Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Electronic Notes in Theoretical Computer Science 117 (2005) 5–50
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Constraint Functional Logic Programming Revisited*

F. Javier López-Fraguas[a], Mario Rodríguez-Artalejo[a], and Rafael del Vado Vírseda[a]

[a] *Departamento de Sistemas Informáticos y Programación, UCM, Madrid, Spain*

Abstract

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

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 [47], where the CLP scheme was first introduced. The aim of the scheme was to define a family of constraint logic

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

doi:10.1016/j.entcs.2004.06.030

programming languages $CLP(\mathcal{D})$ parameterized by a constraint domain $\mathcal{D}$, in such a way that the well established results on the declarative and operational semantics of logic programs [55,3] could be lifted to all the $CLP(\mathcal{D})$ languages in an elegant and uniform way. The best updated presentation of the classical CLP semantics can be found in [49]. 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 [90,48,70].

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 [78]. Some other early proposals for the design of functional + logic languages are described in [26]. A more recent survey of the operational principles and implementation techniques used for the integration of functions into logic programming can be found in [40]. *Narrowing*, a natural combination of rewriting and unification, originally proposed as a theorem proving tool [84,54,31,43], has been used as a goal solving mechanism in functional logic languages such as Curry [41] and $\mathcal{TOY}$ [59,1]. Under various more or less restrictive conditions, several narrowing strategies are known to be complete for goal solving [28,40,72].

To our best knowledge, the first attempt of combining constraint logic programming and functional logic programming was the $CFLP(\mathcal{D})$ scheme proposed by J. Darlington, Y.K. Guo and H. Pull [24,25]. The idea behind this approach can be roughly described by the equation $CFLP(\mathcal{D}) = CLP(FP(\mathcal{D}))$, intended to mean that a $CFLP$ language over the constraint domain $\mathcal{D}$ is viewed as a CLP language over an extended constraint domain $FP(\mathcal{D})$ whose constraints include equations between expressions involving user defined functions, to be solved by narrowing. Other proposals concerning the combination of constraints with functional programming, equational deduction and lambda-calculus appeared around the same time [22,23,51,75,66].

The $CFLP$ scheme proposed by F.J. López-Fraguas in [56,57] tried to provide results on the declarative semantics of $CFLP(\mathcal{D})$ programs closer to those known for CLP. In the classical approach to CLP semantics a constraint domain is viewed as a first order structure $\mathcal{D}$, and constraints are viewed as first order formulas that can be interpreted in $\mathcal{D}$. In [56,57] programs were built as sets of constrained rewrite rules. In order to support a lazy semantics for the user defined functions, constraint domains $\mathcal{D}$ were formalized as continuous structures, with a Scott domain [83,39] as carrier, and a continuous interpre-

tation 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 [67]. This approach introduces $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{L})$, a family of languages parameterized by a constraint domain $\mathcal{D}$, a strategy $\mathcal{S}$ which defines the cooperation of several constraint solvers over $\mathcal{D}$, and a constraint lazy narrowing calculus $\mathcal{L}$ for solving constraints involving functions defined by user given constrained rewrite rules. This approach relies on solid work on higher order lazy narrowing calculi and has been implemented on top of Mathematica [68,69]. 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 $CFLP(\mathcal{D})$ languages, as in the CLP scheme, and also overcomes the limitations of our older $CFLP$ scheme in [56,57]. 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* $CRWL(\mathcal{D})$ parameterized by a constraint domain, which provides a logical characterization of program semantics. $CRWL(\mathcal{D})$ is a natural extension of the rewriting logic $CRWL$ [35,36], originally proposed 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 [44,45,46]. In comparison to Meseguer's Rewriting Logic [71], 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 [76,77], 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 [37], polymorphic types [38], algebraic data constructors [8,9,10], an ad-hoc treatment of certain kinds of constraints [6,7], and finite failure [60,61,62,63,64]. A survey of previous work on $CRWL$ can be found in [79]. 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* [80,81,82] 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 [42,92]. Our $CFLP$ scheme, however, does not deal with concurrency issues.

The reader of this paper is assumed to have some knowledge on the foundations of logic programming [55,3] and term rewriting [27,52,11]. The rest of the paper is organized as follows: section 2 presents a new formalization of constraint domains $\mathcal{D}$, tailored to the needs of constraint functional logic programming. Section 3 presents $CFLP(\mathcal{D})$ programs and their interpretations, along with results concerning the existence of least program models. Section 4 introduces the constraint rewriting logic $CRWL(\mathcal{D})$, presenting an inference system as well as correctness results w.r.t. the model-theoretic semantics given in the previous section. Section 5 summarizes conclusions and gives an overview of planned future work. Section 6 presents a small sample of $CFLP(\mathcal{D})$ programs written in the concrete syntax of the $\mathcal{TOY}$ language. Finally, section 7 includes some technical proofs that have been moved away from the main text in order to ease reading.

2 Constraint Domains

As already explained, one main aim in this paper is to overcome the limitations of our older $CFLP(\mathcal{D})$ scheme [56,57]. As a first step in this direction, we propose a new view of constraint domains $\mathcal{D}$ as structures with carrier set $GPat_{\perp}(\mathcal{U})$, consisting of ground patterns built from the symbols in a universal signature Σ and a set of urelements $\mathcal{U}$. 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 in well-known CLP language $CLP(\mathcal{R})$ [50], 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 $CFLP(\mathcal{D})$ scheme [56,57], namely the lack of a type system, can be easily overcome by adopting the approach of [38], which shows how to refine a $CRWL$-based semantics for untyped programs with a polymorphic type system in Damas-Milner's style [73,21]. 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 $\Sigma_\perp$ for the result of extending DC^0 with the special symbol $\perp$, intended to denote an undefined data value. As notational conventions, we use $c, d \in DC$, $f, g \in FS$ and $h \in DC \cup FS$, and we define the *arity* of $h \in DC^n \cup FS^n$ as $ar(h) = n$. We also assume that DC^0 includes the three constants $true$, $false$ and $success$, which are useful for representing the results returned by various primitive functions.

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

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

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 $X \in \mathcal{V}$ and $m \geq 0$, is called a *flexible expression*, while $u \in \mathcal{U}$ and $(h\, \overline{e}_m)$ with $h \in DC \cup FS$ are called *rigid expressions*. Moreover, a rigid expression $(h\, \overline{e}_m)$ is called *active* iff $h \in FS$ and $m \geq ar(h)$, and *passive* otherwise. 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 $Exp_\perp(\mathcal{U})$ are:

- $GExp_\perp(\mathcal{U})$, the set of the *ground* expressions e such that $var(e) = \emptyset$.

- $Exp(\mathcal{U})$, the set of the *total* expressions e with no occurrences of $\bot$.
- $GExp(\mathcal{U})$, the set of the ground and total expressions $GExp_\bot(\mathcal{U}) \cap Exp(\mathcal{U})$.

