguments and antimonotonically in its result; i.e., whenever $p^\mathscr{D}\,\bar{t}_n \to t$, $\bar{t}_n \sqsubseteq \overline{t'}_n$ and $t \sqsupseteq t'$ one also has $p^\mathscr{D}\,\overline{t'}_n \to t'$.
   (c) $p^\mathscr{D}$ behaves radically in the following sense: whenever $p^\mathscr{D}\,\bar{t}_n \to t$ and $t \neq \perp$, there is some total $t' \in D_\mathscr{U}$ such that $p^\mathscr{D}\,\bar{t}_n \to t'$ and $t' \sqsupseteq t$.

Items 2.(a),(b) in the previous definition are intended to ensure that $p^{\mathscr{D}}$ encodes the behavior of a monotonic and continuous mapping from $\overline{D_{\mathscr{U}}}^n$, the *nth* power of the semantic domain obtained from $D_{\mathscr{U}}$ by ideal completion [82] into Hoare's Powerdomain $\mathscr{HP}(\overline{D_{\mathscr{U}}})$ [91,104,45]. Intuitively, one can think of $p^{\mathscr{D}}$ just as describing the behavior of a possibly non-deterministic function over finite data elements. The kind of non-determinism involved here is borrowed from our previous work on the *CRWL* framework [42,44,12], which in turn was inspired by ideas from Hussmann [51–53].

Item 2.(c), requiring primitive functions to be *radical*, is more novel and important for our present purposes. Requiring primitives to be radical just means that for given arguments, they are expected to return a total result, unless the arguments bear too little information for returning any result different from $\bot$. As far as we know, all the primitive functions used in practical constraint domains are radical in this sense.

Let us illustrate the previous definition by means of some examples. First we present two primitives for equality comparisons. They make sense for any constraint domain $\mathscr{D}$ built over any set of urelements $\mathscr{U}$, and are obviously radical:

Example 1 Two equality primitives:

1. $eq_{\mathscr{U}}$, equality primitive for urelements, interpreted to behave as follows: $eq^{\mathscr{D}}_{\mathscr{U}}\ u\ u \to true$ for all $u \in \mathscr{U}$; $eq^{\mathscr{D}}_{\mathscr{U}}\ u\ v \to false$ for all $u, v \in \mathscr{U}, u \neq v$; $eq^{\mathscr{D}}_{\mathscr{U}}\ t\ s \to \bot$ otherwise.
2. seq, strict equality primitive for ground patterns, interpreted to behave as follows: $seq^{\mathscr{D}}\ t\ t \to true$ for all total $t \in GPat(\mathscr{U})$; $seq^{\mathscr{D}}\ t\ s \to false$ for all $t, s \in GPat_{\bot}(\mathscr{U})$ such that t, s have no common upper bound w.r.t. the information ordering; $seq^{\mathscr{D}}\ t\ s \to \bot$ otherwise.

In the sequel we write $\mathscr{H}_{seq}$ to denote the constraint domain built over the empty set of urelements, and having $seq^{\mathscr{D}}$ as its only primitive. The language $CFLP(\mathscr{H}_{seq})$ can be seen as a new foundation for our previous work on functional logic programming with disequality constraints [60,7,68]. On the other hand, $\mathscr{H}_{seq}$ bears some analogy to the extension of the Herbrand domain with disequality constraints, introduced by A. Colmerauer [22,23] as one of the first constraint extensions of logic programming, and later investigated by M. J. Maher [72]. Some important differences must be noted, however. Firstly, the carrier set of $\mathscr{H}_{seq}$ is a poset of ground partial patterns, including representations of higher order values; while the carrier set in Colmerauer's approach consists of possibly infinite rational trees which cannot be interpreted as higher order values. Secondly, equality and disequality constraints were based on two different predicates in Colmerauer's approach, while in $\mathscr{H}_{seq}$ one single boolean valued primitive function allows to express strict equality and disequality constraints, as we will see in the next subsection. More generally, constraints in the $CFLP(\mathscr{D})$ scheme are always expressed by means of primitive functions with radical semantics.

The next example presents a constraint domain $\mathscr{R}$ similar to the one used in the well known constraint logic language $CLP(\mathscr{R})$ [57,55,77]. In the *CLP* case, the carrier set of $\mathscr{R}$ is defined as the set of all possible ground terms built from real numbers and data constructors, while we use a strictly bigger poset of partial ground patterns.

Example 2 The constraint domain $\mathscr{R}$ has the carrier set $D_{\mathbb{R}} = GPat_{\bot}(\mathbb{R})$ and the radical primitives defined below. We apply some of them in infix notation for convenience.

1. $eq_{\mathbb{R}}$, equality primitive for real numbers, interpreted as in Example 1.

2. *seq*, strict equality primitive for ground patterns over the real numbers, interpreted as in Example 1.
3. $+, *$, for addition and multiplication, interpreted to behave as follows:
   $x +^{\mathscr{R}} y \to x +^{\mathbb{R}} y$ for all $x, y \in \mathbb{R}$; $t +^{\mathscr{R}} s \to \bot$ whenever $t \notin \mathbb{R}$ or $s \notin \mathbb{R}$; and analogously for $*^{\mathscr{R}}$.
4. $<, \leq, >, \geq$, for numeric comparisons, interpreted to behave as follows:
   $x <^{\mathscr{R}} y \to true$ for all $x, y \in \mathbb{R}$ with $x <^{\mathbb{R}} y$; $x <^{\mathscr{R}} y \to false$ for all $x, y \in \mathbb{R}$ with $x \geq^{\mathbb{R}} y$; $t <^{\mathscr{R}} s \to \bot$ whenever $t \notin \mathbb{R}$ or $s \notin \mathbb{R}$; and analogously for $\leq^{\mathscr{R}}, >^{\mathscr{R}}, \geq^{\mathscr{R}}$.

Other constraint domains known for their practical value in constraint programming include feature tree constraints [4,94,14,15], which can be viewed as an extension of Colmerauer's rational trees [22,23], and finite domain constraints [99–102]. These two kinds of constraints play an important role in the multiparadigm programming language *Oz* [49, 103].

Finite domain constraints have been recently used for solving combinatorial problems in constraint functional logic programming [36], using an extension of the *TOY* system [37, 38]. The following example is intended as a formalization of the constraint domain used in [36].

Example 3 The constraint domain $\mathscr{FD}$ has the carrier set $D_{\mathbb{Z}} = GPat_{\bot}(\mathbb{Z})$ and the radical primitives defined below. We apply some of them in infix notation for convenience.

1. $eq_{\mathbb{Z}}$, equality primitive for integer numbers, interpreted as in Example 1.
2. *seq*, strict equality primitive for ground patterns over the integer numbers, interpreted as in Example 1.
3. $+, *$, for addition and multiplication, interpreted to behave as follows:
   $x +^{\mathscr{FD}} y \to x +^{\mathbb{Z}} y$ for all $x, y \in \mathbb{Z}$; $t +^{\mathscr{FD}} s \to \bot$ whenever $t \notin \mathbb{Z}$ or $s \notin \mathbb{Z}$; and analogously for $*^{\mathscr{FD}}$.
4. $<, \leq, >, \geq$, for numeric comparisons, interpreted to behave as follows:
   $x <^{\mathscr{FD}} y \to true$ for all $x, y \in \mathbb{Z}$ with $x <^{\mathbb{Z}} y$; $x <^{\mathscr{FD}} y \to false$ for all $x, y \in \mathbb{Z}$ with $x \geq^{\mathbb{Z}} y$; $t <^{\mathscr{FD}} s \to \bot$ whenever $t \notin \mathbb{Z}$ or $s \notin \mathbb{Z}$; and analogously for $\leq^{\mathscr{FD}}, >^{\mathscr{FD}}, \geq^{\mathscr{FD}}$.
5. *domain*, finite domain primitive interpreted to behave as follows:
   $domain^{\mathscr{FD}}\ x\ [x_1, \ldots, x_n] \to true$ if x and all the $x_i\ (1 \leq i \leq n)$ are integers, $x_i <^{\mathbb{Z}} x_{i+1}$ for all $1 \leq i < n$ and $x =^{\mathbb{Z}} x_i$ for some $1 \leq i \leq n$; $domain^{\mathscr{FD}}\ x\ [x_1, \ldots, x_n] \to false$ if x and all the $x_i\ (1 \leq i \leq n)$ are integers, $x_i <^{\mathbb{Z}} x_{i+1}$ for all $1 \leq i < n$ and $x \neq^{\mathbb{Z}} x_i$ for all $1 \leq i \leq n$; $domain^{\mathscr{FD}}\ t\ s \to \bot$ otherwise.
6. *indomain*, labeling primitive interpreted to behave as follows:
   $indomain^{\mathscr{FD}}\ x \to success$ for all $x \in \mathbb{Z}$; $indomain^{\mathscr{FD}}\ t \to \bot$ whenever $t \notin \mathbb{Z}$.

In practice, the *domain* primitive is used for constraining the possible values of an integer variable to be members of a finite domain, represented as an increasing list of integers. On the other hand, the *indomain* primitive corresponds to the so-called *labeling* predicate [99,77,92]; it is used for constraining an integer variable to take concrete values from its domain, thereby allowing for search. The clever combination of *labeling* with the *propagation* techniques used by finite domain solvers in order to prune variable domains, is crucial for the success of finite domain applications.

2.3 Constraints over a given Constraint Domain

Assuming an arbitrarily fixed constraint domain $\mathscr{D}$ built over a certain set of urelements $\mathscr{U}$, we will now define the syntax and semantics of constraints. As in the *CLP* case, we view constraints as logical formulas. In contrast to *CLP*, our constraints can include occurrences of user defined functions. 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 $p\bar{t}_n \rightarrow ! t$, with $p \in PF^n$, $t_i \in Pat_\perp(\mathscr{U})$ for all $1 \leq i \leq n$, and $t \in Pat(\mathscr{U})$. The special constants $\Diamond$ 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 $p\bar{e}_n \rightarrow ! t$, with $p \in PF^n$, $e_i \in Exp_\perp(\mathscr{U})$ for all $1 \leq i \leq n$, and $t \in Pat(\mathscr{U})$. The special constants $\Diamond$ and $\blacklozenge$ are also atomic constraints.
4. *Constraints* are built from atomic constraints by means of logical conjunction $\wedge$ and existential quantification $\exists$.

In the sequel we use the following notations:

- $PCon_\perp(\mathscr{D})$, the set of all the primitive constraints π over $\mathscr{D}$.
- $PGCon_\perp(\mathscr{D})$, the set of all the primitive ground constraints over $\mathscr{D}$, defined as $\{\pi \in PCon_\perp(\mathscr{D}) \mid fvar(\pi) = \emptyset\}$, where $fvar(\pi)$ is defined as the set of all variables which have some free occurrence in π.
- $PCon(\mathscr{D})$, the set of all the total primitive constraints over $\mathscr{D}$, defined as $\{\pi \in PCon_\perp(\mathscr{D}) \mid \pi$ has no occurrences of $\perp\}$.
- $PGCon(\mathscr{D})$, the set of all the primitive ground and total constraints, $PGCon_\perp(\mathscr{D}) \cap PCon(\mathscr{D})$.

We also write $DCon_\perp(\mathscr{D})$ for the set of all the user defined constraints δ over $\mathscr{D}$, as well as $DGCon_\perp(\mathscr{D})$, $DCon(\mathscr{D})$ and $DGCon(\mathscr{D})$ for the subsets of $DCon_\perp(\mathscr{D})$ consisting of ground, total, and ground and total constraints, respectively. We reserve the capital greek letters Π resp. Δ for sets of primitive resp. user defined constraints, usually interpreted as conjunctions. The notations $fvar(\Pi)$ resp. $fvar(\Delta)$ will refer to the set of free variables occurring in such sets. The semantics of user defined constraints depends on the interpretation of user defined functions, and will be investigated in the next section as part of the semantics of $CFLP(\mathscr{D})$-programs. 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. the set of total valuations over $\mathscr{D}$ is defined as $Val_\perp(\mathscr{D}) = GSub_\perp(\mathscr{U})$ resp. $Val(\mathscr{D}) = GSub(\mathscr{U})$.
2. The set of solutions of $\pi \in PCon_\perp(\mathscr{D})$ is a subset $Sol_\mathscr{D}(\pi) \subseteq Val_\perp(\mathscr{D})$ recursively defined as follows:

(a) $Sol_{\mathscr{D}}(\Diamond) = Val_{\perp}(\mathscr{D})$.
(b) $Sol_{\mathscr{D}}(\blacklozenge) = \emptyset$.
(c) $Sol_{\mathscr{D}}(p\bar{t}_n \to !\, t) = \{\eta \in Val_{\perp}(\mathscr{D}) \mid t\eta \text{ is total and } p^{\mathscr{D}}\bar{t}_n\eta \to t\eta\}$.
(d) $Sol_{\mathscr{D}}(\pi_1 \wedge \pi_2) = Sol_{\mathscr{D}}(\pi_1) \cap Sol_{\mathscr{D}}(\pi_2)$.
(e) $Sol_{\mathscr{D}}(\exists X\pi) = \{\eta \in Val_{\perp}(\mathscr{D}) \mid \eta' \in Sol_{\mathscr{D}}(\pi) \text{ for some } \eta' =_{\setminus\{X\}} \eta\}$

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

According to item 2.(c) in this definition, the solutions of a primitive atomic constraint $p\bar{t}_n \to !\, t$ are those valuations for which $p\bar{t}_n$ can return a total value which matches the total pattern t. For instance, $\eta \in Sol_{\mathscr{R}}(X + Y \to !\, 5)$ holds iff $\eta(X) = x \in \mathbb{R}$, $\eta(Y) = y \in \mathbb{R}$, and $x +^{\mathbb{R}} y = 5$. The other items in the definition are quite standard.

The semantics of atomic constraints $p\bar{t}_n \to !\, t$ just discussed shows that atomic constraints in our framework are related to the behavior of primitive functions, which may be boolean or not. In the case of a boolean primitive p, a constraint of the form $p\bar{t}_n \to !\, R$ (where R is a variable) imposes a relationship between the value of R and the result returned by $p\bar{t}_n$, which may be $true$ or $false$. This could be viewed as a formal analogy between constraints of the form $p\bar{t}_n \to !\, R$ and *reified constraints* of the form $c\# \Leftrightarrow R$, used in several constraint languages for relating the value of a boolean variable R to the entailment or disentailment of a constraint c (see [92] for an explanation of the use of reified constraints in SICStus Prolog). The analogy, however, is misleading; in our setting, the constraint $p\bar{t}_n \to !\, R$ cannot be understood as relating the value of variable R to the behavior of another constraint $p\bar{t}_n$. In fact, $p\bar{t}_n$ standing by itself is not a constraint, but a functional expression. Moreover, the *CFLP* scheme presented in this paper is not intended to capture the operational behavior of reified constraints.

In any case, our atomic constraints $p\bar{t}_n \to !\, t$ based on the behavior of primitive functions are more expressive than purely relational atomic constraints. This has been argued elsewhere [70] for the particular case of strict equality and disequality constraints over constructor terms. In order to clarify this matter, let us consider the example of equality and disequality constraints over the real numbers. According to the traditional (relational) view one would use two different primitive predicates, say $=_{\mathbb{R}}$ and $/=_{\mathbb{R}}$, for writing atomic constraints such as $X =_{\mathbb{R}} Y$ or $X /=_{\mathbb{R}} Y$. In $CFLP(\mathscr{R})$ these atomic constraints can be written as $eq_{\mathbb{R}}\, X\, Y \to !\, true$ and $eq_{\mathbb{R}}\, X\, Y \to !\, false$, respectively. Moreover, one can also write the atomic constraint $eq_{\mathbb{R}}\, X\, Y \to !\, R$, whose use in programs can lead to greater expressivity. An improvement of efficiency can also be expected in computations depending on the value obtained for R by constraint solving, because it will be possible to solve the constraint $eq_{\mathbb{R}}\, X\, Y \to !\, R$ one single time instead of checking which of the two constraints $X =_{\mathbb{R}} Y$ and $X /=_{\mathbb{R}} Y$ succeeds. Similar considerations apply to the various inequality primitives in $\mathscr{R}$ and to the strict equality primitive *seq* in any constraint domain where it is available.

In the sequel we allow some useful shorthands for writing atomic constraints, primitive or not:

- $p\bar{e}_n$ abbreviates $p\bar{e}_n \to !\, success$.
- $e_1 =_{\mathscr{U}} e_2$ abbreviates $eq_{\mathscr{U}}\, e_1\, e_2 \to !\, true$. As particular case, $e_1\# = e_2$ abbreviates $e_1 =_{\mathbb{Z}} e_2$.
- $e_1 /=_{\mathscr{U}} e_2$ abbreviates $eq_{\mathscr{U}}\, e_1\, e_2 \to !\, false$.
- $e_1 == e_2$ abbreviates $seq\, e_1\, e_2 \to !\, true$.
- $e_1 /= e_2$ abbreviates $seq\, e_1\, e_2 \to !\, false$.
- $e_1 < e_2$ abbreviates $e_1 < e_2 \to !\, true$ (and analogously for other comparison primitives).

- $e_1 \geq e_2$ abbreviates $e_1 < e_2 \rightarrow ! \; false$ (and analogously for other comparison primitives).
- $e \in d$ abbreviates $domain\, e\, d \rightarrow ! \; true$ and $e_1, \ldots, e_n \in d$ abbreviates $e_1 \in d \wedge \ldots \wedge e_n \in d$.
- $e \notin d$ abbreviates $domain\, e\, d \rightarrow ! \; false$ and $e_1, \ldots, e_n \notin d$ abbreviates $e_1 \notin d \wedge \ldots \wedge e_n \notin d$.
- $labeling \; l \; [e_1, \ldots, e_n]$ abbreviates $indomain \; e_1 \wedge \ldots \wedge indomain \; e_n$, using as additional parameter a list l of items which allows to specify different labeling strategies. The choice of a particular labeling strategy has an effect on the operational behavior and therefore on efficiency; see [38,92] for details. However, any labeling strategy computes the same solutions set, by constraining integer variables to take concrete values from their domains in all possible ways.

Using the notion of solution given in Definition 3, some useful semantic notions related to primitive constraints are easily introduced:

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

1. π is called satisfiable in $\mathscr{D}$ (in symbols $Sat_\mathscr{D}(\pi)$) iff $Sol_\mathscr{D}(\pi) \neq \emptyset$. Otherwise π is called unsatisfiable (in symbols $Unsat_\mathscr{D}(\pi)$). Analogously for constraint sets Π.
2. π is a consequence of Π in $\mathscr{D}$ (in symbols, $\Pi \models_\mathscr{D} \pi$) iff $Sol_\mathscr{D}(\Pi) \subseteq Sol_\mathscr{D}(\pi)$. π is valid in $\mathscr{D}$ (in symbols, $\models_\mathscr{D} \pi$) iff $\emptyset \models_\mathscr{D} \pi$, which is obviously equivalent to $Sol_\mathscr{D}(\pi) = Val_\perp(\mathscr{D})$.
3. $e \sqsubseteq e'$ is a consequence of Π in $\mathscr{D}$ (in symbols, $\Pi \models_\mathscr{D} e \sqsubseteq e'$) iff $e\eta \sqsubseteq e'\eta$ holds for all $\eta \in Sol_\mathscr{D}(\Pi)$. $e \sqsubseteq e'$ is valid in $\mathscr{D}$ (in symbols, $\models_\mathscr{D} e \sqsubseteq e'$) iff $\emptyset \models_\mathscr{D} e \sqsubseteq e'$, which is obviously equivalent to requiring $e\eta \sqsubseteq e'\eta$ to hold for all $\eta \in Val_\perp(\mathscr{D})$. $\Pi \models_\mathscr{D} e \sqsupseteq e'$ and $\models_\mathscr{D} e \sqsupseteq e'$ are defined analogously.
4. $p\bar{t}_n \rightarrow t$ is a consequence of Π in $\mathscr{D}$ (in symbols, $\Pi \models_\mathscr{D} p\bar{t}_n \rightarrow t$) iff $p^\mathscr{D}\bar{t}_n\eta \rightarrow t\eta$ holds for all $\eta \in Sol_\mathscr{D}(\Pi)$. $p\bar{t}_n \rightarrow t$ is valid in $\mathscr{D}$ (in symbols, $\models_\mathscr{D} p\bar{t}_n \rightarrow t$) iff $\emptyset \models_\mathscr{D} p\bar{t}_n \rightarrow t$ iff $p^\mathscr{D}\bar{t}_n\eta \rightarrow t\eta$ holds for all $\eta \in Val_\perp(\mathscr{D})$. $\Pi \models_\mathscr{D} p\bar{t}_n \rightarrow ! t$ and $\models_\mathscr{D} p\bar{t}_n \rightarrow ! t$ are defined analogously.

Items 3 and 4 in the previous definition will be needed for defining some logical inference rules in sections 3.3 and 4.1. Note that the statement $p\bar{t}_n \rightarrow t$ used in item 4 (not to be confused with a primitive atomic constraint $p\bar{t}_n \rightarrow ! t$, which has a different semantics) is intended to mean that evaluation of the primitive function call $p\bar{t}_n$ is able to return a result t. In sections 3.3 and 4.1 this idea will be generalized to *production statements* of the form $e \rightarrow t$ (with $e \in Exp_\perp(\mathscr{U})$ and $t \in Pat_\perp(\mathscr{U})$), intended to mean that evaluation of the expression e can return the value t.

3 A New $CFLP(\mathscr{D})$ Scheme

The *CLP* scheme, originally introduced by Jaffar and Lassez [54], served the purpose of defining a family of constraint logic programming languages $CLP(\mathscr{D})$ parameterized by a constraint domain $\mathscr{D}$, in such a way that the well established results on the semantics of logic programs could be lifted to all the $CLP(\mathscr{D})$ languages in an elegant and uniform way; see [56] for an updated presentation. Previous work on *CFLP* schemes, including our old scheme $CFLP(\mathscr{D})$ [64,65] had similar aims w.r.t. functional logic programming, differing mainly in the kind of semantic framework provided.

We will now complete the presentation of the new $CFLP(\mathscr{D})$ scheme, assuming that constraint domains are as discussed in the previous section. As in other previous approaches, we introduce programs as sets of constrained rewrite rules for defined function symbols. We provide a semantics for $CFLP(\mathscr{D})$-programs by defining a class of interpretations and a model relationship between interpretations and programs. The main results in the section concern the existence of least models and their characterization as least fixpoints of continuous operators.

3.1 $CFLP(\mathscr{D})$-Programs and Goals

In the sequel we assume an arbitrarily fixed constraint domain $\mathscr{D}$ built over a set of urelements $\mathscr{U}$. As $CFLP(\mathscr{D})$-program we allow any set $\mathscr{P}$ of constrained rewrite rules for defined function symbols, also called *program rules*. More precisely, a program rule R for $f \in DF^n$ has the form

$$R: f\bar{t}_n \to r \Leftarrow P \square \Delta$$

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 $1 \leq i \leq n$, $t_i \in Pat(\mathscr{U})$ are total patterns.
2. The *right-hand side* $r \in Exp(\mathscr{U})$ is a total expression.
3. $\Delta \subseteq DCon(\mathscr{D})$ is a finite set of total atomic constraints, intended to be interpreted as conjunction, and possibly including occurrences of defined function symbols.
4. P is a finite set 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 $1 \leq i \leq k$, $e_i \in Exp(\mathscr{U})$ is a total expression, $s_i \in Pat(\mathscr{U})$ is a total linear pattern, and $var(s_i) \cap var(f\bar{t}_n) = \emptyset$.
   (b) For all $1 \leq i \leq j \leq k$, $var(e_i) \cap var(s_j) = \emptyset$.
   (c) For all $1 \leq i < j \leq k$, $var(s_i) \cap var(s_j) = \emptyset$.

The left-linearity condition required in item 1 is quite common in functional and functional logic programming. As in constraint logic programming, the conditional part of a program rule needs no explicit occurrences of existential quantifiers, because a program rule like R above is logically equivalent to

$$R': f\bar{t}_n \to r \Leftarrow \exists \overline{Y}(P \square \Delta)$$

where $\overline{Y} = var(P \square \Delta) \setminus var(f\bar{t}_n \to r)$. The admissibility conditions 4.(a), (b) and (c) are best understood by thinking of each production $e_i \to s_i$ as a local definition, expected to work by obtaining values for the variables in the pattern s_i by matching the result of evaluating e_i to s_i. Admissibility just means that the locally defined variables must be fresh w.r.t. the left-hand side of the program rule, and also that the local definitions are not recursive. Placing $P \square \Delta$ as conditional part in the program rule means that the productions in P *and also* the constraints in Δ must succeed for the rewrite rule to be applicable.

The following example illustrates the previous points by showing some constrained rewrite rules which could be part of a $CFLP(\mathscr{R})$-program. The main function *split* is intended to receive a list *Xs* of real numbers as parameter and to return a pair (*Ys*, *Zs*) of lists, where the members of *Ys* are the positive members of *Xs* and the members of *Zs* are the other members of *Xs*. We assume that (*Ys*, *Zs*) corresponds to the application of a binary

constructor in mixfix notation, and we also use a Prolog-like syntax for list constructors. The program rules for function *case* shows that an empty conditional part can be omitted for the sake of simplicity.

Example 4 Splitting a list of numbers in $CFLP(\mathscr{R})$:

$$
\begin{array}{lllll}
split & [\,] & \to & ([\,],[\,]) & \\
split & [X|Xs] & \to & case\ R\ X\ Ys\ Zs \Leftarrow & split\ Xs \to (Ys,Zs) \\
 & & & & \square X > 0 \to\!!\ R \\
case & true & X & Ys & Zs \to ([X|Ys],Zs) \\
case & false & X & Ys & Zs \to (Ys,[X|Zs])
\end{array}
$$

Goals for $CFLP(\mathscr{D})$-programs have the same form as the conditional part of program rules. Computed answers for a goal $G : P \square \Delta$ are expected to be pairs of the form $S \square \sigma$, where σ is an idempotent substitution, S is a set of *primitive* constraints verifying $dom(\sigma) \cap var(S) = \emptyset$, and any valuation η which is a solution of S is also a solution of $G\sigma$ (in symbols, $Sol_{\mathscr{D}}(S) \subseteq Sol_{\mathscr{D}}(G\sigma)$).

For instance, in the case of Example 4, the expected computed answers for the goal G : $split\ [1.2, X, -0.25] == (Ys,Zs)$ are

$$
\begin{array}{l}
S_1 \square \sigma_1 = \{X > 0\} \square \{Ys \mapsto [1.2, X], Zs \mapsto [-0.25]\} \\
S_2 \square \sigma_2 = \{X \leq 0\} \square \{Ys \mapsto [1.2], Zs \mapsto [X, -0.25]\}
\end{array}
$$

Note that in this case $Sol_{\mathscr{R}}(S_1) \subseteq Sol_{\mathscr{D}}(G\sigma_1)$ amounts to $Sol_{\mathscr{R}}(X > 0) \subseteq Sol_{\mathscr{D}}(split\ [1.2, X, -0.25] == ([1.2, X], [-0.25])$, which is intuitively true; and analogously for the second computed answer.

In general, correctly computed answers are expected to satisfy the requirement $Sol_{\mathscr{D}}(S) \subseteq Sol_{\mathscr{D}}(G\sigma)$, whose meaning depends on the *declarative semantics* of programs. The rest of this paper provides a formal framework for *declarative semantics*. Another very important issue is *operational semantics*, dealing with the formal specification of *goal solving* methods which take care of computing answers. Goal solving lies outside the scope of this paper, but the brief Subsection 3.2 has been included below in order to provide a few essential pointers to the literature.

In the rest of this subsection we present two further program fragments to illustrate other interesting features of $CFLP(\mathscr{D})$-programming. Example 5 illustrates the use of infinite data structures and higher-order programming facilities in $CFLP(\mathscr{R})$:

Example 5 Higher-order Programming in $CFLP(\mathscr{R})$:

$$
\begin{array}{lllll}
inInterval & A & B & X & \to\ and\ (A \leq X)\ (X \leq B) \\
 & & & & \\
from & X & D & \to & [\,X \mid from\ (X+D)\ D\,] \\
 & & & & \\
takeWhile & P & [\,] & \to & [\,] \\
takeWhile & P & [X|Xs] & \to & if\ (P\,X)\ [\,X \mid takeWhile\ P\ Xs\,]\ [\,] \\
if & true & L & R & \to\ L \\
if & false & L & R & \to\ R \\
and & true & X & & \to\ X \\
and & false & X & & \to\ false
\end{array}
$$

Here, the function $from$ is intended to generate an infinite list of real numbers whose members form an arithmetic sequence, and the higher-order function $takeWhile$ can be used for computing a prefix of a given list, up to the point where some element fails to satisfy a given property. Function $inInterval$ is defined by means of the comparison primitives and can be used to build defined constraints asserting membership between numbers and intervals.

One possible goal for a program including the function definitions of Example 5 is

$$takeWhile\,(inInterval\,0\,1)\,(from\,X\,0.5) == [X]$$

Note the use of the pattern $(inInterval\,0\,1)$ within the goal for expressing a functional value, namely the boolean function that decides membership for the interval of real numbers lying between 0 and 1. The expected computed answer for this goal is

$$S \square \sigma = \{X > 0.5, X \leq 1\} \square \{\,\}$$

which can be written more simply as the set of constraints $\{X > 0.5, X \leq 1\}$. The computation of this answer in a practical $CFLP(\mathscr{R})$ system would require a combination of lazy evaluation and constraint solving.

The next (and last) example illustrates the combination of lazy evaluation and constraint solving in $CFLP(\mathscr{FD})$:

Example 6 Lazy evaluation of infinite lists in $CFLP(\mathscr{FD})$:

$$\begin{array}{llcll}
take & N & Xs & \rightarrow & [\,] \quad \Leftarrow \quad N \leq 0 \\
take & N & [\,] & \rightarrow & [\,] \quad \Leftarrow \quad N > 0 \\
take & N & [X \mid Xs] & \rightarrow & [X \mid take\ (N-1)\ Xs] \quad \Leftarrow \quad N > 0
\end{array}$$

$$\begin{array}{lcl}
generate\ N & \rightarrow & [\,] \Leftarrow N \leq 0 \\
generate\ N & \rightarrow & [X \mid generate\ N] \Leftarrow N > 0,\ X \in (interval\ 0\ (N-1))
\end{array}$$

$$interval\ L\ U \quad \rightarrow \quad if\ \ R\ \ [L \mid interval\ (L+1)\ U]\ \ [\,] \Leftarrow L \leq U \rightarrow\!! R$$

A possible goal for a program including the functions defined in this example could be

$$take\,3\,(generate\,10) == List$$

which asks for the 3 first elements of an infinite list of integers, generated under the constraint that each of its elements must be an integer value belonging to the interval [0..9]. This explains the variable bindings and the constraints occurring in the computed answer

$$S \square \sigma = \{X_1, X_2, X_3 \in [0,1,2,3,4,5,6,7,8,9]\} \square \{List \mapsto [X_1, X_2, X_3]\}$$

whose computation in a practical $CFLP(\mathscr{D})$ system would require a combination of lazy evaluation and constraint solving, as in Example 5.

More practical examples of $CFLP(\mathscr{FD})$-programming can be found in [36]. Appendix A includes more small examples of $CFLP$ programs over the constraint domains $\mathscr{H}_{seq}$, $\mathscr{R}$ and $\mathscr{FD}$, which can be executed in the *TOY* system [1,38] and are written in *TOY*'s concrete syntax.

3.2 Goal Solving

This short Subsection is intended to provide a few essential pointers to the literature on goal solving methods which can be used as a formal basis for the operational semantics of $CFLP(\mathscr{D})$-programs.

As explained in the Introduction, the *CFLP* scheme proposed in this paper is intended to provide a clean declarative semantics for the combination of functional logic programming and constraint logic programming. During the last 20 years, a great variety of goal solving methods for functional logic programming have been investigated. Many of them are based on *narrowing*, a combination of rewriting and unification, originally proposed as a theorem proving tool [93,61,35,50]. The literature on narrowing as a goal solving method for functional logic programming includes results establishing the soundness and completeness of different variants of narrowing, with respect to different formalizations of the intended declarative semantics of programs; see the references [31,46,80] for detailed information.

Results on narrowing methods which can be proved sound and complete with respect to the rewriting logic *CRWL* [42,44,12] are particularly relevant for the $CFLP(\mathscr{D})$ scheme, since the declarative semantics for $CFLP(\mathscr{D})$-programs presented in this paper can be formally characterized by means of the constraint rewriting logic $CRWL(\mathscr{D})$, a natural extension of *CRWL* which is presented in Section 4 below. Many narrowing methods are conveniently presented as formal calculi consisting of transformation rules for transforming an initial goal into different goals in solved form that represent different computed answers. This is the case for the Constructor-based Lazy Narrowing Calculus *CLNC* [42] and its higher-order extension *HOLNC* [43], which are both sound and complete w.r.t. *CRWL* semantics. The interested reader is referred to [87] for a tutorial presentation of these calculi and a comparison to other related narrowing methods.

In addition to being logically sound and complete, a narrowing method must satisfy other additional properties in order to be a good guideline for the efficient implementation of goal solving systems. Narrowing methods with the property of performing only needed steps are particularly interesting in this regard. Intuitively, needed narrowing steps are those narrowing steps which are demanded for the successful completion of the overall computation. This idea was investigated in a pragmatic, implementation-oriented way in [63] and formalized in [2,3], where the so-called *needed narrowing* strategy was proved to enjoy a number of optimality properties. Needed narrowing, as presented in [2,3] does not conform to the declarative semantics given by *CRWL*, but the Demand-driven Narrowing Calculus *DNC* presented more recently in [97] maintains the optimality properties of needed narrowing while being sound and complete w.r.t. *CRWL* semantics.

Still more recently, *CLNC* and *DCN* have been taken as a starting point for designing constraint narrowing calculi that are sound and complete w.r.t. $CRWL(\mathscr{D})$ semantics, the declarative semantics for $CFLP(\mathscr{D})$ programs which is one main contribution of this paper. Firstly, *CLNC* was extended to the Constraint Lazy Narrowing Calculus $CLNC(\mathscr{D})$ [67] which does not incorporate a needed narrowing strategy; and later *DNC* was extended to the Constraint Demand-driven Narrowing Calculus $CDNC(\mathscr{D})$ [98] which enjoys the optimality properties of needed narrowing. Both [67] and [98] use the $CRWL(\mathscr{D})$ logic from Section 4 below for defining solution sets $Sol_{\mathscr{D}}(G\sigma)$ of instantiated goals, which are needed for the formulation of soundness and completeness properties. Both $CLNC(\mathscr{D})$ and $CDNC(\mathscr{D})$ combine lazy resp. needed narrowing with constraint solving, postulating a solver for the constraint domain $\mathscr{D}$ which must satisfy some assumptions needed for proving soundness and completeness of the calculi. The proper assumptions for solvers have been obtained by generalizing ideas from previous works [64,7,70]. We believe that the calculus $CDNC(\mathscr{D})$

is quite adequate for formalizing the behavior of actual $CFLP(\mathscr{D})$ programming systems. Some support for this claim in the particular case of finite domain constraints can be found in [32], where $CDNC(\mathscr{FD})$ computations have been shown to provide a faithful modelling of actual computations in the $TOY(\mathscr{FD})$ system [37,38].

3.3 Interpretations and Models for $CFLP(\mathscr{D})$-Programs

In order to interpret $CFLP(\mathscr{D})$-programs, the constraint domain $\mathscr{D}$ has to be extended with interpretations for the defined function symbols. The *$\mathscr{D}$-algebras* defined below achieve this aim in a simple and straightforward way:

Definition 5 $\mathscr{D}$-algebras.
Assume a constraint domain $\mathscr{D}$ with set of urelements $\mathscr{U}$. A $\mathscr{D}$-algebra is any structure of the form

$$\mathscr{A} = \langle \mathscr{D}, \{f^{\mathscr{A}} \mid f \in DF\} \rangle$$

conservatively extending $\mathscr{D}$ with an interpretation $f^{\mathscr{A}}$ of each $f \in DF^n$, which must satisfy the following requirements:

1. $f^{\mathscr{A}} \subseteq D_{\mathscr{U}}^n \times D_{\mathscr{U}}$, which boils down to $f^{\mathscr{A}} \subseteq D_{\mathscr{U}}$ in the case $n = 0$. The notation $f^{\mathscr{A}}\, \bar{t}_n \rightarrow t$ indicates that $(\bar{t}_n, t) \in f^{\mathscr{A}}$. In the case $n = 0$, this notation boils down to $f^{\mathscr{A}} \rightarrow t$.
2. $f^{\mathscr{A}}$ behaves monotonically in its arguments and antimonotonically in its result; i.e., whenever $f^{\mathscr{A}}\, \bar{t}_n \rightarrow t$, $\bar{t}_n \sqsubseteq \overline{t'}_n$ and $t \sqsupseteq t'$ one also has $f^{\mathscr{A}}\, \overline{t'}_n \rightarrow t'$.

Similarly to Definition 1, the monotonicity conditions in item 2 are intended to capture the behavior of a possibly non-deterministic function over finite data elements. The radicality condition in Definition 1 is omitted here, because user defined functions which return potentially infinite data structures as results are useful for programming and obviously not radical.

A full-fledged semantics for $CFLP(\mathscr{D})$-programs could be developed on the basis of $\mathscr{D}$-algebras. This approach would be analogous to the $\mathscr{D}$-interpretations used in the traditional semantics of $CLP(\mathscr{D})$-programs [56], and also formally similar to the structures used as interpretations for functional logic programs in our previous $CFLP(\mathscr{D})$ scheme [64,65] and previous work based on the logic $CRWL$ [42,44,12].

We have nevertheless decided to abandon $\mathscr{D}$-algebras in favor of a more expressive approach, motivated by the π-interpretations for $CLP(\mathscr{D})$-programs proposed in [39,40]. Roughly speaking, π-interpretations in the CLP setting are sets of facts of the form $p\bar{t}_n \Leftarrow \Pi$, intended to mean that the user defined atom $p\bar{t}_n$ is valid for any valuation which is a solution of the primitive constraints Π. As shown in [39,40], π-interpretations can be used as a basis for three different program semantics S_i ($i = 1,2,3$), characterizing *valid ground goals*, *valid answers for goals* and *computed answers for goals*, respectively. In fact, the S_i semantics are the CLP counterpart of previously known semantics for logic programming, namely the least ground Herbrand model semantics [5,62], the open Herbrand model semantics, also known as C-semantics [21,34] and the S-semantics [33,16]. A very concise and readable overview of these semantics can be found in [6].

In order to generalize π-interpretations to $CFLP(\mathscr{D})$ languages, we consider sets of facts of the form $f\bar{t}_n \rightarrow t \Leftarrow \Pi$, intended to describe the behavior of user defined functions $f \in DF^n$. We will use this class of interpretations for defining two different semantics, corresponding to S_1 and S_2, which we will call the *weak* and *strong* semantics, respectively.

The strong semantics is intended to provide a characterization of valid answers for goals, including computed answers as a particular case.

As shown by the rest of this paper and some recent results cited in the last paragraph of Subsection 3.2, weak and strong semantics have a natural declarative characterization and are a useful tool for proving soundness and completeness of various operational semantics. Note that a $CFLP(\mathscr{D})$ analogous of the S_3 semantics would aim at characterizing *exactly* the computed answers. In the pure *CLP* setting, there is wide agreement on a unique formalization of operational semantics which gives rise to a well-defined class of computed answers [56]. On the other hand, as explained in Subsection 3.2, a variety of *narrowing strategies* have been proposed as a basis for operational semantics in the *CFLP* setting, giving rise to different classes of computed answers. For this reason, we have made no attempt of developing a S_3-like semantics for $CFLP(\mathscr{D})$ programs.

Our next step is to define an analog to π-interpretations for $CFLP(\mathscr{D})$-programs. To this purpose, we need some preliminary notions.

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

1. We consider three possible kinds of constrained statements (c-statements):
   (a) c-productions $e \to t \Leftarrow \Pi$, with $e \in Exp_\perp(\mathscr{U})$. In the case that Π is empty they boil down to unconstrained productions written as $e \to t$. A c-production is called trivial iff $t = \perp$ or $Unsat_{\mathscr{D}}(\Pi)$.
   (b) c-facts $f\bar{t}_n \to t \Leftarrow \Pi$, with $f \in DF^n$. They are just a particular kind of c-productions. In the case that Π is empty they boil down to unconstrained facts written as $f\bar{t}_n \to t$. A c-fact is called trivial iff $t = \perp$ or $Unsat_{\mathscr{D}}(\Pi)$.
   (c) c-atoms $p\bar{e}_n \to !\, t \Leftarrow \Pi$, with $p \in PF^n$ and t total. In the case that Π is empty they boil down to unconstrained atoms written as $p\bar{e}_n \to !\, t$. A c-atom is called trivial iff $Unsat_{\mathscr{D}}(\Pi)$.

   In the sequel we use φ and similar symbols to denote any c-statement of the form $e \to^? t \Leftarrow \Pi$, where the symbol $\to^?$ must be understood as $\to !$ in case that φ is a c-atom; otherwise $\to^?$ must be understood as $\to$.

2. Given two c-statements φ and φ', we say that φ $\mathscr{D}$-entails φ' (in symbols, $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$) 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 $\sigma \in Sub_\perp(\mathscr{U})$ such that $\Pi' \models_{\mathscr{D}} \Pi\sigma, \Pi' \models_{\mathscr{D}} e' \sqsupseteq e\sigma, \Pi' \models_{\mathscr{D}} t' \sqsubseteq t\sigma$.
   (b) $\varphi = p\bar{e}_n \to !\, t \Leftarrow \Pi$, $\varphi' = p\overline{e'}_n \to !\, t' \Leftarrow \Pi'$, and there is some $\sigma \in Sub_\perp(\mathscr{U})$ such that $\Pi' \models_{\mathscr{D}} \Pi\sigma, \Pi' \models_{\mathscr{D}} p\overline{e'}_n \sqsupseteq (p\bar{e}_n)\sigma, \Pi' \models_{\mathscr{D}} t' \sqsupseteq t\sigma$.

The intuitive idea behind $\mathscr{D}$-entailment is that, whenever $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$, the c-statement φ' can be accepted as a consequence of φ for any possible interpretation of the defined function symbols. This is indeed reasonable because the definition of the $\mathscr{D}$-entailment relation does rely only on assumptions concerning the monotonic behavior of both primitive and defined functions, as well as on the radical behavior of primitive functions.

The next definition generalizes the idea of π-interpretation [39,40] to our $CFLP(\mathscr{D})$ setting:

Definition 7 For any given constraint domain $\mathscr{D}$:

1. A c-interpretation over $\mathscr{D}$ is any set I of c-facts including all the trivial c-facts and closed under $\mathscr{D}$-entailment. Equivalently, a c-interpretation is any set I of c-facts such that $cl_{\mathscr{D}}(I) \subseteq I$, where $cl_{\mathscr{D}}(I)$ is defined as follows:
$$cl_{\mathscr{D}}(I) = \{\varphi' \mid \varphi' \text{ is a trivial c-fact, or else } \exists \varphi \in I(\varphi \succcurlyeq_{\mathscr{D}} \varphi')\}$$
2. The *ground kernel* of a c-interpretation I is defined as
$$gk_{\mathscr{D}}(I) = \{\varphi \in I \mid \varphi \text{ is a ground c-fact}\}$$
3. The $\mathscr{D}$-*grounding* of a c-interpretation I is defined as the closure of its ground kernel w.r.t. trivial c-facts and $\mathscr{D}$-entailment, i.e.:
$$[I]_{\mathscr{D}} = cl_{\mathscr{D}}(gk_{\mathscr{D}}(I))$$

Note that the ground kernel $gk_{\mathscr{D}}(I)$ is technically not a c-interpretation, although it represents the ground information given by I; while the grounding $[I]_{\mathscr{D}}$ is the least c-interpretation which includes the ground information given by I. Clearly, two different c-interpretations can have the same kernel, and thus the same grounding.

The next definition assumes a constraint domain $\mathscr{D}$ with urelements $\mathscr{U}$, and a given c-interpretation I over $\mathscr{D}$. The purpose of the calculus is to infer the semantic validity of arbitrary c-statements in I.

Definition 8 Semantic Calculus.
We write $I \vdash_{\mathscr{D}} \varphi$ to indicate that the c-statement φ can be derived from I using the following inference rules:

TI Trivial Inference:

$$\frac{}{\varphi}$$

If φ is a trivial c-statement.

RR Restricted Reflexivity:

$$\frac{}{t \to t \Leftarrow \Pi}$$

If $t \in \mathscr{U} \cup \mathscr{V}$.

SP Simple Production:

$$\frac{}{s \to t \Leftarrow \Pi}$$

If $s \in Pat_{\perp}(\mathscr{U})$, $s \in \mathscr{V}$ or $t \in \mathscr{V}$, and $\Pi \models_{\mathscr{D}} s \sqsupseteq t$.

DC Decomposition:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_m \to t_m \Leftarrow \Pi}{h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi}$$

If $h\bar{e}_m$ is passive.

IR Inner Reduction:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_m \to t_m \Leftarrow \Pi}{h\bar{e}_m \to X \Leftarrow \Pi}$$

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

DF$_I$ I-Defined Function:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{f\bar{e}_n \to t \Leftarrow \Pi}$$

If $f \in DF^n$, $(f\bar{t}_n \to t \Leftarrow \Pi) \in I$.

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi, s\bar{a}_k \to t \Leftarrow \Pi}{f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi}$$

If $f \in DF^n$, $k > 0$, $(f\bar{t}_n \to s \Leftarrow \Pi) \in I$, $s \in Pat_\perp(\mathscr{U})$.