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

$$t ::= X \ (X \in \mathcal{V}) \mid \bot \mid u \ (u \in \mathcal{U}) \mid$$
$$(c\,\overline{t}_m) \ (c \in DC^n,\ m \leq n) \mid (f\,\overline{t}_m) \ (f \in FS^n,\ m < n)$$

Note that expressions $(f\,\overline{t}_m)$ with $f \in FS^n$, $m \geq n$, are not allowed as patterns, because they are potentially evaluable using a primitive or user given definition for function f. Patterns of the form $(f\,\overline{t}_m)$ with $f \in FS^n$, $m < n$, are used in $CRWL$ [37,38] as a convenient representation of higher order values. The subsets $Pat(\mathcal{U})$, $GPat_\bot(\mathcal{U})$, $GPat(\mathcal{U}) \subseteq Pat_\bot(\mathcal{U})$ consisting of the *total*, *ground* and *ground and total* patterns, respectively, are defined in the natural way.

Following the spirit of denotational semantics [83,39], we view $Pat_\bot(\mathcal{U})$ as the set of finite elements of a semantic domain, and we define the *information ordering* $\sqsubseteq$ as the least partial ordering over $Pat_\bot$ satisfying the following properties: $\bot \sqsubseteq t$ for all $t \in Pat_\bot(\mathcal{U})$, and $(h\,\overline{t}_m) \sqsubseteq (h\,\overline{t'}_m)$ 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 $t \in Pat_\bot(\mathcal{U})$ is maximal w.r.t. the information ordering iff t is a total pattern, i.e. $t \in Pat(\mathcal{U})$.

Any partially ordered set (shortly, poset), can be converted into a semantic domain by means of a technique called *ideal completion*; see e.g. [74]. Therefore, in the rest of this paper we will use the poset $GPat_\bot(\mathcal{U})$ 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 $CFLP(\mathcal{D})$ scheme [56,57].

For some purposes it is useful to extend the information ordering to the set of all partial expressions. This extension is simply defined as the least partial ordering over $Exp_\bot(\mathcal{U})$ which verifies $\bot \sqsubseteq e$ for all $e \in Exp_\bot(\mathcal{U})$, and $(e\,e_1) \sqsubseteq (e'\,e'_1)$ whenever $e \sqsubseteq e'$ and $e_1 \sqsubseteq e'_1$.

As usual, we define *substitutions* $\sigma \in Sub_\bot(\mathcal{U})$ as mappings $\sigma : \mathcal{V} \to Pat_\bot(\mathcal{U})$ extended to $\sigma : Exp_\bot(\mathcal{U}) \to Exp_\bot(\mathcal{U})$ in the natural way. Similarly, we consider *total substitutions* $\sigma \in Sub(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to Pat(\mathcal{U})$, *ground substitutions* $\sigma \in GSub_\bot(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to GPat_\bot(\mathcal{U})$, and *ground total substitutions* $\sigma \in GSub(\mathcal{U})$ given by mappings $\sigma : \mathcal{V} \to GPat(\mathcal{U})$. By convention, we write ε for the identity substitution, $e\sigma$ instead of $\sigma(e)$, and $\sigma\theta$ for the composition of σ and θ, such that $e(\sigma\theta) = (e\sigma)\theta$ for any $e \in Exp_\bot(\mathcal{U})$. We define the *domain* and the *variable range* of a substitution

in the usual way, namely:

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

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

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

2.2 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 2.1 Constraint Domains.

(i) 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.

(ii) A *constraint domain* of signature Γ is any structure

$$\mathcal{D} = \langle D_\mathcal{U}, \{p^\mathcal{D} \mid p \in PF\}\rangle$$

such that the carrier set $D_\mathcal{U} = GPat_\perp(\mathcal{U})$ coincides with the set of ground patterns for some set of urelements $\mathcal{U}$, and the interpretation $p^\mathcal{D}$ of each $p \in PF^n$ satisfies the following requirements:

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

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

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

Items (ii).(a),(b) in the previous definition are intended to ensure that $p^{\mathcal{D}}$ encodes the behaviour of a monotonic and continuous mapping from $\overline{D_{\mathcal{U}}}^n$, the nth power of the semantic domain obtained from $D_{\mathcal{U}}$ by ideal completion [74] into Hoare's Powerdomain $\mathcal{HP}(\overline{D_{\mathcal{U}}})$ [83,93,39]. Intuitively, one can think of $p^{\mathcal{D}}$ just as describing the behaviour 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 [36,38,10], which in turn was inspired by ideas from Hussmann [44,45,46].

Item (ii).(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 few information for returning any result different of $\bot$. 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 two examples. First we present two primitives for equality comparisons. They make sense for any constraint domain $\mathcal{D}$ built over any set of urelements $\mathcal{U}$, and are obviously radical:

Example 2.2 Two equality primitives:

(i) $eq_{\mathcal{U}}$, equality primitive for urelements, interpreted to behave as follows: $eq_{\mathcal{U}}^{\mathcal{D}}\, u\, u \to true$ for all $u \in \mathcal{U}$; $eq_{\mathcal{U}}^{\mathcal{D}}\, u\, v \to false$ for all $u, v \in \mathcal{U}$, $u \neq v$; $eq_{\mathcal{U}}^{\mathcal{D}}\, t\, s \to \bot$ otherwise.

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

In the sequel we write $\mathcal{H}_{seq}$ to denote the constraint domain built over the empty set of urelements, and having $seq^{\mathcal{D}}$ as its only primitive. The language $CFLP(\mathcal{H}_{seq})$ can be seen as a new foundation for our previous work on functional logic programming with disequality constraints [53,5,58]. On the other hand, $\mathcal{H}_{seq}$ is analogous to the extension of the Herbrand domain with disequality constraints, introduced by A. Colmerauer [19,20] as one of the first constraint extensions of logic programming, and later investigated by M. J. Maher [65]. Some important differences must be noted, however. Firstly, the carrier set of $\mathcal{H}_{seq}$ 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 $\mathcal{H}_{seq}$ 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 $CFLP(\mathcal{D})$ scheme are always expressed by means of primitive functions with radical semantics.

The next example presents a constraint domain $\mathcal{R}$ similar to the one used in the well known constraint logic language $CLP(\mathcal{R})$ [50,48,70]. In the CLP case, the carrier set of $\mathcal{R}$ 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 2.3 The constraint domain $\mathcal{R}$ has the carrier set $D_{\mathbb{R}} = GPat_{\perp}(\mathbb{R})$ and the radical primitives defined below. We apply some of them in infix notation for convenience.

(i) $eq_{\mathbb{R}}$, equality primitive for real numbers, interpreted as in Example 2.2.

(ii) seq, strict equality primitive for ground patterns over the real numbers, interpreted as in Example 2.2.