PF Primitive Function:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{p\bar{e}_n \to t \Leftarrow \Pi}$$

If $p \in PF^n$, $t_i \in Pat_\perp(\mathscr{U})$ for each $1 \leq i \leq n$, and $\Pi \models_{\mathscr{D}} p\bar{t}_n \to t$.

AC Atomic Constraint:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{p\bar{e}_n \to\!! \, t \Leftarrow \Pi}$$

If $p \in PF^n$, $t_i \in Pat_\perp(\mathscr{U})$ for each $1 \leq i \leq n$, and $\Pi \models_{\mathscr{D}} p\bar{t}_n \to\!! \, t$.

By convention, we agree that no inference rule of the semantic 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.

Any derivation in the semantic 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. For each label $\mathbf{RL} \in \{\mathbf{TI},\ \mathbf{RR},\ \cdots,\ \mathbf{PF},\ \mathbf{AC}\ \}$, we write $T = \mathbf{RL}(\varphi, [T_1,\ \cdots, T_p])$ for representing a proof tree whose root c-statement φ is inferred with the inference rule labelled **RL**, taking as premises p previously derived c-statements with proof trees T_i $(1 \leq i \leq p)$.
2. $T : I \vdash_{\mathscr{D}} \varphi$ indicates that $I \vdash_{\mathscr{D}} \varphi$ is witnessed by the proof tree T.
3. T is called an easy proof tree iff T makes no use of the inference rules **DF$_I$**, **PF** and **AC**.
4. $\|T\|$ denotes the full size of the proof tree T, defined as the total number of nodes in T.

5. $\mid T \mid$ 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 $\mathbf{DF}_I$, **PF** or **AC**. Obviously, $\mid T \mid \leq \|T\|$ and $\mid T \mid = 0$ iff T is an easy proof tree.

The next lemma states several useful properties of the semantic calculus. The proof is rather technical and can be found in Appendix B.1.

Lemma 1 *Properties of the Semantic Calculus.*

1. *Compactness Property:* $\mathrm{I} \vdash_{\mathscr{D}} \varphi$ *implies* $cl_{\mathscr{D}}(\mathrm{I}_0) \vdash_{\mathscr{D}} \varphi$ *for some finite subset* $\mathrm{I}_0 \subseteq \mathrm{I}$.
2. *Extension Property:* $\mathrm{I} \vdash_{\mathscr{D}} \varphi$ *and* $\mathrm{I} \subseteq \mathrm{I}'$ *implies* $\mathrm{I}' \vdash_{\mathscr{D}} \varphi$.
3. *Approximation Property: For any* $e \in Exp_{\perp}(\mathscr{U})$, $t \in Pat_{\perp}(\mathscr{U})$: $\Pi \models_{\mathscr{D}} e \sqsupseteq t$ *iff there is some easy proof tree* T *such that* $T : \vdash_{\mathscr{D}} e \rightarrow t \Leftarrow \Pi$ *(derivation from the trivial c-interpretation* $\perp = cl_{\mathscr{D}}(\emptyset)$*).*
4. *Primitive c-atoms: For any primitive atom* $p\bar{t}_n \rightarrow !t$, $\mathrm{I} \vdash_{\mathscr{D}} p\bar{t}_n \rightarrow !t \Leftarrow \Pi$ *iff* $\Pi \models_{\mathscr{D}} p\bar{t}_n \rightarrow !t$.
5. *Entailment Property:* $T : \mathrm{I} \vdash_{\mathscr{D}} \varphi$ *and* $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$ *implies* $T' : \mathrm{I} \vdash_{\mathscr{D}} \varphi'$ *with proof tree* T' *such that* $\mid T' \mid \leq \mid T \mid$.
6. *Conservation Property: For any c-fact* φ, $\mathrm{I} \vdash_{\mathscr{D}} \varphi$ *iff* $\varphi \in \mathrm{I}$.
7. *Grounding Property: For any ground c-statement* φ, $\mathrm{I} \vdash_{\mathscr{D}} \varphi$ *iff* $[\mathrm{I}]_{\mathscr{D}} \vdash_{\mathscr{D}} \varphi$.

Using the semantic calculus, solutions of user defined constraints can be easily defined. The next definition generalizes Definition 3, assuming a given c-interpretation I over a constraint domain $\mathscr{D}$ with urelements $\mathscr{U}$:

Definition 9 Solutions of User Defined Constraints.

1. The set of solutions of $\delta \in DCon_{\perp}(\mathscr{D})$ is a subset $Sol_I(\delta) \subseteq Val_{\perp}(\mathscr{D})$ recursively defined as follows:
   (a) $Sol_I(\Diamond) = Val_{\perp}(\mathscr{D})$.
   (b) $Sol_I(\blacklozenge) = \emptyset$.
   (c) $Sol_I(\delta) = \{\eta \in Val_{\perp}(\mathscr{D}) \mid I \vdash_{\mathscr{D}} \delta\eta\}$, for any atomic constraint $\delta \in DCon_{\perp}(\mathscr{D}) \setminus \{\Diamond, \blacklozenge\}$.
   (d) $Sol_I(\delta_1 \wedge \delta_2) = Sol_I(\delta_1) \cap Sol_I(\delta_2)$.
   (e) $Sol_I(\exists X \delta) = \{\eta \in Val_{\perp}(\mathscr{D}) \mid \eta' \in Sol_I(\delta) \text{ for some } \eta' =_{\setminus\{X\}} \eta\}$
2. The set of solutions of a set of constraints $\Delta \subseteq DCon_{\perp}(\mathscr{D})$ is defined as $Sol_I(\Delta) = \bigcap_{\delta \in \Delta} Sol_I(\delta)$, corresponding to a logical reading of Δ as the conjunction of its members. In particular, $Sol_I(\emptyset) = Val_{\perp}(\mathscr{D})$, corresponding to the logical reading of an empty conjunction as the identically true constraint $\Diamond$.

For primitive constraints one can easily check that $Sol_I(\pi) = Sol_{\mathscr{D}}(\pi)$ and $Sol_I(\Pi) = Sol_{\mathscr{D}}(\Pi)$, using the obvious correspondence between Definitions 9 and 3.

The semantic calculus also allows to define the denotation of arbitrary expressions in a given interpretation, as follows:

Definition 10 Denotation of Expressions.
Assume a given c-interpretation I over a constraint domain $\mathscr{D}$. The denotation of any expression $e \in Exp_{\perp}(\mathscr{U})$ in I under a valuation $\eta \in Val_{\perp}(\mathscr{D})$ is defined as the set $[\![e]\!]^I_\eta = \{t \in D_{\mathscr{U}} \mid I \vdash_{\mathscr{D}} e\eta \rightarrow t\}$. For the case of a ground expression $e \in GExp_{\perp}(\mathscr{U})$ we will abbreviate $[\![e]\!]^I_\varepsilon$ as $[\![e]\!]^I$.

Using Lemma 1, it is easy to prove that $[\![e]\!]^I_\eta \subseteq D_{\mathscr{U}}$ includes the undefined element $\bot$ and is downwards closed w.r.t. the information ordering $\sqsubseteq$; i.e., $t' \in [\![e]\!]^I_\eta$ holds whenever $t \in [\![e]\!]^I_\eta$ for some $t \sqsupseteq t'$. Due to these properties, $[\![e]\!]^I_\eta$ turns out to be an element of Hoare's Powerdomain $\mathscr{HP}(\overline{D_{\mathscr{U}}})$ [91,104,45], corresponding to so-called *call-time choice* semantics for non-determinism. This kind of semantics is inspired by Hussmann's work on nondeterministic algebraic specifications and programs [51–53] and shown to be convenient for programming in previous work on *CRWL*; see [42,87].

Definitions 9 and 10 just rely on the ground facts provided by the ground kernel of c-interpretations. On the contrary, the first item in the next definition really exploits the non-ground information provided by c-interpretations.

Definition 11 Strong and Weak Models.
For any given $CFLP(\mathscr{D})$-program $\mathscr{P}$ and c-interpretation I we say

1. I is a strong model of $\mathscr{P}$ (in symbols $I \models^s_{\mathscr{D}} \mathscr{P}$) iff
for any $(f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}$, $\theta \in Sub_\bot(\mathscr{U})$, $\Pi \subseteq PCon_\bot(\mathscr{D})$ and $t \in Pat_\bot(\mathscr{U})$ such that $I \vdash_{\mathscr{D}} (P \square \Delta)\theta \Leftarrow \Pi$ and $I \vdash_{\mathscr{D}} r\theta \to t \Leftarrow \Pi$ one has $((f\bar{t}_n)\theta \to t \Leftarrow \Pi) \in I$.
2. I is a weak model of $\mathscr{P}$ (in symbols $I \models^w_{\mathscr{D}} \mathscr{P}$) iff
for any $(f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}$, $\eta \in GSub_\bot(\mathscr{U})$ and $t \in GPat_\bot(\mathscr{U})$ such that $(f\bar{t}_n \to r \Leftarrow P \square \Delta)\eta$ is ground, $I \vdash_{\mathscr{D}} (P \square \Delta)\eta$ and $I \vdash_{\mathscr{D}} r\eta \to t$ one has $((f\bar{t}_n)\eta \to t) \in I$.

Roughly speaking, the weak model semantics $I \models^w_{\mathscr{D}} \mathscr{P}$ means that all the individual instances of program rules from $\mathscr{P}$ must be valid in I. Therefore, a technical variant of weak semantics could be also defined using the $\mathscr{D}$-algebras from Definition 5. On the other hand, the strong model semantics would not make sense for $\mathscr{D}$-algebras. The rough meaning of the strong model relationship $I \models^s_{\mathscr{D}} \mathscr{P}$ is that all those c-facts that are "immediate consequences" from c-facts belonging to I via program rules from $\mathscr{P}$ must belong to I. In comparison with previous works, the weak model semantics is roughly similar to the model notion used for *CRWL* in [42,44,12], to the semantics in our older $CFLP(\mathscr{D})$ scheme [64, 65], and to the more traditional $CLP(\mathscr{D})$ semantics in [54–56]; while the strong model semantics is analogous to the S_2-semantics for $CLP(\mathscr{D})$-programs proposed in [39,40].

The next proposition establishes a natural relationship between strong and weak models:

Proposition 1 *Strong versus Weak Models.*
For any $CFLP(\mathscr{D})$-program $\mathscr{P}$ and any c-interpretation I *one has:* $\mathrm{I} \models^s_{\mathscr{D}} \mathscr{P} \Rightarrow \mathrm{I} \models^w_{\mathscr{D}} \mathscr{P}$.
The reciprocal is false in general.

Proof Any strong model of a given $CFLP(\mathscr{D})$-program $\mathscr{P}$ is also a weak model of $\mathscr{P}$, because item 2 in Definition 11 is a particular case of item 1 obtained when θ is a ground substitution, Π is empty and t is a ground pattern. As a counterexample for the reciprocal, consider the $CFLP(\mathscr{R})$-program $\mathscr{P}$ consisting of one single program rule $notZero\,X \to true \Leftarrow X /=_{\mathbb{R}} 0$ and the following c-interpretation over $\mathscr{R}$:

$$I =_{def} cl_{\mathscr{R}}(\{notZero\,X \to true \Leftarrow X > 0,\\ notZero\,X \to true \Leftarrow X < 0\})$$

For this particular program and c-interpretation the following holds:

1. $I \models^w_{\mathscr{R}} \mathscr{P}$, because item 2 in Definition 11 holds. Indeed, for any $\eta \in GSub_\bot(\mathbb{R})$ which gives a ground instance of the program rule, $I \vdash_{\mathscr{R}} (X /=_{\mathbb{R}} 0)\eta$ iff $\eta(X) \in \mathbb{R} \setminus \{0\}$. Therefore, for such η one also has $((notZero\,X)\eta \to true) \in I$, since I is closed under $\mathscr{R}$-entailment.

2. $I \not\models^s_{\mathscr{R}} \mathscr{P}$, because item 1 in Definition 11 fails when choosing ε as θ and $X /=_{\mathbb{R}} 0$ as Π. Indeed, $I \vdash_{\mathscr{R}} X /=_{\mathbb{R}} 0 \Leftarrow X /=_{\mathbb{R}} 0$, $I \vdash_{\mathscr{R}} true \rightarrow true \Leftarrow X /=_{\mathbb{R}} 0$, and $(notZero X \rightarrow true \Leftarrow X /=_{\mathbb{R}} 0) \notin I$, since this c-fact does not follow by $\mathscr{R}$-entailment from the c-facts used to define I.

The two kinds of models naturally give rise to different notions of logical consequence:

Definition 12 Strong and Weak Consequence.
For any given $CFLP(\mathscr{D})$-program $\mathscr{P}$ and c-statement φ we say

1. φ is a strong consequence of $\mathscr{P}$ (in symbols $\mathscr{P} \models^s_{\mathscr{D}} \varphi$) iff $I \vdash_{\mathscr{D}} \varphi$ holds for every strong model $I \models^s_{\mathscr{D}} \mathscr{P}$.
2. φ is a weak consequence of $\mathscr{P}$ (in symbols $\mathscr{P} \models^w_{\mathscr{D}} \varphi$) iff $I \vdash_{\mathscr{D}} \varphi\eta$ holds for every weak model $I \models^w_{\mathscr{D}} \mathscr{P}$ and every valuation $\eta \in Val_{\perp}(\mathscr{D})$ such that $\varphi\eta$ is ground.

As we will prove in Section 4.2, strong consequence always implies weak consequence; but the reciprocal is false in general.

3.4 A Fixpoint Characterization of Least Models

In this subsection we prove the existence of least models for $CFLP(\mathscr{D})$-programs and we characterize them as least fixpoints, exploiting the lattice structure of the family of all c-interpretations. Similar results are well-known in logic programming [5,62] and constraint logic programming [56,39,40], as well as in our older $CFLP(\mathscr{D})$ scheme [64,65]. In our current $CFLP(\mathscr{D})$ scheme, the lattice structure is revealed by the following result:

Proposition 2 *Interpretation Lattice.*
$\mathbb{I}_{\mathscr{D}}$, defined as the set of all possible c-interpretations I *over the constraint domain $\mathscr{D}$, is a complete lattice w.r.t. the set inclusion ordering. Moreover, the bottom element $\bot$ and the top element $\top$ of this lattice can be characterized as follows:*

$$\begin{array}{lcl} \bot & = & cl_{\mathscr{D}}(\{\varphi \mid \varphi \text{ is a trivial c-fact}\}) \\ \top & = & \{\varphi \mid \varphi \text{ is any c-fact}\} \end{array}$$

Proof $\top$ is trivially the top element of $\mathbb{I}_{\mathscr{D}}$ w.r.t. to the set inclusion ordering. Moreover, $\bot$ is the bottom element because any c-interpretation is required to include all the trivial c-facts and to be closed under $cl_{\mathscr{D}}$. It only remains to show that any subset $\mathfrak{I} \subseteq \mathbb{I}_{\mathscr{D}}$ has a least upper bound $\sqcup\mathfrak{I}$ and a greatest lower bound $\sqcap\mathfrak{I}$ w.r.t. the set inclusion ordering. Let us see why this is true:

- $\sqcup\mathfrak{I} = cl_{\mathscr{D}}(\bigcup \mathfrak{I})$, which is obviously the smallest set of c-facts closed under $cl_{\mathscr{D}}$ and including all $I \in \mathfrak{I}$ as subsets. Note that $\sqcup\emptyset = \bot$ and $\sqcup\mathfrak{I} = \bigcup \mathfrak{I}$ (which is already closed under $cl_{\mathscr{D}}$) for non-empty $\mathfrak{I}$.
- $\sqcap\mathfrak{I} = \bigcap \mathfrak{I}$ (understood as $\top$ if $\mathfrak{I}$ is empty), which is closed under $cl_{\mathscr{D}}$ and the greatest set of c-facts included as a subset in all $I \in \mathfrak{I}$.

The strong and weak interpretation transformers defined below are intended to formalize the computation of strong resp. weak "immediate consequences" from the c-facts belonging to a given c-interpretation.

Definition 13 Interpretation Transformers.
For any given $CFLP(\mathscr{D})$-program $\mathscr{P}$ and c-interpretation I we define:

1. $ST_{\mathscr{P}}(I) =_{def} cl_{\mathscr{D}}(preST_{\mathscr{P}}(I))$, where: $preST_{\mathscr{P}}(I) =_{def} \{(f\bar{t}_n)\theta \to t \Leftarrow \Pi \mid (f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}, \theta \in Sub_{\perp}(\mathscr{U}), \Pi \subseteq PCon_{\perp}(\mathscr{D}), t \in Pat_{\perp}(\mathscr{U}), I \vdash_{\mathscr{D}} (P \square \Delta)\theta \Leftarrow \Pi, I \vdash_{\mathscr{D}} r\theta \to t \Leftarrow \Pi\}$
2. $WT_{\mathscr{P}}(I) =_{def} cl_{\mathscr{D}}(preWT_{\mathscr{P}}(I))$, where: $preWT_{\mathscr{P}}(I) =_{def} \{(f\bar{t}_n)\eta \to t \mid (f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}, \eta \in GSub_{\perp}(\mathscr{U}),$ with $(f\bar{t}_n \to r \Leftarrow P \square \Delta)\eta$ ground, $t \in GPat_{\perp}(\mathscr{U}),\ I \vdash_{\mathscr{D}} (P \square \Delta)\eta, I \vdash_{\mathscr{D}} r\eta \to t\}$

The crucial properties of the interpretation transformers are given in the next proposition, whose proof can be found in Appendix B.1:

Proposition 3 *Properties of the Interpretation Transformers.*
For any fixed CFLP($\mathscr{D}$)-program $\mathscr{P}$, the transformers $ST_{\mathscr{P}}, WT_{\mathscr{P}} : \mathbb{I}_{\mathscr{D}} \to \mathbb{I}_{\mathscr{D}}$ are well defined continuous mappings, whose pre-fixpoints are the strong resp. weak models of $\mathscr{P}$. More precisely, for any $\mathrm{I} \in \mathbb{I}_{\mathscr{D}}$ *one has* $ST_{\mathscr{P}}(\mathrm{I}) \subseteq \mathrm{I}$ *iff* $\mathrm{I} \models^{s}_{\mathscr{D}} \mathscr{P}$, *and* $WT_{\mathscr{P}}(\mathrm{I}) \subseteq \mathrm{I}$ *iff* $\mathrm{I} \models^{w}_{\mathscr{D}} \mathscr{P}$.

Using the previous proposition, the desired characterization of least models is easy to obtain:

Theorem 1 *Least Program Models.*
For every CFLP($\mathscr{D}$)-program $\mathscr{P}$ there exist:

1. *A least strong model* $\mathrm{S}_{\mathscr{P}} = lfp(ST_{\mathscr{P}}) = \bigcup_{k\in\mathbb{N}} ST_{\mathscr{P}} \uparrow^k (\perp)$.
2. *A least weak model* $\mathrm{W}_{\mathscr{P}} = lfp(WT_{\mathscr{P}}) = \bigcup_{k\in\mathbb{N}} WT_{\mathscr{P}} \uparrow^k (\perp)$.

Proof Due to a well known theorem by Knaster and Tarski [95], a monotonic mapping from a complete lattice into itself always has a least fixpoint which is also its least pre-fixpoint. In the case that the mapping is continuous, its least fixpoint can be characterized as the lub of the sequence of lattice elements obtained by reiterated application of the mapping to the bottom element. Combining these results with Proposition 3 trivially proves the theorem.

In the rest of this subsection we investigate the relationship between the two least models $S_{\mathscr{P}}$ and $W_{\mathscr{P}}$ of a given program $\mathscr{P}$. Obviously, $W_{\mathscr{P}} \subseteq S_{\mathscr{P}}$, because $S_{\mathscr{P}}$ is a weak model of $\mathscr{P}$ due to Proposition 1, and therefore $S_{\mathscr{P}}$ must include the least weak model. A sharper characterization of $W_{\mathscr{P}}$ as a subset of $S_{\mathscr{P}}$ follows from the following result, whose proof is given in Appendix B.1:

Proposition 4 *Relationship between both Interpretation Transformers.*
For any given CFLP($\mathscr{D}$)-program $\mathscr{P}$ and any c-interpretation I *over $\mathscr{D}$:*
$WT_{\mathscr{P}}([\mathrm{I}]_{\mathscr{D}}) = [ST_{\mathscr{P}}(\mathrm{I})]_{\mathscr{D}}$.

Using this proposition, we can prove:

Theorem 2 *Relationship between both Least Models.*
For every CFLP($\mathscr{D}$)-program $\mathscr{P}$, the least weak model is the grounding of the least strong model, i.e. $\mathrm{W}_{\mathscr{P}} = [\mathrm{S}_{\mathscr{P}}]_{\mathscr{D}}$.

Proof Due to Theorem 1, it is sufficient to prove that $WT_{\mathscr{P}} \uparrow^k (\perp) = [ST_{\mathscr{P}} \uparrow^k (\perp)]_{\mathscr{D}}$ holds for all $k \in \mathbb{N}$. We reason by induction on k.

1. *Base case*: the equation holds for $k = 0$, since
$WT_{\mathscr{P}} \uparrow^0 (\perp) = \perp = [\perp]_{\mathscr{D}} = [ST_{\mathscr{P}} \uparrow^0 (\perp)]_{\mathscr{D}}$.

2. *Inductive step*: let us assume as *induction hypothesis* that the desired equation holds for k. Then:

$$\begin{array}{rcl} WT_{\mathscr{P}} \uparrow^{k+1} (\bot) & = & WT_{\mathscr{P}}(WT_{\mathscr{P}} \uparrow^{k} (\bot)) \\ & = & WT_{\mathscr{P}}([ST_{\mathscr{P}} \uparrow^{k} (\bot)]_{\mathscr{D}}) \\ & = & [ST_{\mathscr{P}}(ST_{\mathscr{P}} \uparrow^{k} (\bot))]_{\mathscr{D}} \\ & = & [ST_{\mathscr{P}} \uparrow^{k+1} (\bot)]_{\mathscr{D}} \end{array}$$

where the second equation holds by *induction hypothesis*, and the third equation holds by Proposition 4.

In Section 4.2 we will present an example showing that the inclusion $W_{\mathscr{P}} \subseteq S_{\mathscr{P}}$ can be strict for some programs.

4 A Logical Framework for $CFLP(\mathscr{D})$

In this section we generalize the $CRWL$ approach [42,44,12,87] to a new rewriting logic $CRWL(\mathscr{D})$, parameterized by a constraint domain $\mathscr{D}$, and aimed as a logical framework for $CFLP(\mathscr{D})$ programming. We start by presenting a logical calculus for $CRWL(\mathscr{D})$ and investigating its main proof theoretical properties. Next, we investigate the relationship between formal derivability in this calculus and the model theoretic semantics studied in the subsections 3.3 and 3.4. The relevance of $CRWL(\mathscr{D})$ w.r.t. past work and planned future work will be briefly discussed in the concluding section 5.