(iii) $+$, $*$, for addition and multiplication, interpreted to behave as follows: $x +^{\mathcal{R}} y \to x +^{\mathbb{R}} y$ for all $x, y \in \mathbb{R}$; $t +^{\mathcal{R}} s \to \perp$ whenever $t \notin \mathbb{R}$ or $s \notin \mathbb{R}$; and analogously for $*^{\mathcal{R}}$.

(iv) $<, \leq, >, \geq$, for numeric comparisons, interpreted to behave as follows: $x <^{\mathcal{R}} y \to true$ for all $x, y \in \mathbb{R}$ with $x <^{\mathbb{R}} y$; $x <^{\mathcal{R}} y \to false$ for all $x, y \in \mathbb{R}$ with $x \geq^{\mathbb{R}} y$; $t <^{\mathcal{R}} s \to \perp$ whenever $t \notin \mathbb{R}$ or $s \notin \mathbb{R}$; and analogously for $\leq^{\mathcal{R}}$, $>^{\mathcal{R}}$, $\geq^{\mathcal{R}}$. In the sequel, $e_1 < e_2$ abbreviates $e_1 < e_2 \to! \; true$, $e_1 \geq e_2$ abbreviates $e_1 < e_2 \to! \; false$ and analogously for the rest of primitives.

Other constraint domains known for their practical value in constraint programming include feature tree constraints [2,85,12,13], which can be viewed as an extension of Colmerauer's rational trees [19,20], and finite domain constraints [89,90,91,92]. These two kinds of constraints play an important role in the multiparadigm programming language Oz [42]. Finite domain constraints have been recently used for solving combinatorial problems in constraint functional logic programming [32], using an extension of the $\mathcal{TOY}$ system [59,1] which we hope to formalize as an instance of the $CFLP(\mathcal{D})$ scheme in some future work.

2.3 Constraints over a given Constraint Domain

Assuming an arbitrarily fixed constraint domain $\mathcal{D}$ built over a certain set of urelements $\mathcal{U}$, 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 2.4 Syntax of Constraints.

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

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

(iii) *Atomic Constraints* have the syntactic form $p\,\overline{e}_n \rightarrow\!!\, t$, with $p \in PF^n$, $e_i \in Exp_\perp(\mathcal{U})$ for all $1 \leq i \leq n$, and $t \in Pat(\mathcal{U})$. The special constants $\square$ and $\blacksquare$ are also atomic constraints.

(iv) *Constraints* are built from atomic constraints by means of logical conjunction $\wedge$ and existential quantification $\exists$.

In the sequel we use the following notations:

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

We also write $DCon_\perp(\mathcal{D})$ for the set of all the user defined constraints δ over $\mathcal{D}$, as well as $DGCon_\perp(\mathcal{D})$, $DCon(\mathcal{D})$ and $DGCon(\mathcal{D})$ for the subsets of $DCon_\perp(\mathcal{D})$ 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 $CFLP(\mathcal{D})$-programs. The semantics of

primitive constraints depends on the notion of solution, presented in the next definition.

Definition 2.5 Solutions of Primitive Constraints.

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

(ii) The set of solutions of $\pi \in PCon_\bot(\mathcal{D})$ is a subset $Sol_\mathcal{D}(\pi) \subseteq Val_\bot(\mathcal{D})$ recursively defined as follows:
  (a) $Sol_\mathcal{D}(\square) = Val_\bot(\mathcal{D})$.
  (b) $Sol_\mathcal{D}(\blacksquare) = \emptyset$.
  (c) $Sol_\mathcal{D}(p\,\overline{t}_n \to\!!\, t) = \{\eta \in Val_\bot(\mathcal{D}) \mid t\eta \text{ is total and } p^\mathcal{D}\,\overline{t}_n\eta \to t\eta\}$.
  (d) $Sol_\mathcal{D}(\pi_1 \wedge \pi_2) = Sol_\mathcal{D}(\pi_1) \cap Sol_\mathcal{D}(\pi_2)$.
  (e) $Sol_\mathcal{D}(\exists X\pi) = \{\eta \in Val_\bot(\mathcal{D}) \mid \eta' \in Sol_\mathcal{D}(\pi) \text{ for some } \eta' =_{\setminus\{X\}} \eta\}$

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

According to item (ii).(c) in this definition, the solutions of a primitive atomic constraint $p\,\overline{t}_n \to\!!\, t$ are those valuations for which $p\,\overline{t}_n$ can return a total value which matches the total pattern t. For instance, $\eta \in Sol_\mathcal{R}(X+Y \to\!!\, 5)$ 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.

As argued also in [63] for the particular case of strict equality and disequality constraints over constructor terms, a functional view of atomic constraints as proposed here has some advantages w.r.t. the traditional view of atomic constraints as predicates. In order to clarify this point, 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 $CFLP(\mathcal{R})$ 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 $\mathcal{R}$ 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:

- $p\,\overline{e}_n$ abbreviates $p\,\overline{e}_n \rightarrow! \; success$.
- $e_1 =_{\mathcal{U}} e_2$ abbreviates $eq_{\mathcal{U}}\, e_1\, e_2 \rightarrow! \; true$.
- $e_1 \not=_{\mathcal{U}} e_2$ abbreviates $eq_{\mathcal{U}}\, e_1\, e_2 \rightarrow! \; false$.
- $e_1 == e_2$ abbreviates $seq\, e_1\, e_2 \rightarrow! \; true$.
- $e_1 \not= e_2$ abbreviates $seq\, e_1\, e_2 \rightarrow! \; false$.

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

Definition 2.6 Primitive Semantic Notions.
Assuming a finite set $\Pi \subseteq PCon_\perp(\mathcal{D})$ of primitive constraints, a primitive constraint $\pi \in PCon_\perp(\mathcal{D})$, expressions $e, e' \in Exp_\perp(\mathcal{U})$, patterns $\overline{t}_n, t \in Pat_\perp(\mathcal{U})$, and a primitive function symbol $p \in PF^n$, we define:

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

(ii) π is a consequence of Π in $\mathcal{D}$ (in symbols, $\Pi \models_{\mathcal{D}} \pi$) iff $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\pi)$.

(iii) π is valid in $\mathcal{D}$ (in symbols, $\models_{\mathcal{D}} \pi$) iff $\emptyset \models_{\mathcal{D}} \pi$, which is obviously equivalent to $Sol_{\mathcal{D}}(\pi) = Val_\perp(\mathcal{D})$.

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

(v) $e \sqsubseteq e'$ is valid in $\mathcal{D}$ (in symbols, $\models_{\mathcal{D}} e \sqsubseteq e'$) iff $\emptyset \models_{\mathcal{D}} e \sqsubseteq e'$, which is obviously equivalent to requiring $e\eta \sqsubseteq e'\eta$ to hold for all $\eta \in Val_\perp(\mathcal{D})$.

(vi) $\Pi \models_{\mathcal{D}} e \sqsupseteq e'$ and $\models_{\mathcal{D}} e \sqsupseteq e'$ are defined analogously.

(vii) $p\,\overline{t}_n \rightarrow t$ is a consequence of Π in $\mathcal{D}$ (in symbols, $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \rightarrow t$) iff $p^{\mathcal{D}}\,\overline{t}_n\eta \rightarrow t\eta$ holds for all $\eta \in Sol_{\mathcal{D}}(\Pi)$.

(viii) $p\,\overline{t}_n \rightarrow t$ is valid in $\mathcal{D}$ (in symbols, $\models_{\mathcal{D}} p\,\overline{t}_n \rightarrow t$) iff $\emptyset \models_{\mathcal{D}} p\,\overline{t}_n \rightarrow t$ iff $p^{\mathcal{D}}\,\overline{t}_n\eta \rightarrow t\eta$ holds for all $\eta \in Val_\perp(\mathcal{D})$.

Items (iv)–(viii) in the previous definition will be needed for defining some logical inference rules in sections 3.2 and 4.1. Note that the statement $p\,\overline{t}_n \rightarrow t$ used in items (vii) and (viii) is intended to mean that evaluation of the primitive function call $p\,\overline{t}_n$ is able to return a result t. In sections 3.2 and 4.1 this idea will be generalized to *production statements* of the form $e \rightarrow t$ (with $e \in Exp_\perp(\mathcal{U})$ and $t \in Pat_\perp(\mathcal{U})$), intended to mean that evaluation of the expression e can return the value t.

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

The CLP scheme, originally introduced by Jaffar and Lassez [47], served the purpose of defining a family of constraint logic programming languages $CLP(\mathcal{D})$ parameterized by a constraint domain $\mathcal{D}$, in such a way that the well established results on the semantics of logic programs could be lifted to all the $CLP(\mathcal{D})$ languages in an elegant and uniform way; see [49] for an updated presentation. Previous work on $CFLP$ schemes, including our old scheme $CFLP(\mathcal{D})$ [56,57] 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 $CFLP(\mathcal{D})$ 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 $CFLP(\mathcal{D})$-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 $CFLP(\mathcal{D})$-Programs and Goals

In the sequel we assume an arbitrarily fixed constraint domain $\mathcal{D}$ built over a set of urelements $\mathcal{U}$. As $CFLP(\mathcal{D})$-program we allow any set $\mathcal{P}$ 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

$$R: \; f\,\overline{t}_n \;\rightarrow r \;\Leftarrow P \,\square\, \Delta$$

and is required to satisfy the conditions listed below:

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

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

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

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

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

  (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 (i) 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

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

where $\overline{Y} = var(P \,\square\, \Delta) \setminus var(f\,\overline{t}_n \rightarrow r)$. The admissibility conditions (iv).(a), (b) and (c) are best understood by thinking of each production $e_i \rightarrow s_i$ 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 local definitions 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 $CFLP(\mathcal{R})$-program $\mathcal{P}$. 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.

Example 3.1 Splitting a list of numbers in $CFLP(\mathcal{R})$:

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

Function *case* in this example shows that an empty conditional part can be omitted when writing program rules. Section 6 includes a small sample of $CFLP$ programs over the constraint domains $\mathcal{H}_{seq}$ and $\mathcal{R}$, which can be executed in the $\mathcal{TOY}$ system and are written in $\mathcal{TOY}$'s concrete syntax.

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

Coming back to Example 3.1, the expected computed answers for the goal $G : split\,[1.2,\ X,\ -0.25] == (Ys, Zs)$ are

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

Note that in this case $Sol_{\mathcal{R}}(S_1) \subseteq Sol_{\mathcal{P}}(G\sigma_1)$ amounts to $Sol_{\mathcal{R}}(X\ >\ 0) \subseteq Sol_{\mathcal{P}}(split\,[1.2,\ X,\ -0.25]\ ==\ ([1.2,\ X],\ [-0.25]))$, which is intuitively true; and analogously for the second computed answer. In the general case, the meaning of the requirement $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{P}}(G\sigma)$ depends on the semantics for $CFLP(\mathcal{D})$-programs to be developed in the rest of this paper. The formalization of a *constrained lazy narrowing calculus* for solving $CFLP(\mathcal{D})$-goals is left for future work.

3.2 Interpretations and Models for $CFLP(\mathcal{D})$-Programs

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

Definition 3.2 $\mathcal{D}$-algebras.
Assume a constraint domain $\mathcal{D}$ with sets of urelements $\mathcal{U}$. A $\mathcal{D}$-algebra is any structure of the form

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

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

(i) $f^{\mathcal{A}} \subseteq D_{\mathcal{U}}^n \times D_{\mathcal{U}}$, which boils down to $f^{\mathcal{A}} \subseteq D_{\mathcal{U}}$ in the case $n = 0$. The notation $f^{\mathcal{A}}\,\overline{t}_n \to t$ indicates that $(\overline{t}_n,\ t) \in f^{\mathcal{A}}$. In the case $n = 0$, this notation boils down to $f^{\mathcal{A}} \to t$.

(ii) $f^{\mathcal{A}}$ behaves monotonically in its arguments and antimonotonically in its result; i.e., whenever $f^{\mathcal{A}}\,\overline{t}_n \to t$, $\overline{t}_n \sqsubseteq \overline{t'}_n$ and $t \sqsupseteq t'$ one also has $f^{\mathcal{A}}\,\overline{t'}_n \to t'$.

Similarly as in Definition 2.1, the monotonicity conditions in item (ii) are intended to capture the behaviour of a possibly non-deterministic function over finite data elements. The radicality condition in Definition 2.1 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 $CFLP(\mathcal{D})$-programs could be developed on the basis of $\mathcal{D}$-algebras. This approach would be analogous to the $\mathcal{D}$-interpretations used in the traditional semantics of $CLP(\mathcal{D})$-programs [49], and also formally similar to the structures used as interpretations for functional logic programs in our previous $CFLP(\mathcal{D})$ scheme [56,57] and previous work based on the logic $CRWL$ [36,38,10].

We have nevertheless decided to abandon $\mathcal{D}$-algebras in favour of a more expressive approach, motivated by the π-interpretations for $CLP(\mathcal{D})$-programs proposed in [33,34]. Roughly speaking, π-interpretations in the CLP setting are sets of facts of the form $p\,\overline{t}_n \Leftarrow \Pi$, intended to mean that the user defined atom $p\,\overline{t}_n$ is valid for any valuation which is a solution of the primitive constraint Π. As shown in [33,34], π-interpretations can be used as a basis for three different program semantics $\mathcal{S}_i$ $(i = 1, 2, 3)$, characterizing *valid ground goals*, *valid answers for goals* and *computed answers for goals*, respectively. In fact, the $\mathcal{S}_i$ semantics are the CLP counterpart of previously known semantics for logic programming, namely the least ground Herbrand model semantics [3,55], the open Herbrand model semantics, also known as $\mathcal{C}$-semantics [18,30] and the $\mathcal{S}$-semantics [29,14]. A very concise and readable overview of these semantics can be found in [4].