4.1 The Constraint Rewriting Logic $CRWL(\mathscr{D})$: Proof Theory

The next definition assumes a constraint domain $\mathscr{D}$ with urelements $\mathscr{U}$, and a given $\mathscr{D}$-program $\mathscr{P}$. The purpose of the calculus is to infer the semantic validity of arbitrary c-statements from the program rules in $\mathscr{P}$.

Definition 14 Constrained Rewriting Calculus.
We write $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ to indicate that the c-statement φ can be derived from $\mathscr{P}$ in the constrained rewriting calculus $CRWL(\mathscr{D})$, which consists of the inference rules **TI**, **RR**, **SP**, **DC**, **IR**, **PF** and **AC** already presented in the semantic calculus from Definition 8, plus the following inference rule:

DF$_{\mathscr{D}}$ $\mathscr{P}$-Defined Function:

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi,\ P \square \Delta \ \Leftarrow \Pi, r \to t \Leftarrow \Pi}{f\,\bar{e}_n \to t \Leftarrow \Pi}$$

If $f \in DF^n$, $(f\,\bar{t}_n \to r \Leftarrow P \square \Delta) \in [\mathscr{P}]_{\bot}$.

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi,\ P \square \Delta \ \Leftarrow \Pi, r \to s \Leftarrow \Pi, s\,\bar{a}_k \to t \Leftarrow \Pi}{f\,\bar{e}_n \bar{a}_k \to t \Leftarrow \Pi}$$

If $f \in DF^n$, $k > 0$, $(f\,\bar{t}_n \to r \Leftarrow P \square \Delta) \in [\mathscr{P}]_{\bot}$, $s \in Pat_{\bot}(\mathscr{U})$.

The crucial difference between $CRWL(\mathscr{D})$ and the semantic calculus is that $CRWL(\mathscr{D})$ infers the behavior of defined functions from a given program $\mathscr{P}$, rather than from a given interpretation I. This is clear from the formulation of rule $\mathbf{DF}_{\mathscr{P}}$, where $[\mathscr{P}]_{\perp}$ denotes the set $\{R\theta \mid R \in \mathscr{P}, \theta \in Sub_{\perp}(\mathscr{U})\}$ consisting of all the possible instances of the function defining rules belonging to $\mathscr{P}$.

As in the semantic calculus, we agree that no inference rule is applied in case that some textually previous rule can be used. Moreover, we also agree that the premise $P \,\square\, \Delta \Leftarrow \Pi$ in rule $\mathbf{DF}_{\mathscr{P}}$ must be understood as a shorthand for several premises $\alpha \Leftarrow \Pi$, one for each atomic statement α occurring in $P \,\square\, \Delta$. This harmless convention allows to dispense with an explicit inference rule for conjunctions.

$CRWL(\mathscr{D})$-derivations can be represented as *proof trees* whose nodes are labelled by c-statements, where each node has been inferred from its children by means of some $CRWL(\mathscr{D})$ inference rule. Concerning proof trees and their sizes, we will use the same notation and terminology already introduced for the semantic calculus in subsection 3.3, modulo the replacement of rule $\mathbf{DF}_I$ by rule $\mathbf{DF}_{\mathscr{P}}$. In particular, $T :: \mathscr{P} \vdash_{\mathscr{D}} \varphi$ will indicate that $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ is witnessed by the proof tree T.
The results in the next Subsection show a natural correspondence between $CRWL(\mathscr{D})$-derivations, logical consequence from a given $CFLP(\mathscr{D})$-program as presented in Subsection 3.3, and least models of a given $CFLP(\mathscr{D})$-program as presented in Subsection 3.4. Therefore, $CRWL(\mathscr{D})$-derivations of the form $T :: \mathscr{P} \vdash_{\mathscr{D}} G\sigma \Leftarrow S$ can be expected to exist iff the pair $S \,\square\, \sigma$ is a declaratively correct answer (in the sense of the $CFLP(\mathscr{D})$ framework) for the goal G w.r.t. the program $\mathscr{P}$. For this reason, proof trees witnessing such derivations play an important role in the mathematical proofs of soundness and completeness for the goal solving calculi cited in the last paragraph of Subsection 3.2. The next example is related to these ideas:

Example 7 The proof tree T displayed below corresponds to a $CRWL(\mathscr{R})$-derivation

$$T :: \mathscr{P} \vdash_{\mathscr{R}} takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X] \Leftarrow X > 0.5, X \leq 1$$

witnessing the declarative correctness of the computed answer previously discussed in Example 5. In order to ease an explanation given below, a subtree of T has been given the separate name T_0.

$$\begin{array}{l}
\mathbf{AC}\ takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X] \Leftarrow X > 0.5, X \leq 1 \\
\quad X > 0.5, X \leq 1 \models_{\mathscr{R}} [X] == [X] \\
\quad \mathbf{DC}\ [X] \rightarrow [X] \Leftarrow X > 0.5, X \leq 1 \\
\qquad \mathbf{RR}\ X \rightarrow X \Leftarrow X > 0.5, X \leq 1 \\
\qquad \mathbf{DC}\ [\,] \rightarrow [\,] \Leftarrow X > 0.5, X \leq 1 \\
\quad \mathbf{DF}\ takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) \rightarrow [X] \Leftarrow X > 0.5, X \leq 1 \\
\qquad \mathbf{DC}\ inInterval\ 0\ 1 \rightarrow inInterval\ 0\ 1 \Leftarrow X > 0.5, X \leq 1 \\
\qquad\quad \mathbf{RR}\ 0 \rightarrow 0 \Leftarrow X > 0.5, X \leq 1 \\
\qquad\quad \mathbf{RR}\ 1 \rightarrow 1 \Leftarrow X > 0.5, X \leq 1 \\
\qquad \mathbf{DF}\ from\ X\ 0.5 \rightarrow [\,X, X + 0.5 \mid \perp\,] \Leftarrow X > 0.5, X \leq 1 \\
\qquad\quad \mathbf{RR}\ X \rightarrow X \Leftarrow X > 0.5, X \leq 1 \\
\qquad\quad \mathbf{RR}\ 0.5 \rightarrow 0.5 \Leftarrow X > 0.5, X \leq 1 \\
\qquad\quad \mathbf{DC}\ [\,X \mid from\ (X + 0.5)\ 0.5\,] \rightarrow [\,X, X + 0.5 \mid \perp\,] \Leftarrow X > 0.5, \\
\qquad\qquad X \leq 1 \\
\qquad\qquad \mathbf{RR}\ X \rightarrow X \Leftarrow X > 0.5, X \leq 1
\end{array}$$

DF $from\ (X+0.5)\ 0.5 \rightarrow [\ X+0.5 \mid \bot\ ] \Leftarrow X > 0.5,$
$X \leq 1$
PF $X+0.5 \rightarrow X+0.5 \Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
RR $0.5 \rightarrow 0.5 \Leftarrow X > 0.5, X \leq 1$
$X > 0.5, X \leq 1 \models_{\mathscr{R}} X+0.5 \rightarrow X+0.5$
RR $0.5 \rightarrow 0.5 \Leftarrow X > 0.5, X \leq 1$
DC $[\ X+0.5 \mid from\ ((X+0.5)+0.5)\ 0.5\ ] \rightarrow$
$[\ X+0.5 \mid \bot\ ] \Leftarrow X > 0.5, X \leq 1$
PF $X+0.5 \rightarrow X+0.5 \Leftarrow X > 0.5, X \leq 1$
TI $from\ ((X+0.5)+0.5)\ 0.5 \rightarrow \bot \Leftarrow X > 0.5,$
$X \leq 1$
DF $if\ (inInterval\ 0\ 1\ X)\ [\ X \mid takeWhile\ (inInterval\ 0\ 1)$
$[X+0.5 \mid \bot]\ ]\ [\ ] \rightarrow [X] \Leftarrow X > 0.5, X \leq 1$
DF $inInterval\ 0\ 1\ X \rightarrow true \Leftarrow X > 0.5, X \leq 1$
RR $0 \rightarrow 0 \Leftarrow X > 0.5, X \leq 1$
RR $1 \rightarrow 1 \Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
DF $and\ (0 \leq X)\ (X \leq 1) \rightarrow true \Leftarrow X > 0.5, X \leq 1$
PF $0 \leq X \rightarrow true \Leftarrow X > 0.5, X \leq 1$
RR $0 \rightarrow 0 \Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
$X > 0.5, X \leq 1 \models_{\mathscr{R}} 0 \leq X \rightarrow true$
PF $X \leq 1 \rightarrow true \Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
RR $1 \rightarrow 1 \Leftarrow X > 0.5, X \leq 1$
$X > 0.5, X \leq 1 \models_{\mathscr{R}} X \leq 1 \rightarrow true$
DC $true \rightarrow true \Leftarrow X > 0.5, X \leq 1$
DC $[\ X \mid takeWhile\ (inInterval\ 0\ 1)\ [X+0.5 \mid \bot]\ ] \rightarrow [X]$
$\Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
DC $takeWhile\ (inInterval\ 0\ 1)\ [X+0.5 \mid \bot] \rightarrow [\ ]$
$\Leftarrow X > 0.5, X \leq 1$
$\ldots\ T_0\ \ldots$
DC $[\ ] \rightarrow [\ ] \Leftarrow X > 0.5, X \leq 1$
DC $[X] \rightarrow [X] \Leftarrow X > 0.5, X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5, X \leq 1$
DC $[\ ] \rightarrow [\ ] \Leftarrow X > 0.5, X \leq 1$

Note that the subtree T_0 is such that $T_0 : \mathscr{P} \vdash_{\mathscr{R}} takeWhile\ (inInterval\ 0\ 1)\ [\ X+0.5 \mid \bot\ ]$ $\rightarrow [\ ] \Leftarrow X > 0.5, X \leq 1$. **Fig.** 1 shows a graphical representation of T_0, where the common constrained part $X > 0.5, X \leq 1$ has been omitted for readability.

DF $takeWhile\ (inInterval\ 0\ 1)\ [\ X+0.5 \mid \bot\ ] \rightarrow [\ ] \Leftarrow X > 0.5, X \leq 1$
DC $inInterval\ 0\ 1 \rightarrow inInterval\ 0\ 1 \Leftarrow X > 0.5, X \leq 1$
DC $[\ X+0.5 \mid \bot\ ] \rightarrow [\ X+0.5 \mid \bot\ ] \Leftarrow X > 0.5, X \leq 1$
DF $if\ (inInterval\ 0\ 1\ (X+0.5))\ [\ X+0.5 \mid takeWhile\ (inInterval\ 0\ 1)\ \bot\ ]\ [\ ]$
$\rightarrow [\ ] \Leftarrow X > 0.5, X \leq 1$
DF $inInterval\ 0\ 1\ (X+0.5) \rightarrow false \Leftarrow X > 0.5, X \leq 1$

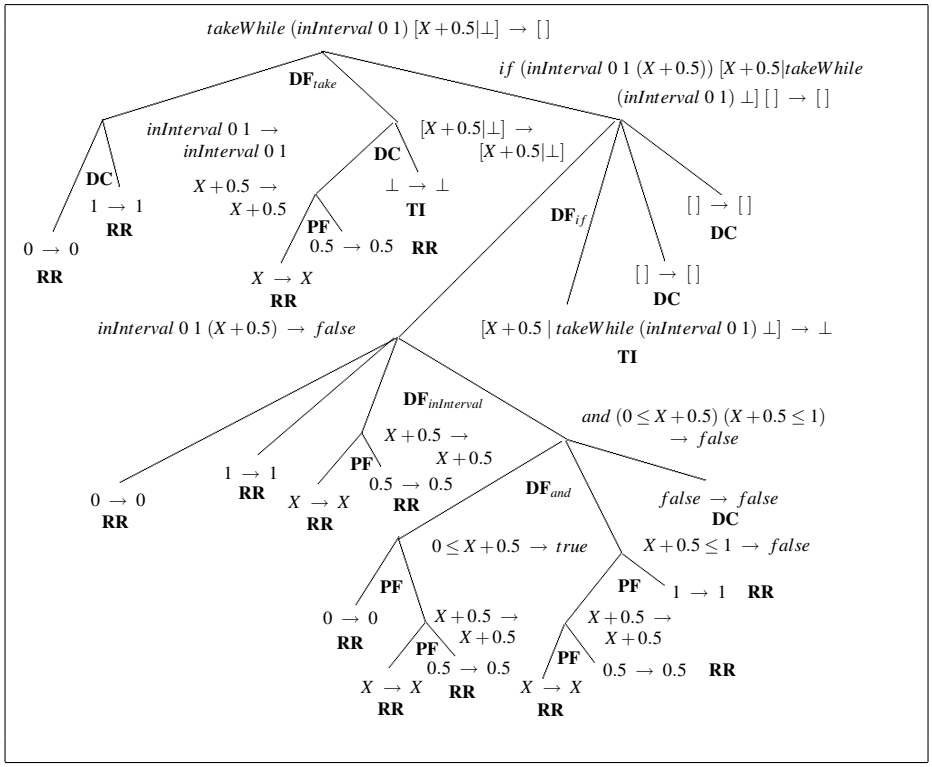

Fig. 1 A $CRWL(\mathscr{R})$-proof tree for Example 5

RR $0 \to 0 \Leftarrow X > 0.5, X \leq 1$
RR $1 \to 1 \Leftarrow X > 0.5, X \leq 1$
PF $X + 0.5 \to X + 0.5 \Leftarrow X > 0.5, X \leq 1$
DF *and* $(0 \leq X + 0.5)\ (X + 0.5 \leq 1) \to false \Leftarrow X > 0.5, X \leq 1$
  PF $0 \leq X + 0.5 \to true \Leftarrow X > 0.5, X \leq 1$
    RR $0 \to 0 \Leftarrow X > 0.5, X \leq 1$
    PF $X + 0.5 \to X + 0.5 \Leftarrow X > 0.5, X \leq 1$
    $X > 0.5, X \leq 1 \models_{\mathscr{R}} 0 \leq X + 0.5 \to true$
  PF $X + 0.5 \leq 1 \to false \Leftarrow X > 0.5, X \leq 1$
    PF $X + 0.5 \to X + 0.5 \Leftarrow X > 0.5, X \leq 1$
    RR $1 \to 1 \Leftarrow X > 0.5, X \leq 1$
    $X > 0.5, X \leq 1 \models_{\mathscr{R}} X + 0.5 \leq 1 \to false$
  DC $false \to false \Leftarrow X > 0.5, X \leq 1$
TI $[\ X + 0.5 \mid takeWhile\ (inInterval\ 0\ 1)\ \bot\ ] \to \bot \Leftarrow X > 0.5, X \leq 1$
DC $[\,] \to [\,] \Leftarrow X > 0.5, X \leq 1$
DC $[\,] \to [\,] \Leftarrow X > 0.5, X \leq 1$

Most of the properties proved in Lemma 1 for the semantic calculus translate into analogous valid properties of the rewriting calculus $CRWL(\mathscr{D})$, with the only exception of items

6 and 7 in Lemma 1, which seem to have no natural analogy in $CRWL(\mathscr{D})$. The properties are stated in the next lemma. Again, the rather technical proof can be found in Appendix B.1.

Lemma 2 *Properties of the Constrained Rewriting Calculus.*

1. *Compactness Property:* $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ *implies* $\mathscr{P}_0 \vdash_{\mathscr{D}} \varphi$ *for some finite subset* $\mathscr{P}_0 \subseteq \mathscr{P}$.
2. *Extension Property:* $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ *and* $\mathscr{P} \subseteq \mathscr{P}'$ *implies* $\mathscr{P}' \vdash_{\mathscr{D}} \varphi$.
3. *Approximation Property: For any* $e \in Exp_{\bot}(\mathscr{U})$, $t \in Pat_{\bot}(\mathscr{U})$: $\Pi \models_{\mathscr{D}} e \sqsupseteq t$ *iff there is some easy proof tree* T *such that* $T :\vdash_{\mathscr{D}} e \to t \Leftarrow \Pi$ *(derivation from empty program).*
4. *Primitive c-atoms: For any primitive atom* $p\bar{t}_n \to !\, t$, $\mathscr{P} \vdash_{\mathscr{D}} p\bar{t}_n \to !\, t \Leftarrow \Pi$ *iff* $\Pi \models_{\mathscr{D}} p\bar{t}_n \to !\, t$.
5. *Entailment Property:* $T : \mathscr{P} \vdash_{\mathscr{D}} \varphi$ *and* $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$ *implies* $T' : \mathscr{P} \vdash_{\mathscr{D}} \varphi'$ *for some proof tree* T' *such that* $|\, T' \,| \leq |\, T \,|$.

4.2 The Constraint Rewriting Logic $CRWL(\mathscr{D})$: Model Theory

In this section we investigate the relationship between $CRWL(\mathscr{D})$-derivability and the two model-theoretic semantics presented in sections 3.3 and 3.4. Our first result is the next theorem, showing a nice correspondence between $CRWL(\mathscr{D})$-derivability, strong consequence, and validity in least strong models:

Theorem 3 *Correctness Results for Strong Semantics.*
For any $CFLP(\mathscr{D})$*-program* $\mathscr{P}$ *and any c-statement* φ*, the following three conditions are equivalent:*

$$(a)\ \mathscr{P} \vdash_{\mathscr{D}} \varphi \qquad (b)\ \mathscr{P} \models^{s}_{\mathscr{D}} \varphi \qquad (c)\ \mathrm{S}_{\mathscr{P}} \vdash_{\mathscr{D}} \varphi$$

Moreover, we also have:

1. *Soundness: for any c-statement* φ, $\mathscr{P} \vdash_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \models^{s}_{\mathscr{D}} \varphi$.
2. *Completeness: for any c-statement* φ, $\mathscr{P} \models^{s}_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \vdash_{\mathscr{D}} \varphi$.
3. *Canonicity:* $\mathrm{S}_{\mathscr{P}} = \{\varphi \mid \varphi \text{ is a c-fact and } \mathscr{P} \vdash_{\mathscr{D}} \varphi\}$.

Proof A proof of the equivalence among $(a), (b), (c)$ is given in Appendix B.2. Soundness and completeness are just a trivial consequence of this equivalence. In order to prove canonicity, consider any c-fact φ. We know that $\varphi \in S_{\mathscr{P}}$ iff $S_{\mathscr{P}} \vdash_{\mathscr{D}} \varphi$, because of the *Conservation Property* from Lemma 1. On the other hand, $S_{\mathscr{P}} \vdash_{\mathscr{D}} \varphi$ iff $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ is ensured by the equivalence between (c) and (a).

Concerning the relationship between $CRWL(\mathscr{D})$-derivability and the weak semantics, most of the results (with the exception of soundness) must be restricted to ground c-statements:

Theorem 4 *Correctness Results for Weak Semantics.*
For any $CFLP(\mathscr{D})$*-program* $\mathscr{P}$ *and any ground c-statement* φ*, the following three conditions are equivalent:*

$$(a)\ \mathscr{P} \vdash_{\mathscr{D}} \varphi \qquad (b)\ \mathscr{P} \models^{w}_{\mathscr{D}} \varphi \qquad (c)\ \mathrm{W}_{\mathscr{P}} \vdash_{\mathscr{D}} \varphi$$

Moreover, we also have:

1. *Soundness: for any c-statement* φ, $\mathscr{P} \vdash_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \models^{w}_{\mathscr{D}} \varphi$.
2. *Ground Completeness: for any ground c-statement* φ, $\mathscr{P} \models^{w}_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \vdash_{\mathscr{D}} \varphi$. *This does not hold in general for arbitrary c-statements.*
3. *Ground Canonicity:* $gk_{\mathscr{D}}(\mathrm{W}_{\mathscr{P}}) = \{\varphi \mid \varphi \text{ is a ground c-fact and } \mathscr{P} \vdash_{\mathscr{D}} \varphi\}$.

Proof These results can be easily deduced from Theorem 3, using already proved relationships between weak and strong models. The detailed arguments can be found in Appendix B.2.

Using Theorems 3 and 4 we can now easily obtain two results that were announced at the end of sections 3.3 and 3.4, respectively.

Proposition 5 *Strong versus Weak Consequence.*
For any CFLP$(\mathscr{D})$*-program* $\mathscr{P}$ *and any c-fact* φ *one has:* $\mathscr{P} \models^{s}_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \models^{w}_{\mathscr{D}} \varphi$. *The reciprocal is false in some cases.*

Proof Assume that $\mathscr{P} \models^{s}_{\mathscr{D}} \varphi$. By the *Completeness Property* in Theorem 3, we can conclude that $\mathscr{P} \vdash_{\mathscr{D}} \varphi$, which implies $\mathscr{P} \models^{w}_{\mathscr{D}} \varphi$ by the *Soundness Property* in Theorem 4.

On the other hand, in the proof of Theorem 4 we have shown a *CFLP*$(\mathscr{R})$-program $\mathscr{P}$ and a non-ground c-statement φ such that $\mathscr{P} \models^{w}_{\mathscr{R}} \varphi$ and $\mathscr{P} \not\vdash_{\mathscr{R}} \varphi$, which is the same as $\mathscr{P} \not\models^{s}_{\mathscr{R}} \varphi$ because of Theorem 3.

Proposition 6 *Strong versus Weak Least Models.*
For any CFLP$(\mathscr{D})$*-program* $\mathscr{P}$ *one has* $\mathrm{W}_{\mathscr{P}} \subseteq \mathrm{S}_{\mathscr{P}}$. *The inclusion is strict in some cases.*

Proof The inclusion $W_{\mathscr{P}} \subseteq S_{\mathscr{P}}$ has been proved already in the subsection 3.4. As a counterexample for the opposite inclusion, consider an arbitrary constraint domain $\mathscr{D}$ with urelements $\mathscr{U}$, the *CFLP*$(\mathscr{D})$-program $\mathscr{P}$ consisting of one single program rule $id\ X \to X$ defining the identity function, and the c-interpretation $I = cl_{\mathscr{D}}(\{id\ t \to t \mid t \in D_{\mathscr{U}}\})$. Note that the c-fact $\varphi = (id\ X \to X)$ does not belong to I, since it is neither a trivial c-fact nor follows by $\mathscr{D}$-entailment from the ground c-facts used for defining I. On the other hand, φ belongs to $S_{\mathscr{P}}$ by the *Canonicity Property* in Theorem 3, because $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ is obviously true. Therefore, $S_{\mathscr{P}} \not\subseteq I$. But $W_{\mathscr{P}} \subseteq I$ holds by Theorem 1, because I is clearly a weak model of $\mathscr{P}$. From $S_{\mathscr{P}} \not\subseteq I$ and $W_{\mathscr{P}} \subseteq I$ we conclude $S_{\mathscr{P}} \not\subseteq W_{\mathscr{P}}$.

5 Conclusions

We have proposed a new generic scheme *CFLP*$(\mathscr{D})$ which provides a uniform foundation for the semantics of constraint functional logic programs. As main novelties w.r.t. previous related approaches, we have presented a new formalization of constraint domains, a new notion of interpretation giving rise to weak and strong semantics for programs, and a new constraint rewriting logic *CRWL*$(\mathscr{D})$ whose proof theory is sound and complete w.r.t. strong semantics, and sound and ground complete w.r.t. weak semantics.