In order to generalize π-interpretations to $CFLP(\mathcal{D})$ languages, we consider sets of facts of the form $f\,\overline{t}_n \to t \Leftarrow \Pi$, intended to describe the behaviour of user defined functions $f \in DF^n$. We will use this class of interpretations for defining two different semantics, corresponding to $\mathcal{S}_1$ and $\mathcal{S}_2$, which we will call the *weak* and *strong* semantics, respectively. In future work on constrained lazy narrowing for goal solving in $CFLP(\mathcal{D})$ languages, we expect that the strong semantics will provide a characterization of valid answers for goals, including computed answers as a particular case.

Note that a $CFLP(\mathcal{D})$ analogous of the $\mathcal{S}_3$ semantics would characterize *exactly* the computed answers, being therefore dependent on the choice of a particular narrowing strategy for goal solving; a complication which does not exist in the CLP setting. The scope of the present paper is limited to results which make sense independently of any particular goal solving method. Therefore, we present no results on $\mathcal{S}_3$-like semantics.

In order to define an analogous of π-interpretations for $CFLP(\mathcal{D})$-programs, we must first introduce some preliminary notions.

Definition 3.3 Constrained Statements and $\mathcal{D}$-entailment.
Let $\mathcal{D}$ be any fixed constraint domain over a set of urelemets $\mathcal{U}$. In what follows we assume partial patterns t, $t_i \in Pat_\perp(\mathcal{U})$, partial expressions e, $e_i \in Exp_\perp(\mathcal{U})$, and a finite set $\Pi \subseteq PCon_\perp(\mathcal{D})$ of primitive constraints.

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

   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 $\to$.

(ii) Given two c-statements φ and φ', we say that φ $\mathcal{D}$-entails φ' (in symbols, $\varphi \succcurlyeq_\mathcal{D} \varphi'$) iff one of the two following cases holds:
   (a) $\varphi = e \to t \Leftarrow \Pi$, $\varphi' = e' \to t' \Leftarrow \Pi'$, and there is some $\sigma \in Sub_\perp(\mathcal{U})$ such that $\Pi' \models_\mathcal{D} \Pi\sigma$, $\Pi' \models_\mathcal{D} e' \sqsupseteq e\sigma$, $\Pi' \models_\mathcal{D} t' \sqsubseteq t\sigma$.
   (b) $\varphi = p\,\overline{e}_n \to!\, t \Leftarrow \Pi$, $\varphi' = p\,\overline{e'}_n \to!\, t' \Leftarrow \Pi'$, and there is some $\sigma \in Sub_\perp(\mathcal{U})$ such that $\Pi' \models_\mathcal{D} \Pi\sigma$, $\Pi' \models_\mathcal{D} p\,\overline{e'}_n \sqsupseteq (p\,\overline{e}_n)\sigma$, $\Pi' \models_\mathcal{D} t' \sqsupseteq t\sigma$.

The intuitive idea behind $\mathcal{D}$-entailment is that, whenever $\varphi \succcurlyeq_\mathcal{D} \varphi'$, 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 $\mathcal{D}$-entailment relation does rely only on assumptions concerning the monotonic behaviour of both primitive and defined functions, as well as on the radical behaviour of primitive functions.

The next definition generalizes the idea of π-interpretation [33,34] to our $CFLP(\mathcal{D})$ setting:

Definition 3.4 For any given constraint domain $\mathcal{D}$:

(i) A c-interpretation over $\mathcal{D}$ is any set $\mathcal{I}$ of c-facts including all the trivial c-facts and closed under $\mathcal{D}$-entailment. Equivalently, a c-interpretation is any set $\mathcal{I}$ of c-facts such that $cl_\mathcal{D}(\mathcal{I}) \subseteq \mathcal{I}$, where $cl_\mathcal{D}(\mathcal{I})$ is defined as follows:

$$cl_\mathcal{D}(\mathcal{I}) = \{\varphi' \mid \varphi' \text{ is a trivial c-fact, or else } \exists \varphi \in \mathcal{I}\,(\varphi \succcurlyeq_\mathcal{D} \varphi')\}$$

(ii) The $\mathcal{D}$-*grounding* of a c-interpretation $\mathcal{I}$ is defined as

$$gd_\mathcal{D}(\mathcal{I}) = \{\varphi \in \mathcal{I} \mid \varphi \text{ is a ground c-fact}\}$$

The grounding of a c-interpretation $\mathcal{I}$ is technically not a c-interpretation,

since it neither includes all the trivial c-facts, nor is closed under $\mathcal{D}$-entailment. Nevertheless, it is clear that $gd_{\mathcal{D}}(\mathcal{I})$ can be viewed as a description of a $\mathcal{D}$-algebra in the sense of Definition 3.2. Obviously, different c-interpretations can have the same grounding.

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

Definition 3.5 Semantic Calculus.
We write $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ to indicate that the c-statement φ can be derived from $\mathcal{I}$ 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 $t \in \mathcal{U} \cup \mathcal{V}$.

SP Simple Production:

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

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

DC Decomposition:

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

If $h\,\overline{e}_m$ is passive.

IR Inner Reduction:

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

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

DF$_{\mathcal{I}}$ $\mathcal{I}$-Defined Function:

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

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

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

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

PF Primitive Function:

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

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

AC Atomic Constraint:

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

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

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:

(i) $T = \mathbf{RL}(\varphi, [T_1 \cdots, T_p])$ represents a proof tree whose root φ is inferred with the inference rule **RL** from p previously derived c-statements with proof trees T_i $(1 \leq i \leq p)$.

(ii) $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ indicates that $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ is witnessed by the proof tree T.

(iii) T is called an easy proof tree iff T makes no use of the inference rules $\mathbf{DF}_{\mathcal{I}}$, **PF** and **AC**.

(iv) $\|T\|$ denotes the full size of the proof tree T, defined as the total number of nodes in T.

(v) $\mid T \mid$ denotes the restricted size of the proof tree T, defined as the number of nodes in T which are inferred with some of the rules $\mathbf{DF}_{\mathcal{I}}$, **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 Section 7.1.

Lemma 3.6 *Properties of the Semantic Calculus.*

(i) *Compactness Property:* $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *implies* $cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$ *for some finite subset* $\mathcal{I}_0 \subseteq \mathcal{I}$.

(ii) *Extension Property:* $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *and* $\mathcal{I} \subseteq \mathcal{I}'$ *implies* $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.

(iii) *Approximation Property: For any* $e \in Exp_{\perp}(\mathcal{U})$, $t \in Pat_{\perp}(\mathcal{U})$: $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ *iff there is some easy proof tree* T *such that* $T : \Vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$ *(derivation from the trivial c-interpretation* $\perp\!\!\!\perp = cl_{\mathcal{D}}(\emptyset)$*).*

(iv) *Conservation Property: For any c-fact* φ, $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *iff* $\varphi \in \mathcal{I}$.

(v) *Primitive c-atoms: For any primitive atom* $p\,\overline{t}_n \rightarrow\!!\, t$, $\mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{t}_n \rightarrow\!!\, t \Leftarrow \Pi$ *iff* $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \rightarrow\!!\, t$.

(vi) *Entailment Property:* $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *and* $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ *implies* $T' : \mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ *with proof tree* T' *such that* $\mid T' \mid \leq \mid T \mid$.

Using the semantic calculus, solutions of user defined constraints can be easily defined. The next definition generalizes Definition 2.5, assuming a given c-interpretation $\mathcal{I}$ over a constraint domain $\mathcal{D}$ with urelements $\mathcal{U}$:

Definition 3.7 Solutions of User Defined Constraints.

(i) The set of solutions of $\delta \in DCon_{\perp}(\mathcal{D})$ is a subset $Sol_{\mathcal{I}}(\delta) \subseteq Val_{\perp}(\mathcal{D})$ recursively defined as follows:
(a) $Sol_{\mathcal{I}}(\Box) = Val_{\perp}(\mathcal{D})$.

(b) $Sol_{\mathcal{I}}(\blacksquare) = \emptyset$.
(c) $Sol_{\mathcal{I}}(\delta) = \{\eta \in Val_{\perp}(\mathcal{D}) \mid \mathcal{I} \Vdash_{\mathcal{D}} \delta\eta\}$, for any atomic constraint $\delta \in DCon_{\perp}(\mathcal{D}) \setminus \{\square, \blacksquare\}$.
(d) $Sol_{\mathcal{I}}(\delta_1 \wedge \delta_2) = Sol_{\mathcal{I}}(\delta_1) \cap Sol_{\mathcal{I}}(\delta_2)$.
(e) $Sol_{\mathcal{I}}(\exists X \delta) = \{\eta \in Val_{\perp}(\mathcal{D}) \mid \eta' \in Sol_{\mathcal{I}}(\delta) \text{ for some } \eta' =_{\setminus\{X\}} \eta\}$

(ii) The set of solutions of a set of constraints $\Delta \subseteq DCon_{\perp}(\mathcal{D})$ is defined as $Sol_{\mathcal{I}}(\Delta) = \bigcap_{\delta \in \Delta} Sol_{\mathcal{I}}(\delta)$, corresponding to a logical reading of Δ as the conjunction of its members. In particular, $Sol_{\mathcal{I}}(\emptyset) = Val_{\perp}(\mathcal{D})$, corresponding to the logical reading of an empty conjunction as the identically true constraint $\square$.

For primitive constraints one can easily check that $Sol_{\mathcal{I}}(\pi) = Sol_{\mathcal{D}}(\pi)$ and $Sol_{\mathcal{I}}(\Pi) = Sol_{\mathcal{D}}(\Pi)$, using the obvious correspondence between Definitions 3.7 and 2.5.

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

Definition 3.8 Denotation of Expressions.
Assume a given c-interpretation $\mathcal{I}$ over a constraint domain $\mathcal{D}$. The denotation of any expression $e \in Exp_{\perp}(\mathcal{U})$ in $\mathcal{I}$ under a valuation $\eta \in Val_{\perp}(\mathcal{D})$ is defined as the set $[\![e]\!]^{\mathcal{I}}_{\eta} = \{t \in D_{\mathcal{U}} \mid \mathcal{I} \Vdash_{\mathcal{D}} e\eta \rightarrow t\}$. For the case of a ground expression $e \in GExp_{\perp}(\mathcal{U})$ we will abbreviate $[\![e]\!]^{\mathcal{I}}_{\varepsilon}$ as $[\![e]\!]^{\mathcal{I}}$.

Using Lemma 3.6, it is easy to prove that $[\![e]\!]^{\mathcal{I}}_{\eta} \subseteq D_{\mathcal{U}}$ includes the undefined element $\perp$ and is downwards closed w.r.t. the information ordering $\sqsubseteq$; i.e., $t' \in [\![e]\!]^{\mathcal{I}}_{\eta}$ holds whenever $t \in [\![e]\!]^{\mathcal{I}}_{\eta}$ for some $t \sqsupseteq t'$. Due to these properties, $[\![e]\!]^{\mathcal{I}}_{\eta}$ turns out to be an element of Hoare's Powerdomain $\mathcal{HP}(\overline{D_{\mathcal{U}}})$ [83,93,39], 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 [44,45,46] and shown to be convenient for programming on previous work on the $CRWL$ logic; see [36,79].

Definitions 3.7 and 3.8 just rely on the ground facts provided by the grounding of c-interpretations. On the contrary, the first item in the next definition really exploits the non-ground information provided by c-interpretations.

Definition 3.9 Strong and Weak Models.
For any given $CFLP(\mathcal{D})$-program $\mathcal{P}$ and c-interpretation $\mathcal{I}$ we say

(i) $\mathcal{I}$ is a strong model of $\mathcal{P}$ (in symbols $\mathcal{I} \models^{s}_{\mathcal{D}} \mathcal{P}$) iff
for any $(f\,\overline{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$, $\theta \in Sub_{\perp}(\mathcal{U})$, $\Pi \subseteq PCon_{\perp}(\mathcal{D})$ and $t \in Pat_{\perp}(\mathcal{U})$ such that $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi$ and $\mathcal{I} \Vdash_{\mathcal{D}} r\theta \rightarrow t \Leftarrow \Pi$ one has $((f\,\overline{t}_n)\theta \rightarrow t \Leftarrow \Pi) \in \mathcal{I}$.

(ii) $\mathcal{I}$ is a weak model of $\mathcal{P}$ (in symbols $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$) iff

for any $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$, $\eta \in GSub_\perp(\mathcal{U})$ and $t \in GPat_\perp(\mathcal{U})$ such that $\mathcal{I} \Vdash_\mathcal{D} (P \,\square\, \Delta)\eta$ and $\mathcal{I} \Vdash_\mathcal{D} r\eta \to t$ one has $((f\,\overline{t}_n)\eta \to t) \in \mathcal{I}$.

Roughly speaking, the weak model semantics $\mathcal{I} \models^w_\mathcal{D} \mathcal{P}$ means that all the individual instances of program rules from $\mathcal{P}$ must be valid in $\mathcal{I}$. Therefore, a technical variant of weak semantics could be also defined using the $\mathcal{D}$-algebras from Definition 3.2. On the other hand, the strong model semantics would not make sense for $\mathcal{D}$-algebras. The rough meaning of the strong model relationship $\mathcal{I} \models^s_\mathcal{D} \mathcal{P}$ is that all those c-facts that are "immediate consequences" from c-facts belonging to $\mathcal{I}$ via program rules from $\mathcal{P}$ must belong to $\mathcal{I}$. In comparison with previous works, the weak model semantics is similar to the model notion used for CRWL in [36,38,10], to the semantics in our older $CFLP(\mathcal{D})$ scheme [56,57], and to the more traditional $CLP(\mathcal{D})$ semantics in [47,48,49]; while the strong model semantics is analogous to the $\mathcal{S}_2$-semantics for $CLP(\mathcal{D})$-programs proposed in [33,34].