Our results can be viewed as a natural but not trivial extension of known results on the semantics of success in the *CLP*$(\mathscr{D})$ scheme for constraint logic programming [56,40]. In comparison to previous work on constraint functional logic programming, we have improved our older *CFLP*$(\mathscr{D})$ scheme [64,65] in several respects, and we have provided a rigorous declarative semantics which was missing in other approaches.

The improvements in the new scheme provide a satisfactory foundation for our previous work on functional logic programming with disequality constraints [60,7,68] and a solid

starting point for a better foundation of our previous work on functional logic programming with multiset constraints [8,9]. Multiset constraints are outside the scope of the present paper because they use algebraic data constructors, while the $CFLP(\mathscr{D})$ scheme presented here assumes free data constructors.

The new scheme $CFLP(\mathscr{D})$ is intended as a basis for several lines of ongoing and future work. Some recent results cited in Subsection 3.2 have already shown the existence of various goal solving calculi that are sound and complete w.r.t. $CRWL(\mathscr{D})$-semantics, some of which can be regarded as a reasonable formal model for the actual behavior of implemented $CFLP(\mathscr{D})$ systems. Concerning applications of the $CFLP(\mathscr{D})$ scheme, we are working with the instance $CFLP(\mathscr{FD})$, which provides a foundation for previous work on functional logic programming with finite domain constraints, started in [36]. We plan to continue the investigation of practical constraint solving methods and applications of the resulting language. Last but not least, we are working on declarative debugging techniques for $CFLP(\mathscr{D})$-programs, following previous work for functional logic programs [17–19] as well as related work for $CLP(\mathscr{D})$ programs [96]. The Constraint Rewriting Logic $CRWL(\mathscr{D})$ already provides a formal framework for the declarative debugging of *wrong answers* [20]. We are designing an extension of $CRWL(\mathscr{D})$ which will serve as a formal framework for the declarative debugging of *missing answers*. As a byproduct of this research, we expect to obtain a formal characterization of finite failure in $CLP(\mathscr{D})$ programming, generalizing some of the already known results on the finite failure semantics of functional logic programs with disequality constraints [70,71].

A Small sample of $CFLP(\mathscr{D})$-programming in *TOY*

This appendix contains a short collection of simple programs corresponding to different instances of $CFLP(\mathscr{D})$, aiming to give the reader an impression of how constraint functional logic programs look like in practice. Programs are executable in the system *TOY* [1], and use some syntactical facilities like type declarations or `where` constructions for local definitions, usual in many functional languages.

A.1 Permutation sort in $CFLP(\mathscr{H}_{seq})$ using strict equality

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%                       Programming in CFLP(H_seq)                         %
%                                                                          %
%                  using strict equality constraints                       %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lazy generate & test as a higher order scheme.
%
% Problem: Given a generator, a tester and an input, find a solution.
%          The generator will be a non-deterministic lazy function.
% Method:  Lazy generate and test.

  findSol :: (Input -> Solution) -> (Solution -> bool) -> Input -> Solution
  findSol Generate Test Input = Candidate <== Test Candidate
                                where Candidate = Generate Input

% Intended Goals: findSol generate test input == Sol
% Application: permutation sort.
```

```
  permSort :: [int] -> [int]
  permSort = findSol permute isSorted

% The generator computes permutations of a list:

  permute []     = []
  permute [X|Xs] = [Y|permute Ys] where (Y,Ys) = split_one [X|Xs]

  split_one [X|Xs] = (X,Xs)
  split_one [X|Xs] = (Y,[X|Ys]) where (Y,Ys) = split_one Xs

% The tester accepts sorted lists:

  isSorted :: [int] -> bool
  isSorted []       = true
  isSorted [X]      = true
  isSorted [X,Y|Zs] = (X <= Y) /\ (isSorted [Y|Zs])

% /\ behaves as sequential conjunction:

  infixr 40 /\

  (/\) :: bool -> bool -> bool
  false /\ Y = false
  true  /\ Y = Y


% Goal:     permSort [3,11,8,10,1,14,12,5,6,9,2,7,15,13,4] == Xs
% Solution: Xs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
% Elapsed time in the system Toy: 8513 ms.
% Comments: This program does not attempt to be efficient, but to illustrate
%           a useful programming technique. It is remarkable that in this
%           example the use of a non-deterministic function as lazy generator
%           gives much better results than  similar 'generate and test'
%           programs written in Haskell or Prolog, which take more than one
%           hour for the goal above.
```

A.2 List difference in $CFLP(\mathscr{H}_{seq})$ using disequality constraints

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                        Programming in CFLP(H_seq)                             %
%                                                                               %
%                      using disequality constraints                            %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Problem: given two lists Xs, Ys, compute the list difference Xs -- Ys,
%          obtained by deleting from Xs the first occurrence of each member
%          of Ys, failing in case that some member of Ys does not occur
%          in Xs with the same multiplicity; i.e., compute the difference
%          Xs -- Ys viewing the lists Xs, Ys as representations of multisets.

infixl 50 --

(--) :: [A] -> [A] -> [A]

Xs -- []     = Xs
Xs -- [Y|Ys] = (delete Y Xs) -- Ys
```

```
delete :: A -> [A] -> [A]

delete Y [X|Xs] = if Y == X then Xs else [X|delete Y Xs]

% Note: (delete Y Xs) fails if Y does not occur in Xs.
%
% Moreover, the rule of "delete" is TOY code for:
%
% delete Y [X|Xs] -> if R then Xs else [X|delete Y Xs] <== seq X Y ->! R
%
% The use of the seq primitive here involves disequality constraints.
% An equivalent but less efficient definition of delete would be:
%
% delete Y [X|Xs] -> Xs              <== Y == X
% delete Y [X|Xs] -> [X|delete Y Xs] <== Y /= X
%
% Disequality constraints are apparent in this version

% Goal:      [1,2,3,2,4] -- [2,4] == Xs
% Solution:  Xs = [1,3,2]

% Goal:      ("angle" -- Xs) ++ Xs == "angel"
% Solutions: Xs = "l"; Xs = "el"; Xs = "gel"; etc.

% Application: computing permutations.
% (alternative to the function "permute" above)
% Not good for using in cooperation with "permSort",
% because "permutation" is not a lazy generator!

permutation :: [A] -> [A]

permutation Xs = Ys <== Ys -- Xs == []

% Goal:       permutation [1,2,3] == Xs
% Solutions:  Xs == [1,2,3] ;
%             Xs == [1,3,2] ;
%             Xs == [2,1,3] ;
%             Xs == [2,3,1] ;
%             Xs == [3,1,2] ;
%             Xs == [3,2,1] ;
%             no
```

A.3 Computing a mortgage in *CFLP*($\mathscr{R}$)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%                           Programming in CFLP(R)                          %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Computing a mortgage (adapted from K. Marriott and P.J. Stuckey)

% Some useful type alias.

type principal   = real      % principal

type time        = real      % number of time periods
```

```
type interest    = real      % percentual interest rate

type repayment   = real      % repayment for one time period

type balance     = real      % outstanding balance

% Computation of the new principal NP after one repayment R:
%
% NP = P + P*I - R

mortgage :: (principal,time,interest,repayment) -> balance

mortgage (P, T, I, R) = P <== T == 0
mortgage (P, T, I, R) = mortgage (P + P*I - R, T-1, I, R) <== T >= 1

% Several modes of use are possible:

% What is the balance corresponding to borrowing 1000 Euros for 10 years at
% an interest rate of 10% and repaying 150 Euros per year?
%
% Goal:     mortgage (1000, 10, 10/100, 150) == B
% Solution: B == 203.12876995000016

% How much can be borrowed in a 10 year loan at 10% with
% annual repayments of 150 Euros?
%
% Goal:     mortgage (P, 10, 10/100, 150) == 0
% Solution: P == 921.6850658557024

% What must be the relationship between the initial principal, the
% repayment an the balance in a 10 year loan at 10%?
%
% Goal:     mortgage(P, 10, 10/100, R) == B
% Solution: B == 2.5937424601*P-15.937424601000002*R
%           (a linear constraint relating P, R and B)
```

A.4 Magic series in $CFLP(\mathcal{FD})$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                          Programming in CFLP(FD)                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This example illustrates the combination of finite domain constraints
% with typical functional logic programming features, such as lazy
% evaluation and higher order functions.

% Magic series problem (adapted from Van Hentenryck)
%
% Problem: Let S = [s_0, s_1,...,s_N-1] be a non-empty finite sequence of
%          non-negative integers. S is called a N-magic series if and only
%          if there are s_i occurrences of i in S, for all i in 0,...,N-1.
%          We propose a TOY(FD) program to calculate magic series where the
%          functions 'take' and 'generate' are as defined in Example 6.

lazymagic :: int -> [int]
```

```
lazymagic N = L <== take N (generate N) == L,
                    constrain L  L  0  Cs,
                    sum L (#=) N,
                    scalar_product Cs  L (#=) N,
                    labeling [ff] L

constrain :: [int] -> [int] -> int -> [int] -> bool

constrain [] A B []         = true
constrain [X|Xs] L I [I|S2] = true <== count I L (#=) X,
                                       constrain Xs L (I+1) S2

% 'sum', 'scalar_product' and 'count' are predefined higher-order constraints,
% that can accept a FD primitive (e.g. the constraint #=) as argument.
%
% 'sum L C N' means that the summation of the elements in the list L is
% related through C with the integer N (in the example, the summation
% is constrained to be equal to N).
%
% 'scalar_product' and 'count' stand for scalar product and element counting.
% Their parameters are understood as those of sum.

% 'labeling [ff] L' controls the order in which variables of the list L are
% chosen for assignment (i.e., variable ordering). In this case, the variable
% with the smallest domain.

% Given a natural number N, (lazymagic N) returns a N-magic series.
% Since (generate N) produces an infinite list, (take N (generate N))
% requires lazy evaluation.

% A more interesting problem is to compute a list of different magic series
% of increasing lengths, starting with a given N. This is done by the recursive
% function magicfrom:

magicfrom :: int -> [[int]]

magicfrom N = [lazymagic N | magicfrom (N+1)]

% Solving the following goal requires lazy evaluation.
% The answer provides a list consisting of a 7-magic,
% a 8-magic and a 9-magic series:

% Goal:     take 3 (magicfrom 7) == L
% Solution: L ==  [ [ 3, 2, 1, 1, 0, 0, 0 ],
%                   [ 4, 2, 1, 0, 1, 0, 0, 0 ],
%                   [ 5, 2, 1, 0, 0, 1, 0, 0, 0 ] ]

% The TOY(FD) code shown below shows another way of computing a
% list of magic series of increasing lengths, this time using
% higher-order functions.

from :: int -> [int]
from N = [N | from (N+1)]

(.) :: (B -> C) -> (A -> B) -> (A -> C)
(F . G) X = F (G X)

map :: (A -> B) -> [A] -> [B]

map F [] = []
```

```
map F [X|Xs] = [F X | map F Xs]

lazyseries :: int -> [[int]] lazyseries = map lazymagic . from

% The next goal produces the same answer as the previous one:

% Goal:     take 3 (lazyseries 7) == L
% Solution: L ==  [ [ 3, 2, 1, 1, 0, 0, 0 ],
%                   [ 4, 2, 1, 0, 1, 0, 0, 0 ],
%                   [ 5, 2, 1, 0, 0, 1, 0, 0, 0 ] ]
```

B Proofs of the main results

B.1 Proofs of the main results from section 3

Proof of Lemma 1

Proof (1) *Compactness Property*. Assume a given proof tree T for $I \Vdash_{\mathscr{J}} \varphi$. Reasoning by induction on $\|T\|$ we prove the existence of some finite subset $I_0 \subseteq I$ and a proof tree T' such that $T' : cl_{\mathscr{J}}(I_0) \Vdash_{\mathscr{J}} \varphi$. We distinguish cases according to the inference rule applied at the root of T.

First, if T is an easy proof tree the property holds trivially because $T : I \Vdash_{\mathscr{J}} \varphi$ is a derivation from the trivial c-interpretation $cl_{\mathscr{J}}(\emptyset)$. Therefore, T and T' are the same easy proof tree for $cl_{\mathscr{J}}(\emptyset) \Vdash_{\mathscr{J}} \varphi$, and of course, for $cl_{\mathscr{J}}(I_0) \Vdash_{\mathscr{J}} \varphi$ with $I_0 \subseteq I$ every finite subset.

In other case, using the *induction hypothesis* and the fact that we only use almost one c-fact of I in each step of the derivation, the property is obvious for all the rest of inference rules applied at the root of T.

For example, if $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ and $T = \mathbf{DF}_I(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ for some c-fact $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in I$ such that $T_i : I \Vdash_{\mathscr{J}} e_i \to t_i \Leftarrow \Pi$ with $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : I \Vdash_{\mathscr{J}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $\|T_s\| < \|T\|$, by *induction hypothesis* we obtain $T_i' : cl_{\mathscr{J}}(I_i) \Vdash_{\mathscr{J}} e_i \to t_i \Leftarrow \Pi$ for some finite subset $I_i \subseteq I$ $(1 \leq i \leq n)$ and $T_s' : cl_{\mathscr{J}}(I_s) \Vdash_{\mathscr{J}} s\,\overline{a}_k \to t \Leftarrow \Pi$ for some finite subset $I_s \subseteq I$.

Then, we can define the finite subset $I_0 =_{def} \bigcup_{i=1}^{m} I_i \cup I_s \cup \{f\,\overline{t}_n \to s \Leftarrow \Pi\}$. We note that $I_0 \subseteq I$ and $cl_{\mathscr{J}}(I_0) = \bigcup_{i=1}^{m} cl_{\mathscr{J}}(I_i) \cup cl_{\mathscr{J}}(I_s) \cup cl_{\mathscr{J}}(\{f\,\overline{t}_n \to s \Leftarrow \Pi\})$. Moreover, we have $T_i' : cl_{\mathscr{J}}(I_0) \Vdash_{\mathscr{J}} e_i \to t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$, $T_s' : cl_{\mathscr{J}}(I_0) \Vdash_{\mathscr{J}} s\,\overline{a}_k \to t \Leftarrow \Pi$ and $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in cl_{\mathscr{J}}(I_0)$. Hence, $T' =_{def} \mathbf{DF}_{cl_{\mathscr{J}}(I_0)}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1', \ldots, T_n', T_s'])$.

(2) *Extension Property*. Assume a given proof tree T for $I \Vdash_{\mathscr{J}} \varphi$. Reasoning by induction on $\|T\|$, we prove the existence of a proof tree T' for $I' \Vdash_{\mathscr{J}} \varphi$.

First, if T is an easy proof tree then the property holds trivially because $T : I \Vdash_{\mathscr{J}} \varphi$ is a derivation from the trivial c-interpretation $cl_{\mathscr{J}}(\emptyset)$. Therefore, T and T' are the same easy proof tree for $cl_{\mathscr{J}}(\emptyset) \Vdash_{\mathscr{J}} \varphi$, and of course, for $I' \Vdash_{\mathscr{J}} \varphi$.

In other case, using the *induction hypothesis* and the fact that $I \subseteq I'$ if I is necessary in the derivation, the property is obvious for all the rest of inference rules applied at the root of T.

For example, if $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ and $T = \mathbf{DF}_I(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ for some c-fact $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in I$ s. t. $T_i : I \Vdash_{\mathscr{J}} e_i \to t_i \Leftarrow \Pi$ with $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : I \Vdash_{\mathscr{J}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $\|T_s\| < \|T\|$, by *induction hypothesis* we obtain $T_i' : I' \Vdash_{\mathscr{J}} e_i \to t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ and $T_s' : I' \Vdash_{\mathscr{J}} s\,\overline{a}_k \to t \Leftarrow \Pi$.

Moreover, since $I \subseteq I'$, we also have $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in I'$. Hence, $T' =_{def} \mathbf{DF}_{I'}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1', \ldots, T_n', T_s'])$, which verifies $T' : I' \Vdash_{\mathscr{J}} \varphi$.

(3) *Approximation Property*. In case that $Sol_{\mathscr{J}}(\Pi) = \emptyset$, $\Pi \models_{\mathscr{J}} e \sqsupseteq t$ is trivially true and $T :\Vdash_{\mathscr{J}} e \to t \Leftarrow \Pi$ with just one **TI** inference. In the rest of this proof we can assume $Sol_{\mathscr{J}}(\Pi) \neq \emptyset$ and reason by induction on the syntactic size of e. We distinguish cases for t:

- $t = \bot$. In this case, $\Pi \models_{\mathscr{J}} e \sqsupseteq \bot$ is trivially true and $T :\Vdash_{\mathscr{J}} e \to \bot \Leftarrow \Pi$ with just one **TI** inference.
- $t = u \in \mathscr{U}$. We consider several subcases for e. If $e = u$ then $\Pi \models_{\mathscr{J}} u \sqsupseteq u$ is true and $T :\Vdash_{\mathscr{J}} u \to u \Leftarrow \Pi$ with just one **RR** inference. If $e = X \in \mathscr{V}$ and $\Pi \models_{\mathscr{J}} X \sqsupseteq u$ then $T :\Vdash_{\mathscr{J}} X \to u \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathscr{J}} X \sqsupseteq u$ then $\not\Vdash_{\mathscr{J}} X \to u \Leftarrow \Pi$ (since no inference rule is applicable).

Finally, if e is neither u nor a variable then $\Pi \not\models_{\mathscr{D}} e \sqsupseteq u$. Assume a proof tree $T : \vdash_{\mathscr{D}} e \to u \Leftarrow \Pi$ (if there is no proof tree, then we are done). Since the c-interpretation is $cl_{\mathscr{D}}(\emptyset)$ and $e \neq u$, $e \notin \mathscr{V}$, the inference rule applied at the root of T must be either **PF** or **AC**. In either case, T is not easy.

- $t = X \in \mathscr{V}$. We consider several subcases for e. If $e = X$ then $\Pi \models_{\mathscr{D}} X \sqsupseteq X$ and $T : \vdash_{\mathscr{D}} X \to X \Leftarrow \Pi$ with just one **RR** inference. If e is a pattern $s \neq X$ and $\Pi \models_{\mathscr{D}} s \sqsupseteq X$ then $T : \vdash_{\mathscr{D}} s \to X \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathscr{D}} s \sqsupseteq X$ then $\not\vdash_{\mathscr{D}} s \to X \Leftarrow \Pi$ (since no inference rule is applicable).

  Finally, if e is not a pattern, we consider any $\mu \in Sol_{\mathscr{D}}(\Pi)$ such that $X\mu$ is a total pattern. Then $e\mu \sqsupseteq X\mu$ is not true and therefore $\Pi \not\models_{\mathscr{D}} e \sqsupseteq X$. Assume a proof tree $T : \vdash_{\mathscr{D}} e \to X \Leftarrow \Pi$ (if there is no proof tree, then we are done). Since the c-interpretation is $cl_{\mathscr{D}}(\emptyset)$ and e is not a pattern, the inference rule applied at the root of T must be **IR**, **PF** or **AC**. In the last two cases, T is not easy.

  In the first case, we can assume that $e = h\overline{e_m}$ is a rigid and passive expression but not a pattern. Hence $T = \mathbf{IR}(h\bar{e}_m \to X \Leftarrow \Pi, [T_1, \ldots, T_m])$, and for each $1 \leq i \leq m$, $T_i : \vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$ such that $\Pi \models_{\mathscr{D}} h\bar{t}_m \sqsupseteq X$. Then, for some $1 \leq i \leq m$, $\Pi \not\models_{\mathscr{D}} e_i \sqsupseteq t_i$. Otherwise we would have $\Pi \models_{\mathscr{D}} e_i \sqsupseteq t_i$ for all $1 \leq i \leq m$, and then $\Pi \models_{\mathscr{D}} h\bar{e}_m \sqsupseteq h\bar{t}_m$ and $\Pi \models_{\mathscr{D}} h\bar{e}_m \sqsupseteq X$, which is not the case. Fix any $1 \leq i \leq m$ such that $\Pi \not\models_{\mathscr{D}} e_i \sqsupseteq t_i$. By *induction hypothesis* (note that the size of e_i is smaller than the size of $h\bar{e}_m$), T_i is not an easy proof tree. Therefore, T is not easy either.

- $t = h\overline{t_m}$ with $t_1, \ldots, t_m$ patterns. We consider again several subcases for e. If $e = X \in \mathscr{V}$ and $\Pi \models_{\mathscr{D}} X \sqsupseteq h\bar{t}_m$ then $T : \vdash_{\mathscr{D}} X \to h\bar{t}_m \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathscr{D}} X \sqsupseteq h\bar{t}_m$ then $\not\vdash_{\mathscr{D}} X \to h\bar{t}_m \Leftarrow \Pi$ (since no inference rule is applicable).

  If $e = h\overline{e_m}$ and $\Pi \models_{\mathscr{D}} h\bar{e}_m \sqsupseteq h\bar{t}_m$ then $\Pi \models_{\mathscr{D}} e_i \sqsupseteq t_i$ for all $1 \leq i \leq m$. Hence, by *induction hypothesis* (the size of e_i is smaller than the size of $h\bar{e}_m$), $\vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$ with an easy proof tree T_i for all $1 \leq i \leq m$ and $T : \vdash_{\mathscr{D}} h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi$ with $T = \mathbf{DC}(h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$ is an easy proof true. Moreover, if $\Pi \not\models_{\mathscr{D}} h\bar{e}_m \sqsupseteq h\bar{t}_m$ then $\Pi \not\models_{\mathscr{D}} e_i \sqsupseteq t_i$ for some $1 \leq i \leq m$. Hence, by *induction hypothesis*, there is some $1 \leq i \leq m$ such that no easy proof tree T_i for $\vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$ exists. Therefore, no easy proof tree T exists for $\vdash_{\mathscr{D}} h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi$ (**DC** doesn't work and no other inference rule applies).

  Finally, if e is neither a variable nor of the form $h\bar{e}_m$ then we consider any total $\mu \in Sol_{\mathscr{D}}(\Pi)$. Clearly $e\mu \sqsupseteq (h\bar{t}_m)\mu$ is not true. Hence, $\Pi \not\models_{\mathscr{D}} e \sqsupseteq h\bar{t}_m$. If $\not\vdash_{\mathscr{D}} e \to h\bar{t}_m \Leftarrow \Pi$ we are done. If there is some proof tree $T : \vdash_{\mathscr{D}} e \to h\bar{t}_m \Leftarrow \Pi$, the inference rule applied at the root must be either **PF** or **AC** ($\mathbf{DF}_I$ cannot be used with the trivial c-interpretation $cl_{\mathscr{D}}(\emptyset)$). In any case, T is not easy.