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

Proposition 3.10 *Strong versus Weak Models.*
For any $CFLP(\mathcal{D})$-program $\mathcal{P}$ and any c-interpretation $\mathcal{I}$ one has: $\mathcal{I} \models^s_\mathcal{D} \mathcal{P} \Rightarrow \mathcal{I} \models^w_\mathcal{D} \mathcal{P}$. The reciprocal is false in general.

Proof. Any strong model of a given $CFLP(\mathcal{D})$-program $\mathcal{P}$ is also a weak model of $\mathcal{P}$, because item (ii) in Definition 3.9 is the particular case of item (i) obtained when θ is a ground substitution, Π is empty and t is a ground pattern. As a counterexample for the reciprocal, consider the $CFLP(\mathcal{R})$-program $\mathcal{P}$ consisting of one single program rule $notZero\,X \to true \Leftarrow X /=_\mathbb{R} 0$ and the following c-interpretation over $\mathcal{R}$:

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

For this particular program and c-interpretation we can claim:

(i) $\mathcal{I} \models^w_\mathcal{R} \mathcal{P}$, because item (ii) in Definition 3.9 holds. Indeed, for any $\eta \in GSub_\perp(\mathbb{R})$, $\mathcal{I} \Vdash_\mathcal{R} (X /=_\mathbb{R} 0)\eta$ iff $\eta(X) \in \mathbb{R} \setminus \{0\}$. Therefore, for such η one also has $((notZero\,X)\eta \to true) \in \mathcal{I}$, since $\mathcal{I}$ is closed under $\mathcal{R}$-entailment.

(ii) $\mathcal{I} \not\models^s_\mathcal{R} \mathcal{P}$, because item (i) in Definition 3.9 fails when choosing ε as θ and $X /=_\mathbb{R} 0$ as Π. Indeed, $\mathcal{I} \Vdash_\mathcal{R} X /=_\mathbb{R} 0 \Leftarrow X /=_\mathbb{R} 0$, $\mathcal{I} \Vdash_\mathcal{R} true \to true \Leftarrow X /=_\mathbb{R} 0$, and $(notZero\,X \to true \Leftarrow X /=_\mathbb{R} 0) \notin \mathcal{I}$, since this c-fact does not follow by $\mathcal{R}$-entailment from the c-facts used to define $\mathcal{I}$.

□

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

Definition 3.11 Strong and Weak Consequence.
For any given $CFLP(\mathcal{D})$-program $\mathcal{P}$ and c-statement φ we say

(i) φ is a strong consequence of $\mathcal{P}$ (in symbols $\mathcal{P} \models^s_{\mathcal{D}} \varphi$) iff $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ holds for every strong model $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$.

(ii) φ is a weak consequence of $\mathcal{P}$ (in symbols $\mathcal{P} \models^w_{\mathcal{D}} \varphi$) iff $\mathcal{I} \Vdash_{\mathcal{D}} \varphi\eta$ holds for every weak model $\mathcal{I} \models^w_{\mathcal{D}} \mathcal{P}$ and every valuation $\eta \in Val_\perp(\mathcal{D})$.

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

3.3 A Fixpoint Characterization of Least Models

In this subsection we prove the existence of least models for $CFLP(\mathcal{D})$-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 [3,55] and constraint logic programming [49,33,34], as well as in our older $CFLP(\mathcal{D})$ scheme [56,57]. In our current $CFLP(\mathcal{D})$ scheme, the lattice structure is revealed by the following result:

Proposition 3.12 *Interpretation Lattice.*
$\mathbb{I}_{\mathcal{D}}$, defined as the set of all possible c-interpretations $\mathcal{I}$ over the constraint domain $\mathcal{D}$, is a complete lattice w.r.t. the set inclusion ordering. Moreover, the bottom element $\perp\!\!\!\perp$ and the top element $\top\!\!\!\top$ of this lattice can be characterized as follows:

$$\begin{aligned} \perp\!\!\!\perp &= cl_{\mathcal{D}}(\{\varphi \mid \varphi \text{ is a trivial c-fact}\}) \\ \top\!\!\!\top &= \{\varphi \mid \varphi \text{ is any c-fact}\} \end{aligned}$$

Proof. $\top\!\!\!\top$ is trivially the top element of $\mathbb{I}_{\mathcal{D}}$ w.r.t. to the set inclusion ordering. Moreover, $\perp\!\!\!\perp$ is the bottom element because any c-interpretation is required to include all the trivial c-facts and to be closed under $cl_{\mathcal{D}}$. It only remains to show that any subset $\mathfrak{I} \subseteq \mathbb{I}_{\mathcal{D}}$ 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:

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

□

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

(i) $ST_{\mathcal{P}}(\mathcal{I}) =_{def} cl_{\mathcal{D}}(\{(f\,\overline{t}_n)\theta \rightarrow t \Leftarrow \Pi \mid (f\,\overline{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}, \theta \in Sub_{\perp}(\mathcal{U}), \Pi \subseteq PCon_{\perp}(\mathcal{D}), t \in Pat_{\perp}(\mathcal{U}), \mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi, \mathcal{I} \Vdash_{\mathcal{D}} r\theta \rightarrow t \Leftarrow \Pi\})$

(ii) $WT_{\mathcal{P}}(\mathcal{I}) =_{def} cl_{\mathcal{D}}(\{(f\,\overline{t}_n)\eta \rightarrow t \mid (f\,\overline{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}, \eta \in GSub_{\perp}(\mathcal{U}), t \in GPat_{\perp}(\mathcal{U}), \mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta, \mathcal{I} \Vdash_{\mathcal{D}} r\eta \rightarrow t\})$

The crucial properties of the interpretation transformers are given in the next proposition, whose proof can be found in Section 7.1:

Proposition 3.14 *Properties of the Interpretation Transformers.*
For any fixed $CFLP(\mathcal{D})$-program $\mathcal{P}$, the transformers $ST_{\mathcal{P}}, WT_{\mathcal{P}} : \mathbb{I}_{\mathcal{D}} \rightarrow \mathbb{I}_{\mathcal{D}}$ are well defined continuous mappings, whose pre-fixpoints are the strong resp. weak models of $\mathcal{P}$. More precisely, for any $\mathcal{I} \in \mathbb{I}_{\mathcal{D}}$ one has $ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ iff $\mathcal{I} \models^{s}_{\mathcal{D}} \mathcal{P}$, and $WT_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ iff $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$.