(4) *Primitive c-atoms*. The "only if" part. By initial hypothesis, a proof tree T for $I \vdash_{\mathscr{D}} p\bar{t}_n \to !t \Leftarrow \Pi$ must have the form $T = \mathbf{AC}(p\bar{t}_n \to !t \Leftarrow \Pi, [T_1, \ldots, T_n])$, where $\Pi \models_{\mathscr{D}} p\overline{t'_n} \to !t$ for some $t'_1, \ldots, t'_n \in Pat_{\perp}(\mathscr{U})$ and $T_i : I \vdash_{\mathscr{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ are easy proof trees. Moreover, $\Pi \models_{\mathscr{D}} t_i \sqsupseteq t'_i$ $(1 \leq i \leq n)$ follows using the *Approximation Property*, and then $\Pi \models_{\mathscr{D}} p\bar{t}_n \to !t$.

Now, the "if" part. Since $\Pi \models_{\mathscr{D}} p\bar{t}_n \to !t$ by initial hypothesis and $T_i : \vdash_{\mathscr{D}} t_i \to t_i \Leftarrow \Pi$ are easy proof trees for all $1 \leq i \leq n$ using the *Approximation Property*, we can build a proof tree $T =_{def} \mathbf{AC}(p\bar{t}_n \to !t \Leftarrow \Pi, [T_1, \ldots, T_n])$ for $I \vdash_{\mathscr{D}} p\bar{t}_n \to !t \Leftarrow \Pi$.

(5) *Entailment Property*. Assume $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$ and a substitution σ which relates φ and φ' as expected by the *entailment relation* (see definition 6). We can also assume $Sol_{\mathscr{D}}(\Pi') \neq \emptyset$, otherwise $T' : I \vdash_{\mathscr{D}} \varphi'$ with an easy proof tree $T' =_{def} \mathbf{TI}(\varphi', [])$ and $\mid T \mid \geq 0 = \mid T' \mid$. Let T be a given proof tree for $I \vdash_{\mathscr{D}} \varphi$. Reasoning by induction on $\|T\|$ we prove the existence of a proof tree T' for $I \vdash_{\mathscr{D}} \varphi'$ such that $\mid T \mid \geq \mid T' \mid$. We distinguish various possible cases:

- T is an easy proof tree and $\varphi = e \to t \Leftarrow \Pi$. This covers the cases where T has some of the forms $\mathbf{TI}(\varphi, [])$, $\mathbf{RR}(\varphi, [])$, $\mathbf{SP}(\varphi, [])$ or $\mathbf{DC}(\varphi, [])$. Since T is easy, $\mathbf{DF}_I$ is not used. Therefore, $T : \vdash_{\mathscr{D}} e \to t \Leftarrow \Pi$. By the *Approximation Property*, $\Pi \models_{\mathscr{D}} e \sqsupseteq t$. This implies $\Pi\sigma \models_{\mathscr{D}} e\sigma \sqsupseteq t\sigma$. Since $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$, we know that $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e' \sqsupseteq e\sigma$ and $\Pi' \models_{\mathscr{D}} t\sigma \sqsupseteq t'$. We can conclude $\Pi' \models_{\mathscr{D}} e' \sqsupseteq t'$. By the *Approximation Property* again, there is some easy $T' : \vdash_{\mathscr{D}} e' \to t' \Leftarrow \Pi'$, and of course, $T' : I \vdash_{\mathscr{D}} e' \to t' \Leftarrow \Pi'$. Since T and T' are both easy, $\mid T \mid = 0 \geq 0 = \mid T' \mid$.

- $T = \mathbf{DC}(h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$. In this case, we know $\varphi = h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi$ with $T_i : I \vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq m)$ and $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e' \sqsupseteq (h\bar{e}_m)\sigma$, $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq t'$.

  We can assume that T is not easy; otherwise we could reason as in the previous case. Since T is not easy, $h\bar{e}_m$ is not a pattern. Then it must be the case that $e' = h\overline{e'_m}$ with $\Pi' \models_{\mathscr{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq m$ (otherwise, any total $\mu \in Sol_{\mathscr{D}}(\Pi')$ would be such that $e'\mu \sqsupseteq (h\bar{e}_m)\sigma\mu$ is not true). Moreover, $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq t'$ and $Sol_{\mathscr{D}}(\Pi') \neq \emptyset$ leave only two possible cases for t'.

First, $t' = h\overline{t'_m}$ with $\Pi' \models_{\mathscr{D}} t_i\sigma \sqsupseteq t'_i$ for all $1 \leq i \leq m$. In this case, for each $1 \leq i \leq m$ we have $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (e'_i \to t'_i \Leftarrow \Pi')$ because $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e'_i \sqsupseteq e_i\sigma$, $\Pi' \models_{\mathscr{D}} t_i\sigma \sqsupseteq t'_i$. By *induction hypothesis*, we can assume proof trees $T'_i : I \Vdash_{\mathscr{D}} e'_i \to t'_i \Leftarrow \Pi'$ with $|\, T_i \,| \geq |\, T'_i \,|$ $(1 \leq i \leq m)$. Therefore, $T' =_{def} \mathbf{DC}(h\overline{e'}_m \to h\overline{t'}_m \Leftarrow \Pi', [T'_1, \ldots, T'_m])$ verifies that $T' : I \Vdash_{\mathscr{D}} h\overline{e'}_m \to h\overline{t'}_m \Leftarrow \Pi'$ and $|\, T \,| = \sum_{i=1}^{m} |\, T_i \,| \geq \sum_{i=1}^{m} |\, T'_i \,| = |\, T' \,|$.

Second, if $t' = X \in \mathscr{V}$ with $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq X$. In this case, for each $1 \leq i \leq m$ we have $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ because $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e'_i \sqsupseteq e_i\sigma$, $\Pi' \models_{\mathscr{D}} t_i\sigma \sqsupseteq t_i\sigma$. By *induction hypothesis*, we can assume proof trees $T'_i : I \Vdash_{\mathscr{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$ with $|\, T_i \,| \geq |\, T'_i \,|$ $(1 \leq i \leq m)$. Since $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq X$, we can build the proof tree $T' =_{def} \mathbf{IR}(h\overline{e'}_m \to X \Leftarrow \Pi', [T'_1, \ldots, T'_m])$, which verifies $T' : I \Vdash_{\mathscr{D}} h\overline{e'}_m \to X \Leftarrow \Pi'$ and $|\, T \,| = \sum_{i=1}^{m} |\, T_i \,| \geq \sum_{i=1}^{m} |\, T'_i \,| = |\, T' \,|$.

– $T = \mathbf{IR}(h\bar{e}_m \to X \Leftarrow \Pi, [T_1, \ldots, T_m])$. In this case, we know $\varphi = h\bar{e}_m \to X \Leftarrow \Pi$ with $h\bar{e}_m$ a rigid and passive expression but not a pattern, $\Pi \models_{\mathscr{D}} h\bar{t}_m \sqsupseteq X$ (and hence $\Pi\sigma \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq X\sigma$), $T_i : I \Vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq m)$, and $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{D}} \Pi\sigma$ (and hence $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq X\sigma$), $\Pi' \models_{\mathscr{D}} e' \sqsupseteq (h\bar{e}_m)\sigma$, $\Pi' \models_{\mathscr{D}} X\sigma \sqsupseteq t'$ (and hence also $\Pi' \models_{\mathscr{D}} (h\bar{t}_m)\sigma \sqsupseteq t'$). Now we can reason similarly to the previous case.

– $T = \mathbf{DF}_I(f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$. Assume $k > 0$ (the case $k = 0$ is analogous and easier). In this case, we know $\varphi = f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi$ with $t \neq \bot$, and there are some c-fact $(f\bar{t}_n \to s \Leftarrow \Pi) \in I$ and some partial pattern $s \neq \bot$ such that $T_i : I \Vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : I \Vdash_{\mathscr{D}} s\bar{a}_k \to t \Leftarrow \Pi$, $\|T_s\| < \|T\|$.

Since $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$, we know $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e' \sqsupseteq (f\bar{e}_n\bar{a}_k)\sigma$, $\Pi' \models_{\mathscr{D}} t\sigma \sqsupseteq t'$ and $t' \neq \bot$ (if $t' = \bot$ then T' consists of just one **TI** step and $|\, T \,| > 0 = |\, T' \,|$). From $\Pi' \models_{\mathscr{D}} e' \sqsupseteq (f\bar{e}_n\bar{a}_k)\sigma$, it follows that $e' = f\overline{e'}_n\overline{a'}_k$ with $\Pi' \models_{\mathscr{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$ and $\Pi' \models_{\mathscr{D}} a'_j \sqsupseteq a_j\sigma$ for all $1 \leq j \leq k$ (otherwise, for any total $\mu \in Sol_{\mathscr{D}}(\Pi')$ we would have $e'\mu \sqsupseteq (f\bar{e}_n\bar{a}_k)\sigma\mu$ not true).

Using the former conditions, it is easy to check that $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ for all $1 \leq i \leq n$ and $(s\bar{a}_k \to t \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (s\sigma\overline{a'}_k \to t' \Leftarrow \Pi')$. By *induction hypothesis* (applied to T_i, T_s), we get $T'_i : I \Vdash_{\mathscr{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $|\, T_i \,| \geq |\, T'_i \,|$ $(1 \leq i \leq n)$ and $T'_s : I \Vdash_{\mathscr{D}} s\sigma\overline{a'}_k \to t' \Leftarrow \Pi'$, $|\, T_s \,| \geq |\, T'_s \,|$.

Since $(f\bar{t}_n \to s \Leftarrow \Pi) \in I$ and $(f\bar{t}_n \to s \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (f\overline{t_n\sigma} \to s\sigma \Leftarrow \Pi')$, it implies that $(f\overline{t_n\sigma} \to s\sigma \Leftarrow \Pi') \in I$ by definition of c-interpretation, with $s\sigma \neq \bot$ a partial pattern (if $s\sigma = \bot$ then the pattern s must be a variable and the deduction is not possible in the semantic calculus because $I \Vdash_{\mathscr{D}} s\bar{a}_k \to t \Leftarrow \Pi$ with $k > 0$ and $t \neq \bot$). We can build the proof tree $T' =_{def} \mathbf{DF}_I(f\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi', [T'_1, \ldots, T'_n, T'_s])$, which verifies $T' : I \Vdash_{\mathscr{D}} f\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi'$ and $|\, T \,| = 1 + \sum_{i=1}^{m} |\, T_i \,| + |\, T_s \,| \geq 1 + \sum_{i=1}^{m} |\, T'_i \,| + |\, T'_s \,| = |\, T' \,|$.

– $T = \mathbf{PF}(p\bar{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$. In this case, we know $\varphi = p\bar{e}_n \to t \Leftarrow \Pi$ and $T_i : I \Vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ with $\Pi \models_{\mathscr{D}} p\bar{t}_n \to t$.

Since $\varphi \succcurlyeq_{\mathscr{D}} \varphi'$ and $Sol_{\mathscr{D}}(\Pi') \neq \emptyset$, it must be the case that $\varphi' = p\overline{e'}_n \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{D}} \Pi\sigma$, $\Pi' \models_{\mathscr{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$, $\Pi' \models_{\mathscr{D}} t\sigma \sqsupseteq t'$. From the previous conditions, we can deduce $\Pi\sigma \models_{\mathscr{D}} (p\bar{t}_n)\sigma \to t\sigma$ and hence also $\Pi' \models_{\mathscr{D}} (p\bar{t}_n)\sigma \to t'$. We also observe that $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ $(1 \leq i \leq n)$. By *induction hypothesis*, we obtain proof trees $T'_i : I \Vdash_{\mathscr{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $|\, T_i \,| \geq |\, T'_i \,|$ $(1 \leq i \leq n)$.

Since $\Pi' \models_{\mathscr{D}} (p\bar{t}_n)\sigma \to t'$, we can build the proof tree $T' =_{def} \mathbf{PF}(p\overline{e'}_n \to t' \Leftarrow \Pi', [T'_1, \ldots, T'_n])$, which verifies $T' : I \Vdash_{\mathscr{D}} p\overline{e'}_n \to t' \Leftarrow \Pi'$ with $|\, T \,| = 1 + \sum_{i=1}^{m} |\, T_i \,| \geq 1 + \sum_{i=1}^{m} |\, T'_i \,| = |\, T' \,|$.

– $T = \mathbf{AC}(p\bar{e}_n \to\!! t \Leftarrow \Pi, [T_1, \ldots, T_n])$. Similar to the case of the rule **PF**.

(6) *Conservation Property*. Let φ be a c-fact of the form $f\bar{t}_n \to t \Leftarrow \Pi$. We prove first the "if" part. Taking into account that $T_i : I \Vdash_{\mathscr{D}} t_i \to t_i \Leftarrow \Pi$ are easy proof trees for all $1 \leq i \leq n$ by the *Approximation Property* (from the definition of the approximation ordering, $t_i \sqsupseteq t_i$ always holds for all $t_i \in Pat_{\bot}(\mathscr{V})$ and therefore $\Pi \models_{\mathscr{D}} t_i \sqsupseteq t_i$ also holds for all $1 \leq i \leq n$), we can also suppose that $t \neq \bot$ (the case $I \Vdash_{\mathscr{D}} f\bar{t}_n \to \bot \Leftarrow \Pi$ is trivial by **TI**) and build directly the deduction $\mathbf{DF}_I(f\bar{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$ using the initial hypothesis $(f\bar{t}_n \to t \Leftarrow \Pi) \in I$.

The "only if" part. First, if we suppose that $t = \bot$ or $Sol_{\mathscr{D}}(\Pi) = \emptyset$, directly $(f\bar{t}_n \to t \Leftarrow \Pi) \in I$ by definition of c-interpretation. Otherwise, by initial hypothesis $T : I \Vdash_{\mathscr{D}} f\bar{t}_n \to t \Leftarrow \Pi$ must have the form $T = \mathbf{DF}_I(f\bar{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$, where $T_i : I \Vdash_{\mathscr{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ are easy proof trees such that $(f\overline{t'}_n \to t \Leftarrow \Pi) \in I$ with $t'_1, \ldots, t'_n \in Pat_{\bot}(\mathscr{V})$. In this setting, we obtain $\Pi \models_{\mathscr{D}} t'_i \sqsubseteq t_i$ for all $1 \leq i \leq n$ using the *Approximation Property*. It follows that $(f\overline{t'}_n \to t \Leftarrow \Pi) \succcurlyeq_{\mathscr{D}} (f\bar{t}_n \to t \Leftarrow \Pi)$ and consequently $(f\bar{t}_n \to t \Leftarrow \Pi) \in I$ because I is closed under entailment by definition of c-interpretation.

(7) *Grounding Property*. The "if" part follows from the *Extension Property*, since $[I]_{\mathscr{D}} \subseteq I$. In order to prove the "only if" part, we introduce two auxiliary notions:

- A c-statement $\varphi = e \rightarrow^? t \Leftarrow \Pi$ is called *c-ground* iff Π is ground. In particular, every ground c-statement is c-ground.
- The *least grounding* of a c-ground c-statement $\varphi = e \rightarrow^? t \Leftarrow \Pi$ is the ground c-statement $\varphi^\perp = e^\perp \rightarrow^? t^\perp \Leftarrow \Pi$ obtained from φ by substituting $\perp$ in place of all the variable occurrences.

Obviously, the "only if" part follows from the more general property

GP For any c-ground statement φ, $T : I \Vdash_{\mathscr{D}} \varphi$ implies $T' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi^\perp$ with proof tree T' such that $\|T'\| \leq \|T\|$.

Let us prove **GP** by induction on $\|T\|$. We distinguish cases according to the inference rule applied at the root of T. In the rest of this proof, the notation assumed for φ in each case is the same one used in the inference rules given in Definition 8.

TI In this case, $\varphi^\perp$ is also a trivial c-statement and we can take $T' = \mathbf{TI}(\varphi^\perp, [])$.

RR If $t \in \mathscr{V}$, then $\varphi^\perp$ is a trivial c-statement and we can take $T' = \mathbf{TI}(\varphi^\perp, [])$. Otherwise $t \in \mathscr{U}$, $\varphi^\perp = \varphi$, and we can take $T' = T$.

SP If $t \in \mathscr{V}$, then $\varphi^\perp$ is a trivial c-statement and we can take $T' = \mathbf{TI}(\varphi^\perp, [])$. If $t \notin \mathscr{V}$, then $\varphi = X \rightarrow t \Leftarrow \Pi$ for some $X \in \mathscr{V}$, and $\Pi \models_{\mathscr{D}} X \sqsupseteq t$. Since X does not occur in Π, the c-statement φ must be trivial (otherwise we would have $\Pi \models_{\mathscr{D}} \perp \sqsupseteq t$, $Sat_{\mathscr{D}}(\Pi), t \notin \mathscr{V}, t \neq \perp$, which is impossible). In this case, $\varphi^\perp$ is also a trivial c-statement and we can take $T' = \mathbf{TI}(\varphi^\perp, [])$.

DC In this case we have $T = \mathbf{DC}(h\bar{e}_m \rightarrow h\bar{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$ where $T_i : I \Vdash_{\mathscr{D}} e_i \rightarrow t_i \Leftarrow \Pi$ for all $1 \leq i \leq m$. By *induction hypothesis*, there exist $\|T_i'\| \leq \|T_i\|$ such that $T_i' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} e_i^\perp \rightarrow t_i^\perp \Leftarrow \Pi$ for all $1 \leq i \leq m$. Therefore, $T' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi^\perp$ with $T' = \mathbf{DC}(h\overline{e_m^\perp} \rightarrow h\overline{t_m^\perp} \Leftarrow \Pi, [T_1', \ldots, T_m'])$.

IR In this case $\varphi^\perp$ is obviously a trivial c-statement and we can take $T' = \mathbf{TI}(\varphi^\perp, [])$.

$\mathbf{DF_I}$ Let us assume $k > 0$ (the proof for the case $k = 0$ is similar and simpler). In this case $T = \mathbf{DF}_I(f\bar{e}_n\bar{a}_k \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n, T_{n+1}])$ where $T_i : I \Vdash_{\mathscr{D}} e_i \rightarrow t_i \Leftarrow \Pi$ for all $1 \leq i \leq n$, $T_{n+1} : I \Vdash_{\mathscr{D}} s\bar{a}_k \rightarrow t \Leftarrow \Pi$ and $\bar{t}_n$, s patterns such that $(f\bar{t}_n \rightarrow s \Leftarrow \Pi) \in I$. Since I is closed under entailment, in particular under substitution, we can assume that $(f\overline{t_n^\perp} \rightarrow s^\perp \Leftarrow \Pi) \in I$ and hence also $(f\overline{t_n^\perp} \rightarrow s^\perp \Leftarrow \Pi) \in [I]_{\mathscr{D}}$, since this is a ground c-fact. By *induction hypothesis*, there exist $\|T_i'\| \leq \|T_i\|$ such that $T_i' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} e_i^\perp \rightarrow t_i^\perp \Leftarrow \Pi$ for all $1 \leq i \leq n$, $T_{n+1}' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} s^\perp\overline{a_k^\perp} \rightarrow t^\perp \Leftarrow \Pi$. Therefore, $T' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi^\perp$ with $T' = \mathbf{DF}_I(f\overline{e_n^\perp}\,\overline{a_k^\perp} \rightarrow t^\perp \Leftarrow \Pi, [T_1', \ldots, T_n', T_{n+1}'])$.

PF In this case, $T = \mathbf{PF}(p\bar{e}_n \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n])$ where $T_i : I \Vdash_{\mathscr{D}} e_i \rightarrow t_i \Leftarrow \Pi$ for all $1 \leq i \leq n$ and $\bar{t}_n$ patterns such that $\Pi \models_{\mathscr{D}} p\bar{t}_n \rightarrow t$. Since Π is ground, $\Pi \models_{\mathscr{D}} p\bar{t}_n \rightarrow t$ trivially implies $\Pi \models_{\mathscr{D}} p\overline{t_n^\perp} \rightarrow t^\perp$. Moreover, by *induction hypothesis* there exist $\|T_i'\| \leq \|T_i\|$ such that $T_i' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} e_i^\perp \rightarrow t_i^\perp \Leftarrow \Pi$ for all $1 \leq i \leq n$. Therefore, $T' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi^\perp$ with $T' = \mathbf{PF}(p\overline{e_n^\perp} \rightarrow t^\perp \Leftarrow \Pi, [T_1', \ldots, T_n'])$.

AC In this case $T = \mathbf{AC}(p\bar{e}_n \rightarrow !t \Leftarrow \Pi, [T_1, \ldots, T_n])$ where $T_i : I \Vdash_{\mathscr{D}} e_i \rightarrow t_i \Leftarrow \Pi$ for all $1 \leq i \leq n$ and $\bar{t}_n$ patterns such that $\Pi \models_{\mathscr{D}} p\bar{t}_n \rightarrow !t$. By *induction hypothesis* there exist $\|T_i'\| \leq \|T_i\|$ s. t. $T_i' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} e_i^\perp \rightarrow t_i^\perp \Leftarrow \Pi$ for all $1 \leq i \leq n$. Since Π is ground, $\Pi \models_{\mathscr{D}} p\bar{t}_n \rightarrow !t$ trivially implies $\Pi \models_{\mathscr{D}} p\overline{t_n^\perp} \rightarrow !t^\perp$. This can be the case only if $t^\perp$ is total. Therefore, t must be ground, $t^\perp = t$, and $T' : [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi^\perp$ with $T' = \mathbf{AC}(p\overline{e_n^\perp} \rightarrow !t \Leftarrow \Pi, [T_1', \ldots, T_n'])$.

Proof of Proposition 3

Proof We prove only the properties of the strong interpretation transformer $ST_{\mathscr{P}}$; the corresponding properties of $WT_{\mathscr{P}}$ can be proved similarly.

First we note that $ST_{\mathscr{P}} : \mathbb{I}_{\mathscr{D}} \rightarrow \mathbb{I}_{\mathscr{D}}$ is a well defined mapping, because for each $I \in \mathbb{I}_{\mathscr{D}}$ the image $ST_{\mathscr{P}}(I)$ is defined as $cl_{\mathscr{D}}(preST_{\mathscr{P}}(I))$, and hence a c-interpretation.

A careful inspection of Definition 11 reveals that $I \models^s_{\mathscr{D}} \mathscr{P}$ iff $preST_{\mathscr{P}}(I) \subseteq I$. On the other hand, $preST_{\mathscr{P}}(I) \subseteq I$ iff $ST_{\mathscr{P}}(I) = cl_{\mathscr{D}}(preST_{\mathscr{P}}(I)) \subseteq I$, because I is closed under $cl_{\mathscr{D}}$. Therefore, $I \models^s_{\mathscr{D}} \mathscr{P}$ iff $ST_{\mathscr{P}}(I) \subseteq I$, i.e., the strong models of $\mathscr{P}$ are the pre-fixpoints of $ST_{\mathscr{P}}$.

Finally, the fact that $ST_{\mathscr{P}}$ is continuous follows from the two items below:

1. $ST_{\mathscr{P}}$ is monotonic:

   Assume $I, J \in \mathbb{I}_{\mathscr{D}}$ such that $I \subseteq J$. Then, $preST_{\mathscr{P}}(I) \subseteq preST_{\mathscr{P}}(J)$ is an easy consequence of the *Extension Property* from Lemma 1, and we can conclude $ST_{\mathscr{P}}(I) \subseteq ST_{\mathscr{P}}(J)$.

2. $ST_{\mathscr{D}}$ preserves the lubs of non-empty directed sets:

   Assuming a non-empty directed set $\mathfrak{I} \subseteq \mathbb{I}_{\mathscr{D}}$, we must prove $ST_{\mathscr{D}}(\sqcup\mathfrak{I}) = \sqcup ST_{\mathscr{D}}(\mathfrak{I})$. The inclusion $ST_{\mathscr{D}}(\sqcup\mathfrak{I}) \supseteq \sqcup ST_{\mathscr{D}}(\mathfrak{I})$ holds because $ST_{\mathscr{D}}$ is monotonic. Since $\mathfrak{I}$ is not empty, the opposite inclusion can be rewritten as $ST_{\mathscr{D}}(\bigcup\mathfrak{I}) \subseteq \bigcup ST_{\mathscr{D}}(\mathfrak{I})$, which is obviously a consequence of $preST_{\mathscr{D}}(\bigcup\mathfrak{I}) \subseteq \bigcup ST_{\mathscr{D}}(\mathfrak{I})$. In order to prove this last inclusion, assume an arbitrarily fixed c-fact φ belonging to the set $preST_{\mathscr{D}}(\bigcup\mathfrak{I})$. Because of the way this set is defined, φ becomes its member due to the existence of finitely many other c-statements φ_i such that $\bigcup\mathfrak{I} \Vdash_{\mathscr{D}} \varphi_i$. Therefore, by the *Compactness Property* in Lemma 1, there must be some finite set of c-facts $I_0 \subseteq \bigcup\mathfrak{I}$ such that $\varphi \in preST_{\mathscr{D}}(cl_{\mathscr{D}}(I_0))$. Since $\mathfrak{I}$ is directed, there must be also some $I \in \mathfrak{I}$ such that $I_0 \subseteq I$. Since I is closed under $cl_{\mathscr{D}}$, we obtain $cl_{\mathscr{D}}(I_0) \subseteq I$, and therefore (using the *Extension Property* from Lemma 1) $\varphi \in preST_{\mathscr{D}}(I) \subseteq \bigcup ST_{\mathscr{D}}(\mathfrak{I})$.

Proof of Proposition 4

Proof We prove the two inclusions (i) and (ii) below:

1. $WT_{\mathscr{D}}([I]_{\mathscr{D}}) \subseteq [ST_{\mathscr{D}}(I)]_{\mathscr{D}}$

   Since $[ST_{\mathscr{D}}(I)]_{\mathscr{D}}$ is closed under $cl_{\mathscr{D}}$, it is sufficient to prove that $preWT_{\mathscr{D}}([I]_{\mathscr{D}}) \subseteq [ST_{\mathscr{D}}(I)]_{\mathscr{D}}$. Assume any $\varphi' \in preWT_{\mathscr{D}}([I]_{\mathscr{D}})$. Because of Definition 13 (2.), we know that $\varphi' = f\bar{t}'_n \to t'$, where $\bar{t}'_n, t'$ are ground and there exist
   (1) $(f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}$, $\eta \in GSub_{\perp}(\mathscr{U})$
   such that
   (2) $(f\bar{t}_n \to r \Leftarrow P \square \Delta)\eta$ is ground
   (3) $\bar{t}_n\eta = \bar{t}'_n$
   (4) $[I]_{\mathscr{D}} \Vdash_{\mathscr{D}} (P \square \Delta)\eta, [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} r\eta \to t'$
   By the *Extension Property* from Lemma 1, condition (4) implies
   (5) $I \Vdash_{\mathscr{D}} (P \square \Delta)\eta, I \Vdash_{\mathscr{D}} r\eta \to t'$
   From conditions (1), (3) and (5) and Definition 13 (1.), we clearly obtain that $\varphi' \in preST_{\mathscr{D}}(I) \subseteq ST_{\mathscr{D}}(I)$. Since φ' is ground, we can infer that $\varphi' \in gk_{\mathscr{D}}(ST_{\mathscr{D}}(I)) \subseteq [ST_{\mathscr{D}}(I)]_{\mathscr{D}}$ as desired.
2. $[ST_{\mathscr{D}}(I)]_{\mathscr{D}} \subseteq WT_{\mathscr{D}}([I]_{\mathscr{D}})$

   Since $WT_{\mathscr{D}}([I]_{\mathscr{D}})$ is closed under $cl_{\mathscr{D}}$, it is sufficient to prove that all the non-trivial c-facts belonging to $gk_{\mathscr{D}}(ST_{\mathscr{D}}(I))$ also belong to $WT_{\mathscr{D}}([I]_{\mathscr{D}})$. Assume any non-trivial ground c-fact $\varphi' = f\bar{t}'_n \to t' \Leftarrow \Pi' \in ST_{\mathscr{D}}(I)$. By Definition 13 (1.), there exist
   (6) $(f\bar{t}_n \to r \Leftarrow P \square \Delta) \in \mathscr{P}$, $\theta \in Sub_{\perp}(\mathscr{U})$
   (7) $\Pi \subseteq PCon_{\perp}(\mathscr{D}), t \in Pat_{\perp}(\mathscr{U})$
   such that
   (8) $I \Vdash_{\mathscr{D}} (P \square \Delta)\theta \Leftarrow \Pi, I \Vdash_{\mathscr{D}} r\theta \to t \Leftarrow \Pi$
   (9) $f\bar{t}_n\theta \to t \Leftarrow \Pi \succcurlyeq_{\mathscr{D}} f\bar{t}'_n \to t' \Leftarrow \Pi'$
   Because of condition (9) and the definition of $\mathscr{D}$*-entailment* (see Definition 6), there is some $\sigma \in Sub_{\perp}(\mathscr{U})$ such that
   (10) $\Pi' \models_{\mathscr{D}} \Pi\sigma, \Pi' \models_{\mathscr{D}} \bar{t}'_n \sqsupseteq \bar{t}_n\theta\sigma, \Pi' \models_{\mathscr{D}} t' \sqsubseteq t\sigma$
   Since φ' is non-trivial and ground, we know that Π' is ground and $\mathscr{D}$-satisfiable, and thus $\mathscr{D}$-valid. For this reason, condition (10) can be rewritten as follows:
   (11) $\models_{\mathscr{D}} \Pi\sigma, \models_{\mathscr{D}} \bar{t}'_n \sqsupseteq \bar{t}_n\theta\sigma, \models_{\mathscr{D}} t' \sqsubseteq t\sigma$
   From the first condition in (11) and the definition of $\mathscr{D}$*-entailment* we get:
   (12) $(P \square \Delta)\theta \Leftarrow \Pi \succcurlyeq_{\mathscr{D}} (P \square \Delta)\theta\sigma, r\theta \to t \Leftarrow \Pi \succcurlyeq_{\mathscr{D}} r\theta\sigma \to t\sigma$
   From the conditions (8) and (12) and the *Entailment Property* from Lemma 1, we can infer
   (13) $I \Vdash_{\mathscr{D}} (P \square \Delta)\theta\sigma, I \Vdash_{\mathscr{D}} r\theta\sigma \to t\sigma$
   Consider now $\eta \in GSub_{\perp}(\mathscr{U})$ chosen in such a way that
   (14) $(P \square \Delta)\eta = ((P \square \Delta)\theta\sigma)^{\perp}, r\eta \to t\eta = (r\theta\sigma)^{\perp} \to (t\sigma)^{\perp}$
   where the notation $(\ldots)^{\perp}$ indicates substitution of $\perp$ in place of all variable occurrences. From (13), (14) and the generalized *Grounding Property* **GP** used in the proof of Lemma 1 (7), we get:
   (15) $[I]_{\mathscr{D}} \Vdash_{\mathscr{D}} (P \square \Delta)\eta, [I]_{\mathscr{D}} \Vdash_{\mathscr{D}} r\eta \to t\eta$
   From conditions (6) and (15) and Definition 13 (2.), we obtain
   (16) $(f\bar{t}_n\eta \to t\eta) \in preWT_{\mathscr{D}}([I]_{\mathscr{D}})$
   On the other hand, from the second and third conditions in (11), the choice of η and the fact that $\bar{t}'_n$ and t' are ground, we get:
   (17) $\models_{\mathscr{D}} \bar{t}'_n \sqsupseteq \bar{t}_n\eta, \models_{\mathscr{D}} t' \sqsubseteq t\eta$
   Since Π' is ground and $\mathscr{D}$-valid, condition (17) ensure that
   (18) $(f\bar{t}_n\eta \to t\eta) \succcurlyeq_{\mathscr{D}} (f\bar{t}'_n \to t' \Leftarrow \Pi') = \varphi'$
   From conditions (16) and (18) and the *Entailment Property* from Lemma 1, we finally conclude that
   $\varphi' \in cl_{\mathscr{D}}(preWT_{\mathscr{D}}([I]_{\mathscr{D}})) = WT_{\mathscr{D}}([I]_{\mathscr{D}})$
   as desired.

B.2 Proofs of the main results from section 4

Proof of Lemma 2

Proof The proof of this lemma is similar to that of Lemma 1.

In (1) the property holds trivially because we only use almost one program rule instance of $\mathscr{P}$ in each step of the derivation.

(2) is obvious using the fact that $\mathscr{P} \subseteq \mathscr{P}'$ if some program rule instance of $\mathscr{P}$ is necessary in the derivation.

(3) and (4) are proved in the same way as the analogous properties of the semantic calculus.

Finally, in (5) we can use again induction on $\|T\|$ to prove the existence of the proof tree T' for $\mathscr{P} \vdash_{\mathscr{Q}} \varphi'$ such that $|\ T\ | \geq |\ T'\ |$. Now, the only different case is in the application of the rule $\mathbf{DF}_{\mathscr{P}}$.

If $T = \mathbf{DF}_{\mathscr{P}}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, \overline{T}, T_r, T_s])$ and $k > 0$ (the case $k = 0$ is analogous and easier), we know $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ with $t \neq \bot$, and there are some program rule instance $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathscr{P}]_\bot$ and some partial pattern $s \neq \bot$ such that $T_i : \mathscr{P} \vdash_{\mathscr{Q}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$, $\overline{T} : \mathscr{P} \vdash_{\mathscr{Q}} P \,\square\, \Delta \Leftarrow \Pi$, $\|\overline{T}\| < \|T\|$, $T_r : \mathscr{P} \vdash_{\mathscr{Q}} r \to s \Leftarrow \Pi$, $\|T_r\| < \|T\|$ and $T_s : \mathscr{P} \vdash_{\mathscr{Q}} s\,\overline{a}_k \to t \Leftarrow \Pi$, $\|T_s\| < \|T\|$.

Since $\varphi \succcurlyeq_{\mathscr{Q}} \varphi'$, we know $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathscr{Q}} \Pi\sigma$, $\Pi' \models_{\mathscr{Q}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, $\Pi' \models_{\mathscr{Q}} t\sigma \sqsupseteq t'$ and $t' \neq \bot$ (if $t' = \bot$ then T' consists of just one **TI** step and $|\ T\ | > 0 = |\ T'\ |$).

From $\Pi' \models_{\mathscr{Q}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, it follows that $e' = f\,\overline{e'}_n\overline{a'}_k$ with $\Pi' \models_{\mathscr{Q}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$ and $\Pi' \models_{\mathscr{Q}} a'_j \sqsupseteq a_j\sigma$ for all $1 \leq j \leq k$ (otherwise, for any total $\mu \in Sol_{\mathscr{Q}}(\Pi')$ we would have $e'\mu \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma\mu$ not true).

Using the former conditions, it is easy to check that $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathscr{Q}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ for all $1 \leq i \leq n$, $(P \,\square\, \Delta \Leftarrow \Pi) \succcurlyeq_{\mathscr{Q}} ((P \,\square\, \Delta)\sigma \Leftarrow \Pi')$, $(r \to s \Leftarrow \Pi) \succcurlyeq_{\mathscr{Q}} (r\sigma \to s\sigma \Leftarrow \Pi')$ and $(s\,\overline{a}_k \to t \Leftarrow \Pi) \succcurlyeq_{\mathscr{Q}} (s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi')$.

By *induction hypothesis* (applied to $T_i, \overline{T}, T_r, T_s$), we get $T'_i : \mathscr{P} \vdash_{\mathscr{Q}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $|\ T_i\ | \geq |\ T'_i\ |$ $(1 \leq i \leq n)$, $\overline{T'} : \mathscr{P} \vdash_{\mathscr{Q}} (P \,\square\, \Delta)\sigma \Leftarrow \Pi'$, $|\ \overline{T}\ | \geq |\ \overline{T'}\ |$, $T'_r : \mathscr{P} \vdash_{\mathscr{Q}} r\sigma \to s\sigma \Leftarrow \Pi'$, $|\ T_r\ | \geq |\ T'_r\ |$, and $T'_s : \mathscr{P} \vdash_{\mathscr{Q}} s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi'$, $|\ T_s\ | \geq |\ T'_s\ |$.

Since $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta)\sigma \in [\mathscr{P}]_\bot$ and $s\sigma \neq \bot$ is a partial pattern (if $s\sigma = \bot$ then the pattern s must be a variable and the deduction is not possible in the constrained rewriting calculus because $\mathscr{P} \vdash_{\mathscr{Q}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $k > 0$ and $t \neq \bot$), we can build the proof tree $T' =_{def} \mathbf{DF}_{\mathscr{P}}(f\,\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi', [T'_1, \ldots, T'_n, \overline{T'}, T'_r, T'_s])$, which verifies $T' : \mathscr{P} \vdash_{\mathscr{Q}} f\,\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi'$ and $|\ T\ | = 1 + \sum_{i=1}^{m} |\ T_i\ | + |\ \overline{T}\ | + |\ T_r\ | + |\ T_s\ | \geq 1 + \sum_{i=1}^{m} |\ T'_i\ | + |\ \overline{T'}\ | + |\ T'_r\ | + |\ T'_s\ | = |\ T'\ |$.

Proof of Theorem 3

Proof The stated result follows from the following three implications:

1. $\mathscr{P} \vdash_{\mathscr{Q}} \varphi \Rightarrow \mathscr{P} \models^s_{\mathscr{Q}} \varphi$:
   Assume $T : \mathscr{P} \vdash_{\mathscr{Q}} \varphi$. Consider any strong model $I \models^s \mathscr{P}$. We prove $I \Vdash_{\mathscr{Q}} \varphi$ reasoning by induction on $\|T\|$. The base cases correspond to $T = \mathbf{RL}(\varphi, [\,])$, where $\mathbf{RL} \in \{\mathbf{TI}, \mathbf{RR}, \mathbf{SP}\}$. These are trivial since the same tree T verifies $T : I \Vdash_{\mathscr{Q}} \varphi$.
   The inductive cases corresponding to $T = \mathbf{RL}(\varphi, [T_1, \ldots, T_n])$, where $\mathbf{RL} \in \{\mathbf{DC}, \mathbf{IR}, \mathbf{PF}, \mathbf{AC}\}$, are also straightforward, simply noticing that the same rule $\mathbf{RL}$ applies in $CRWL(\mathscr{P})$, and using the *induction hypothesis* for $T_1, \ldots, T_n$.
   The interesting case is that of the rule $\mathbf{DF}_{\mathscr{P}}$ applied at the root step. We consider the first variant of the rule $\mathbf{DF}_{\mathscr{P}}$ (the reasoning for the second one is similar).
   The tree T would have the form

   $$T = \mathbf{DF}_{\mathscr{P}}(f\,\overline{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n, \overline{T}, T_r])$$

   with $T_i : \mathscr{P} \vdash_{\mathscr{Q}} e_i \to t_i \Leftarrow \Pi$, $\overline{T} : \mathscr{P} \vdash_{\mathscr{Q}} P \,\square\, \Delta \Leftarrow \Pi$, $T_r : \mathscr{P} \vdash_{\mathscr{Q}} r \to t \Leftarrow \Pi$, where $f \in DF^n$, $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathscr{P}]_\bot$.
   By the *induction hypothesis* applied to $\overline{T}, T_r$, there must exist $\overline{T'}, T'_r$ such that $\overline{T'} : I \Vdash_{\mathscr{Q}} P \,\square\, \Delta \Leftarrow \Pi$ and $T'_r : I \Vdash_{\mathscr{Q}} r \to t \Leftarrow \Pi$. This, together with the fact that I is a strong model of $\mathscr{P}$, ensures that $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in I$. Combining this with the rest of the *induction hypothesis* which ensures the existence of trees $T_i : I \Vdash_{\mathscr{Q}} e_i \to t_i \Leftarrow \Pi$, for $i = 1 \ldots n$, we can build the tree $T' = \mathbf{DF}_I(f\,\overline{e}_n \to t \Leftarrow \Pi, [T'_1, \ldots, T'_n, \overline{T'}, T'_r])$, which proves $I \Vdash_{\mathscr{Q}} f\,\overline{e}_n \to t$.

2. $\mathscr{P} \models^s_{\mathscr{Q}} \varphi \Rightarrow S_{\mathscr{P}} \Vdash_{\mathscr{Q}} \varphi$:
   This holds simply because $S_{\mathscr{P}} \models^s \mathscr{P}$, as proved in Theorem 1.

3. $S_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \vdash_{\mathscr{D}} \varphi$:
Assume $S_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi$, where $S_{\mathscr{P}} = \bigcup_{k\in\mathbb{N}} ST_{\mathscr{P}} \uparrow^k (\bot)$. Due to the fact that $\bigcup_{k\in\mathbb{N}} ST_{\mathscr{P}} \uparrow^k (\bot) \Vdash_{\mathscr{D}} \varphi$ must be proved by a finite proof tree, and taking into account that $ST_{\mathscr{P}} \uparrow^k (\bot)$ grows with k, it is easy to see that there must exist $k \in \mathbb{N}$ such that $ST_{\mathscr{P}} \uparrow^k (\bot) \Vdash_{\mathscr{D}} \varphi$. Therefore, it suffices to prove the following:

$$\text{For all } k \in \mathbb{N}, ST_{\mathscr{P}} \uparrow^k (\bot) \Vdash_{\mathscr{D}} \varphi \Rightarrow \mathscr{P} \vdash_{\mathscr{D}} \varphi$$

We prove that by induction on k.

$\underline{k = 0:}$
Assume $ST_{\mathscr{P}} \uparrow^0 (\bot) \Vdash_{\mathscr{D}} \varphi$. We prove that $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ by induction on the structure of a tree T with $T : ST_{\mathscr{P}} \uparrow^0 (\bot) \Vdash_{\mathscr{D}} \varphi$. We distinguish cases according to the rule at the root of T:
TI, RR or **SP**: trivial, since the same rule in $CRWL(\mathscr{D})$ proves $\mathscr{P} \vdash_{\mathscr{D}} \varphi$.
DC, IR, PF or **AC**: straightforward using the *induction hypothesis*, since the same rule applies also in $CRWL(\mathscr{D})$.
$\mathbf{DF_I}$: This rule is in fact not applicable. Otherwise, the proof tree T would have the form $T = \mathbf{DF_I}(f\bar{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$, with $T_i : \mathscr{P} \vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, and where $f \in DF^n$, $(f\bar{t}_n \to t \Leftarrow \Pi) \in ST_{\mathscr{P}} \uparrow^0 (\bot)$. But $ST_{\mathscr{P}} \uparrow^0 (\bot) = \bot$, and therefore $(f\bar{t}_n \to t \Leftarrow \Pi)$ is a trivial fact, which implies that $(f\bar{e}_n \to t \Leftarrow \Pi)$ is also trivial, but then the rule **TI** could have been applied.

A similar reasoning holds for the second case of the $\mathbf{DF_I}$ rule.

$\underline{k \mapsto k+1}$:
Assume $ST_{\mathscr{P}} \uparrow^{(k+1)} (\bot) \Vdash_{\mathscr{D}} \varphi$. The *induction hypothesis* says that $ST_{\mathscr{P}} \uparrow^k (\bot) \Vdash_{\mathscr{D}} \psi \Rightarrow \mathscr{P} \vdash_{\mathscr{D}} \psi, \forall \psi$. As before, we prove $\mathscr{P} \vdash_{\mathscr{D}} \varphi$ by induction on the structure of the proof tree for $ST_{\mathscr{P}} \uparrow^{(k+1)} (\bot) \Vdash_{\mathscr{D}} \varphi$. Also as before, the interesting case is when the root step consists of an application of $\mathbf{DF_I}$, that is, when $T = \mathbf{DF_I}(f\bar{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$, with $T_i : \mathscr{P} \vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, and $f \in DF^n$, $(f\bar{t}_n \to t \Leftarrow \Pi) \in ST_{\mathscr{P}} \uparrow^{k+1} (\bot)$.

Now, since $(f\bar{t}_n \to t \Leftarrow \Pi) \in ST_{\mathscr{P}} \uparrow^{(k+1)} (\bot)$, it follows, from the definition of $ST_{\mathscr{P}} \uparrow^{(k+1)} (\bot)$, that there must exist a rule instance $(f\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathscr{P}]_\bot$ such that $r \to t \Leftarrow \Pi \in ST_{\mathscr{P}} \uparrow^k (\bot)$ and $P \,\square\, \Delta \Leftarrow \Pi \in ST_{\mathscr{P}} \uparrow^k (\bot)$. Then, by the *induction* (on k) *hypothesis*, we have $\mathscr{P} \vdash_{\mathscr{D}} r \to t \Leftarrow \Pi$ and $\mathscr{P} \vdash_{\mathscr{D}} P \,\square\, \Delta \Leftarrow \Pi$. This, together with the (proof tree) *induction hypothesis* $\mathscr{P} \vdash_{\mathscr{D}} e_i \to t_i \Leftarrow \Pi$, for $i = 1, \ldots, n$, allows to build, using $\mathbf{DF}_{\mathscr{P}}$, a derivation in $CRWL(\mathscr{D})$ for $\mathscr{P} \vdash_{\mathscr{D}} f\bar{e}_n \to t \Leftarrow \Pi$.
A similar reasoning holds for the second case of the $\mathbf{DF_I}$ rule.

Proof of Theorem 4

Proof "$(a) \Rightarrow (b)$" holds for any c-statement φ, ground or not, since:

$\mathscr{P} \vdash_{\mathscr{D}} \varphi$
$\Rightarrow$ (*Entailment Property* from Lemma 2)
$\mathscr{P} \vdash_{\mathscr{D}} \varphi\eta$ for all $\eta \in GSub_\bot(\mathscr{U})$ such that $\varphi\eta$ is ground
$\Rightarrow$ (*Canonicity Property* from Theorem 3)
$S_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi\eta$ for all $\eta \in GSub_\bot(\mathscr{U})$ such that $\varphi\eta$ is ground
$\Rightarrow$ (*Grounding Property* from Lemma 1)
$[S_{\mathscr{P}}]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi\eta$ for all $\eta \in GSub_\bot(\mathscr{U})$ such that $\varphi\eta$ is ground
$\Rightarrow$ (Theorem 2)
$W_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi\eta$ for all $\eta \in GSub_\bot(\mathscr{U})$ such that $\varphi\eta$ is ground
$\Rightarrow$ (Theorem 1 (2.), *Extension Property* from Lemma 1)
$I \Vdash_{\mathscr{D}} \varphi\eta$ for all $I \models^w_{\mathscr{D}} \mathscr{P}$ and all $\eta \in GSub_\bot(\mathscr{U})$ such that $\varphi\eta$ is ground
$\Rightarrow$ (Definition 12 (2.))
$\mathscr{P} \models^w_{\mathscr{D}} \varphi$

"$(b) \Rightarrow (c)$" holds for any ground c-statement φ, because $W_{\mathscr{P}}$ is a weak model of $\mathscr{P}$, as shown in Theorem 1.

"$(c) \Rightarrow (a)$" also holds for every ground c-statement φ:

$W_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi$
$\Rightarrow$ (Theorem 2)
$[S_{\mathscr{P}}]_{\mathscr{D}} \Vdash_{\mathscr{D}} \varphi$
$\Rightarrow$ (*Extension Property* from Lemma 1)
$S_{\mathscr{P}} \Vdash_{\mathscr{D}} \varphi$
$\Rightarrow$ (*Canonicity Property* from Theorem 3)
$\mathscr{P} \vdash_{\mathscr{D}} \varphi$

Soundness holds because we have proved the implication "$(a) \Rightarrow (b)$" for arbitrary φ. *Ground Completeness* is an immediate consequence of the two implications "$(b) \Rightarrow (c)$" and "$(c) \Rightarrow (a)$". The restriction to ground c-statements is necessary, as shown by the $CFLP(\mathscr{R})$-program:

$$\begin{aligned}\mathscr{P} =_{def} \{&notZero\,X \rightarrow true \Leftarrow X > 0,\\ &notZero\,X \rightarrow true \Leftarrow X < 0\}\end{aligned}$$

and the c-fact $\varphi =_{def} notZero\,X \rightarrow true \Leftarrow X /=_{\mathbb{R}} 0$. For this particular choice of $\mathscr{P}$ and φ we get:

- For every weak model $I \models^{w}_{\mathscr{R}} \mathscr{P}$, it is easy to see that $(notZero\,x \rightarrow true) \in I$ for all $x \in \mathbb{R} \setminus \{0\}$, which implies $I \models^{w}_{\mathscr{R}} \varphi$. Therefore, $\mathscr{P} \models^{w}_{\mathscr{R}} \varphi$.
- On the other hand, $\mathscr{P} \not\vdash_{\mathscr{R}} \varphi$, because the proof (if existing) should use the $CRWL(\mathscr{R})$-rule $\mathbf{DF}_{\mathscr{P}}$ together with some program rule, and neither of the two rules in $\mathscr{P}$ supports such an inference.

Finally, *Ground Canonicity* just follows from the equivalence between (c) and (a) and the *Conservation Property* from Lemma 1, reasoning as in the proof of Theorem 3. Note that $W_{\mathscr{P}}$ includes also some non-ground c-facts, because all c-interpretations over $\mathscr{D}$ are required to be closed under $cl_{\mathscr{D}}$. Nevertheless, for the purposes of weak semantics, the ground c-facts give all the relevant information.

Acknowledgements The development of the research presented in this paper was motivated by the kindness of WRLA'2004 organizers, who first invited the first coauthor to present an invited talk at the Workshop, and then selected the paper to be revised, extended, and submitted, to be considered for publication in a special issue of the Journal Higher-Order and Symbolic Computation (HOSC). The criticisms of three anonymous reviewers of the HOSC submission were also very helpful for improving the quality of the final version. Let our warmest thanks be presented to all these colleagues, as well as to the many other people who have supported our daily work and life in various ways during the elaboration of the paper.

References

1. Abengózar-Carneros M., Arenas-Sánchez P., Caballero-Roldán R.,Gil-Luezas A., González-Moreno J.C., Leach-Albert J., López-Fraguas F.J., Martí-Oliet N., Molina-Bravo J.M., Pimentel-Sánchez E., Rodríguez-Artalejo M., Roldán-García M.M., Ruz-Ortiz J.J., Sánchez-Hernández J.: *TOY*: A Multi-paradigm Declarative Language. Version 2.0. Technical Report, Dpto. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, February (2002). System and documentation available at `http://toy.sourceforge.net`.
2. Antoy S., Echahed R., Hanus M.: A Needed Narrowing Strategy. Proc. ACM Symp. on Principles of Programming Languages (POPL'94), Portland, ACM Press, pp. 268–279, (1994)
3. Antoy S., Echahed R., Hanus M.: A Needed Narrowing Strategy. Journal of the ACM 74 (4), pp. 776–822, (2000)
4. Aït-Kaci H., Podelski A.: A feature constraint system for logic programming with entailment. Theoretical Computer Science 122, pp. 263–283, (1994)
5. Apt K.R.: Logic Programming. In van Leeuwen J. (ed.): Handbook of Theoretical Computer Science, Vol. B, Chapter 10, Elsevier and The MIT Press, pp. 493–574, (1990)
6. Apt K.R., Gabbrielli M.: Declarative Interpretations Reconsidered. Proc. Int. Conf. on Logic Programming (ICLP'94), Santa Margherita Ligure, the MIT Press, pp. 74–89, (1994)
7. Arenas-Sánchez P., Gil-Luezas A., López-Fraguas F.J.: 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)
8. Arenas-Sánchez P., López-Fraguas F.J., Rodríguez-Artalejo M.: Embedding Multiset Constraints into a Lazy Functional Logic Language. Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'98), held jointly with the 6th Int. Conf. on Algebraic and Logic Programming (ALP'98), Pisa, Springer LNCS 1490, pp. 429–444, (1998)

9. Arenas-Sánchez P., López-Fraguas F.J., Rodríguez-Artalejo M.: 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)
10. Arenas-Sánchez P., Rodríguez-Artalejo M.: A Semantic Framework for Functional Logic Programming with Algebraic Polymorphic Types. Proc. Int. Joint Conference on Theory and Practice of Software Development (TAPSOFT'97), Springer LNCS 1214, pp. 453–464, (1997)
11. Arenas-Sánchez P., Rodríguez-Artalejo M.: A Lazy Narrowing Calculus for Functional Logic Programming with Algebraic Polymorphic Types. Proc. Int. Symp. on Logic Programming (ILPS'97), The MIT Press, pp. 53–68, (1997)
12. Arenas-Sánchez P., Rodríguez-Artalejo M.: A general framework for lazy functional logic programming with algebraic polymorphic types. Theory and Practice of Logic Programming 1(2), pp. 185–245, (2001)
13. Baader F., Nipkow T.: Term Rewriting and All That. Cambridge University Press, (1998)
14. Backofen R.: A Complete Axiomatization of a Theory with Feature and Arity Constraints. Journal of Logic Programming 24(1&2), pp. 37–71, (1995)
15. Backofen R., Smolka G.: A complete and recursive feature theory. Theoretical Computer Science 146, pp. 243–268, (1995)
16. Bossi A., Gabbrielli M., Levi G., Martelli M.: The s-Semantics Approach: Theory and Applications. Journal of Logic Programming 19&20, pp. 149–197, (1994)
17. Caballero R., López-Fraguas F.J., Rodríguez-Artalejo M.: Theoretical Foundations for the Declarative Debugging of Lazy Functional Logic Programs. Proc. of the 5th International Symposium on Functional and Logic Programming (FLOPS'2001), Springer LNCS 2024, pp. 170–184, (2001)
18. Caballero R., Rodríguez-Artalejo M.: A Declarative Debugging System for Lazy Functional Logic Programs. Electronic Notes in Theoretical Computer Science 64, 63 pages, (2002)
19. Caballero R., Rodríguez-Artalejo M.: *DDT*: A Declarative Debugging Tool for Functional Logic Languages. Proc. of the 7th International Symposium on Functional and Logic Programming (FLOPS'2004), Springer LNCS 2988, pp. 70–84, (2004)
20. Caballero R., Rodríguez-Artalejo M., del Vado-Vírseda R.: Declarative Diagnosis of Wrong Answers in Constraint Functional-Logic Programming. Proc. of the Twenty Second Interntational Conference on Logic Programming (ICLP 2006), LNCS Vol. 4079, Springer-Verlag, (2006)
21. Clark K.L.: Predicate logic as a computational formalism. Research Report DOC 79/59, Imperial College, Department of Computing, London (1979)
22. Colmerauer A.: Prolog and Infinite Trees. In Clark K.L., Tärnlud S.A. (eds.): Logic Programming, Academic Press, pp. 153–172, (1982)
23. Colmerauer A.: Equations and Inequations on Finite and Infinite Trees. Proc. of the 2nd International Conference on Fifth Generation Computer Systems, pp. 85–89, (1984)
24. Damas L., Milner R.: Principal Type Schemes for Functional Programs. Proc. ACM Symp. on Principles of Programming Languages (POPL'82), ACM Press, pp. 207–212, (1982)
25. Darlington J., Guo Y.K.: Constraint Functional Programming. Technical Report, Imperial College, November (1989)
26. Darlington J., Guo Y.K.: Constraint Equational Deduction. Proc. of 2nd Int. Workshop on Conditional and Typed Rewriting Systems (CTRS'90), Springer LNCS 516, pp. 11–14, (1991)
27. Darlington J., Guo Y.K., Pull H.: Introducing Constraint Functional Logic Programming. PHOENIX Seminar and Workshop on Declarative Programming (DP'91), Springer Workshops in Computing, pp. 20–34, (1992)
28. Darlington J., Guo Y.K., Pull H.: 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)
29. DeGroot D., Lindstrom G. (eds.): Logic Programming: Functions, Relations and Equations. Prentice-Hall, Englewood Cliffs, (1986)
30. Dershowitz N., Jouannaud J.P.: Rewrite Systems. In J.van Leeuwen (ed.), Handbook of Theoretical Computer Science, Vol. B, Chapter 6, Elsevier and The MIT Press, pp. 243–320, (1990)
31. Dershowitz N., Okada M.: A Rationale for Conditional Equational Programming. Theoretical Computer Science 75, pp. 111–138, (1990)
32. Estévez-Martín S., del Vado-Vírseda R.: Designing an Efficient Computation Strategy in $CFLP(\mathcal{FD})$ Using Definitional Trees. Proc. of the International Workshop on Curry and Functional Logic Programming (WCFLP 2005), pp. 23-31, (2005)
33. Falaschi M., Levi G., Martelli M., Palamidessi C.: Declarative Modeling of the Operational Behavior of Logic Languages. Theoretical Computer Science 69(3). pp. 289–318, (1989)
34. Falaschi M., Levi G., Martelli M., Palamidessi C.: A Model-theoretic Reconstruction of the Operational Semantics of Logic Programs. Information and Computation 102(1). pp. 86-113, (1993)

35. Fay M.J.: First-Order Unification in an Equational Theory. Proc. Workshop on Automated Deduction (CADE'79), Academic Press, pp. 161–177, (1979)
36. Fernández A.J., Hortalá-González M.T., Sáenz Pérez F.: 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)
37. Fernández A.J., Hortalá-González M.T., Sáenz Pérez F.: *TOY(FD)*: Sketch of Operational Semantics. Proc. 9th International Conference on Principles and Practice of Constraint Programming (CP'03), Springer LNCS 2833, pp. 827–831, (2003)
38. Fernández A.J., Hortalá-González M.T., Sáenz Pérez F.: *TOY(FD)*: Version 0.8 User Manual, October 27, (2003). System and documentation available at `http://toy.sourceforge.net`.
39. Gabbrielli M., Levi G.: Modeling Answer Constraints in Constraint Logic Programs. Proc. of the Eigth Int. Conf. on Logic Programming (ICLP'91), The MIT Press, pp. 238–252, (1991)
40. Gabbrielli M., Dore G.M., Levi G.: Observable Semantics for Constraint Logic Programs. Journal of Logic and Computation 5 (2), pp. 133–171, (1995)
41. González-Moreno J.C., Hortalá-González M.T., López-Fraguas F.J., Rodríguez-Artalejo M.: A Rewriting Logic for Declarative Programming. Proc. European Symp. on Programming (ESOP'96), Springer LNCS 1058, pp. 156–172, (1996)
42. González-Moreno J.C., Hortalá-González M.T., López-Fraguas F.J., Rodríguez-Artalejo M.: An Approach to Declarative Programming Based on a Rewriting Logic. Journal of Logic Programming 40(1), pp. 47–87, (1999)
43. González-Moreno J.C., Hortalá-González M.T., Rodríguez-Artalejo M.: A Higher Order Rewriting Logic for Functional Logic Programming. Proc. Int. Conf. on Logic Programming, The MIT Press, pp. 153–167, (1997)
44. González-Moreno J.C., Hortalá-González M.T., Rodríguez-Artalejo M.: 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`.
45. Gunter C.A., Scott D.: 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)
46. Hanus M.: The Integration of Functions into Logic Programming: From Theory to Practice. Journal of Logic Programming 19&20, pp. 583–628, (1994)
47. Hanus M.: A Unified Computation Model for Functional and Logic Programming. Proc. 24st ACM Symposium on Principles of Programming Languages (POPL'97), ACM Press, pp. 80–93, (1997)
48. Hanus M.: Curry: an Integrated Functional Logic Language, Version 0.8.2, March 28, (2006). `http://www-i2.informatik.uni-kiel.de/~curry/`.
49. Henz M., Smolka G., Würtz J.: Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P.V. Hentenryck (eds.), *Principles and Practice of Constraint Programming*, The MIT Press, Chapter 2, pp. 27–48, (1995)
50. Hullot J.M.: Canonical Forms and Unification. Proc. Conf. on Automated Deduction (CADE'80), Springer LNCS 87, pp. 318–334, (1980)
51. Hussmann H.: Nichtdeterministische Algebraische Spezifikationen. Ph. D. Thesis, University of Passau, (1988)
52. Hussmann H.: Nondeterministic Algebraic Specifications and Nonconfluent Term Rewriting. Journal of Logic Programming 12, pp. 237–255, (1992)
53. Hussmann H.: Non-determinism in Algebraic Specifications and Algebraic Programs. Birkhäuser Verlag, (1993)
54. Jaffar J., Lassez J.L.: Constraint Logic Programming. In Proc. ACM Symp. on Principles of Programming Languages (POPL'87), ACM Press, pp. 111–119, (1987)
55. Jaffar J., Maher M.J.: Constraint Logic Programming: A Survey. The Journal of Logic Programming 19&20, pp. 503–581, (1994)
56. Jaffar J., Maher M.J., Marriott K., Stuckey P.J.: The Semantics of Constraint Logic Programs. Journal of Logic Programming, 37 (1-3) pp. 1–46, (1998)
57. Jaffar J., Michaylov S., Stuckey P.J., Yap R.H.C.: The CLP($\mathscr{R}$) Language and System. ACM Transactions on Programming Languages and Systems, 14 (3) pp. 339–395, (1992)
58. Kirchner C., Kirchner H., Rusinowitch M.: Deduction with Symbolic Constraints. Revue Française d'Intelligence Artificielle, 4 (3) pp. 9–52, (1990)
59. Klop J.W.: Term Rewriting Systems. In Abramsky S., Gabbay D.M., Maibaum T.S.E. (eds.): Handbook of Logic in Computer Science, Vol. 2, pp. 2–116, Oxford University Press, (1992)
60. Kuchen H., López-Fraguas F.J., Moreno-Navarro J.J., Rodríguez-Artalejo M.: Implementing a Lazy Functional Logic Language with Disequality Constraints. Proc. Joint Int. Conf. and Symposium on Logic Programming (JICSLP'92), The MIT Press, pp. 207–221, (1992)

61. Lankford D.S.: Canonical inference. Technical Report ATP-32, Department of Mathematics and Computer Science, University of Texas at Austin, (1975)
62. Lloyd J.W.: Foundations of Logic Programming. 2nd. ed., Springer Verlag, (1987)
63. Loogen R., López-Fraguas F.J., Rodríguez-Artalejo M.: A Demand Driven Computation Strategy for Lazy Narrowing. Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'93), Springer LNCS 714, pp. 184–200, (1993)
64. López-Fraguas F.J.: 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)
65. López-Fraguas F.J.: Programación Funcional y Lógica con Restricciones. Ph.D. Thesis, Univ. Complutense Madrid, (1994)
66. López-Fraguas F.J., Rodríguez-Artalejo M., del Vado Vírseda R.: Constraint Functional Logic Programming Revisited. Proc. of the 5th International Workshop on Rewriting Logic and its Applications (WRLA'2004), Electronic Notes in Theoretical Computer Science 117, pp. 5–50, (2005)
67. López-Fraguas F.J., Rodríguez-Artalejo M., del Vado Vírseda R.: A Lazy Narrowing Calculus for Declarative Constraint Programming. Proc. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'04), ACM Press, pp. 43–54, (2004)
68. López-Fraguas F.J., Sánchez-Hernández J.: Disequalities May Help to Narrow. Proc. APPIA-GULP-PRODE'99, pp. 89–104, (1999)
69. López-Fraguas F.J., Sánchez-Hernández J.: *TOY*: A Multiparadigm Declarative System. Proc. RTA'99, Springer LNCS 1631, pp 244–247, (1999)
70. López-Fraguas F.J., Sánchez-Hernández J.: Failure and equality in functional logic programming. Electronic Notes in Theoretical Computer Science 86(3), 21 pages, (2003)
71. López-Fraguas F.J., Sánchez-Hernández J.: A Proof Theoretic Approach to Failure in Functional Logic Programming. Theory and Practice of Logic Programming 4(1), pp. 41–74, (2004)
72. Maher M.J.: Complete Axiomatization of the Algebras of Finite, Rational and Infinite Trees. Proc. of the Third Annual Symposium of Logic in Computer Science (LICS'88), IEEE Computer Society Press, pp. 348–357, (1988)
73. Mandel L.: Constrained lambda calculus. Aachen Verlag Shaker, (1995)
74. Marin M.: Functional Logic Programming with Distributed Constraint Solving. Ph. D. Thesis, Johannes Kepler Universität Linz, (2000)
75. Marin M., Ida T., Schreiner W.: 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)
76. Marin M., Ida T., Suzuki.T.: Cooperative Constraint Functional Logic Programming. In International Symposium on Principles of Software Evolution (IPSE'2000), pp. 223–230, November 1–2, (2000)
77. Marriott K., Stuckey P.J.: Programming with Constraints, An Introduction. The MIT Press, (1998)
78. Martí-Oliet N., Meseguer J.: Rewriting logic: roadmap and bibliography. Theoretical Computer Science 285(2), pp. 121-154 (2002)
79. Meseguer J.: Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science 96, pp. 73–155, (1992)
80. Middeldorp A., Hamoen E.: Completeness Results for Basic Narrowing. Applicable Algebra in Engineering, Communications and Computing 5, pp. 213–253, (1994)
81. Milner R.: A Theory of Type Polymorphism in Programming. Journal of Computer and Systems Sciences, 17, pp. 348–375, (1978)
82. Möller B.: On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types. Acta Informatica 22, pp. 537–578, (1985)
83. Mück A., Streicher T.: A Tiny Constrain Functional Logic Language and Its Continuation Semantics. Proc. European Symp. on Programming (ESOP'94), Springer LNCS 788, pp. 439-453, (1994)
84. Palomino Tarjuelo M.: Comparing Meseguer's Rewriting Logic with the Logic CRWL. Electronic Notes in Theoretical Computer Science 64, 22 pages, (2002)
85. Palomino Tarjuelo M.: A Comparison between two logical formalism for rewriting. In Falaschi M., Maher M. (eds.): Multiparadigm Languages and Constraint Programming (special issue). To appear in Theory and Practice of Logic Programming.
86. Robinson J.A., Sibert E.E.: LOGLISP: Motivation, Design and Implementation. In Clark K.L., Tärnlund S.A (eds.): Logic Programming, Academic Press, pp. 299–313, (1982)
87. Rodríguez-Artalejo M.: Functional and Constraint Logic Programming. In Comon H., Marché C., Treinen R. (eds.): Constraints in Computational Logics, Theory and Applications. Revised Lectures of the International Summer School CCL'99, Springer LNCS 2002, Chapter 5, pp. 202–270, (2001)
88. Saraswat V.: Concurrent Constraint Programming Languages. PhD Thesis, Carnegie Mellon University, 1989. In ACM distinguished dissertation series. The MIT press, (1993)

89. Saraswat V., Rinard M.: Concurrent Constraint Programming. Proc. of the 17th Annual Symposium on Principles of Programming Languages (POPL'90), ACM Computer Society Press, pp. 232–245, (1990)
90. Saraswat V., Rinard M., Panangaden P.: Semantic Foundations of Concurrent Constraint Programming. Proc. of the 18th Annual Symposium on Principles of Programming Languages (POPL'91), ACM Computer Society Press, pp. 333–352, (1991)
91. Scott D.S.: Domains for Denotational Semantics. Proc. ICALP'82, Springer LNCS 140, pp. 577–613, (1982)
92. SICStus Prolog user's manual, release 3.11.0, October 2003. Swedish Institute of Computer Science, Sweden. System available at `http://www.sics.se/isl/sicstus`.
93. Slagle J.R.: Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. Journal of the ACM 21(4), pp. 622–642, (1974)
94. Smolka G., Treinen R.: Records for Logic Programming. Journal of Logic Programming 18, pp. 229–258, (1994)
95. Tarski A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, pp. 285–309, (1955)
96. Tessier A., Ferrand G.: Declarative Diagnosis in the CLP Scheme. In Deransart P., Hermenegildo M., Małuszynski J. (eds.): Analysis and Visualization Tools for Constraint Programming, Chapter 5, pp. 151–174. Springer LNCS 1870, (2000)
97. del Vado Vírseda R.: 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)
98. del Vado Vírseda R.: Declarative Constraint Programming with Definitional Trees. Proc. 5th International Workshop on Frontiers of Combining Systems (FroCoS'05), Springer LNAI 3717 pp. 184–199, (2005)
99. Van Hentenryck P.: Constraint Satisfaction in Logic Programming. Logic Programming Series, The MIT Press, (1989)
100. Van Hentenryck P.: Constraint logic programming. The Knowledge Engineering Review, Vol. 6:3, pp. 151–194, (1991)
101. Van Hentenryck P., Simonis H., Dincbas M.: Constraint satisfaction using constraint logic programming. Artificial Intelligence 58, pp. 113–159, (1994)
102. Van Hentenryck P., Saraswat V., Deville Y.: Design, implementation and evaluation of the constraint language `cc(FD)`. Journal of Logic Programming 37, pp. 139–164, (1998)
103. Van Roy P., Brand P., Duchier D., Haridi S., Henz M., Schulte C.: Logic programming in the context of multiparadigm programming: the *Oz* experience. Theory and Practice of Logic Programming 3 (6), pp. 717–763, (2003)
104. Winskel G.: On Powerdomains and Modality. Theoretical Computer Science 36, pp. 127–137, (1985)