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

Theorem 3.15 *Least Program Models.*
For every $CFLP(\mathcal{D})$-program $\mathcal{P}$ there exist:

(i) *A least strong model $\mathcal{S}_{\mathcal{P}} = lfp(ST_{\mathcal{P}}) = \bigcup_{k\in\mathbb{N}} ST_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$.*

(ii) *A least weak model $\mathcal{W}_{\mathcal{P}} = lfp(WT_{\mathcal{P}}) = \bigcup_{k\in\mathbb{N}} WT_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$.*

Proof. Due to a well known theorem by Knaster and Tarski [86], 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 3.14 trivially proves the theorem.□

In Section 4.2 we will see that $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$, the inclusion being strict in general. A deeper investigation of the relationship between both least models is left for future work.

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

In this section we generalize the $CRWL$ approach [36,38,10,79] to a new rewriting logic $CRWL(\mathcal{D})$, parameterized by a constraint domain $\mathcal{D}$, and aimed as a logical framework for $CFLP(\mathcal{D})$ programming. We start by presenting a logical calculus for $CRWL(\mathcal{D})$ 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 3.2 and 3.3. The relevance of $CRWL(\mathcal{D})$ w.r.t. past work and planned future work will be briefly discussed in the concluding section 5.

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

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

Definition 4.1 Constrained Rewriting Calculus.
We write $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ to indicate that the c-statement φ can be derived from $\mathcal{P}$ in the constrained rewriting calculus $CRWL(\mathcal{D})$, which consists of the inference rules **TI**, **RR**, **SP**, **DC**, **IR**, **PF** and **AC** already presented in the semantic calculus from Definition 3.5, plus the following inference rule:

DF$_{\mathcal{P}}$ $\mathcal{P}$-Defined Function:

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

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

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

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

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

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 $\mathbf{DF}_{\mathcal{P}}$ 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.

$CRWL(\mathcal{D})$-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 $CRWL(\mathcal{D})$-inference rule. Concerning proof trees and their sizes, we will use the same notation and terminology already introduced for the semantic calculus in subsection 3.2, modulo the replacement of rule $\mathbf{DF}_{\mathcal{I}}$ by rule $\mathbf{DF}_{\mathcal{P}}$. In particular, $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$ will indicate that $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ is witnessed by the proof tree T.

Most of the properties proved in Lemma 3.6 for the semantic calculus translate into analogous valid properties of the rewriting calculus $CRWL(\mathcal{D})$, with the only exception of item (iv) in Lemma 3.6, which seems to have no natural analogous in $CRWL(\mathcal{D})$. The properties are stated in the next lemma. Again, the rather technical proof can be found in Section 7.1.

Lemma 4.2 *Properties of the Constrained Rewriting Calculus.*

(i) *Compactness Property:* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *implies* $\mathcal{P}_0 \vdash_{\mathcal{D}} \varphi$ *for some finite subset* $\mathcal{P}_0 \subseteq \mathcal{P}$.

(ii) *Extension Property:* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *and* $\mathcal{P} \subseteq \mathcal{P}'$ *implies* $\mathcal{P}' \vdash_{\mathcal{D}} \varphi$.

(iii) *Approximation Property: For any* $e \in Exp_{\perp}(\mathcal{U})$, $t \in Pat_{\perp}(\mathcal{U})$: $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ *iff there is some easy proof tree* T *such that* $T : \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *(derivation from empty program).*

(iv) *Primitive c-atoms: For any primitive atom* $p\,\overline{t}_n \to !\, t$, $\mathcal{P} \vdash_{\mathcal{D}} p\,\overline{t}_n \to !\, t \Leftarrow \Pi$ *iff* $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to !\, t$.

(v) *Entailment Property:* $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$ *and* $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ *implies* $T' : \mathcal{P} \vdash_{\mathcal{D}} \varphi'$ *for some proof tree* T' *such that* $\mid T' \mid \leq \mid T \mid$.

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

In this section we investigate the relationship between $CRWL(\mathcal{D})$-derivability and the two model-theoretic semantics presented in sections 3.2 and 3.3. Our main result is the next theorem, showing a nice correspondence between $CRWL(\mathcal{D})$-derivability, strong consequence, and validity in least strong models:

Theorem 4.3 *Correctness Results for Strong Semantics.*
For any $CFLP(\mathcal{D})$*-program* $\mathcal{P}$ *and any c-statement* φ*, the following three con-*

ditions are equivalent:

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

Moreover, we also have:

(i) *Soundness: for any c-statement* φ, $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^{s}_{\mathcal{D}} \varphi$.

(ii) *Completeness: for any c-statement* φ, $\mathcal{P} \models^{s}_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$.

(iii) *Canonicity:* $\mathcal{S}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a c-fact and } \mathcal{P} \vdash_{\mathcal{D}} \varphi\}$.

Proof. A proof of the equivalence among (a), (b), (c) is given in Section 7.2. Soundness and completeness are just a trivial consequence of this equivalence. In order to prove canonicity, consider any c-fact φ. We know that $\varphi \in \mathcal{S}_{\mathcal{P}}$ iff $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$, because of the Conservation Property from Lemma 3.6. On the other hand, $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$ iff $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ is ensured by the equivalence between (c) and (a). □

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

Theorem 4.4 *Correctness Results for Weak Semantics.*
For any $CFLP(\mathcal{D})$-program $\mathcal{P}$ and any ground c-statement φ, the following three conditions are equivalent:

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

Moreover, we also have:

(i) *Soundness: for any c-statement* φ, $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^{w}_{\mathcal{D}} \varphi$.

(ii) *Ground Completeness: for any ground c-statement* φ, $\mathcal{P} \models^{w}_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$. *This does not hold in general for arbitrary c-statements.*

(iii) *Ground Canonicity:* $gd_{\mathcal{D}}(\mathcal{W}_{\mathcal{P}}) = \{\varphi \mid \varphi \text{ is a ground c-fact and } \mathcal{P} \vdash_{\mathcal{D}} \varphi\}$.

Proof. The equivalence among (a), (b), (c) can be proved by similar reasonings as those used in the proof of Theorem 4.3.

In order to prove soundness, assume any c-statement φ such that $\mathcal{P} \vdash_{\mathcal{D}} \varphi$. By the Entailment Property of Lemma 4.2, we the have $\mathcal{P} \vdash_{\mathcal{D}} \varphi\eta$ for all ground substitutions $\eta \in GSub_{\perp}(\mathcal{U})$. Due to the equivalence between (a) and (b), we get $\mathcal{P} \models^{w}_{\mathcal{D}} \varphi\eta$ for all $\eta \in GSub_{\perp}(\mathcal{U})$, which amounts to $\mathcal{I} \Vdash_{\mathcal{D}} \varphi\eta$ for every $\eta \in GSub_{\perp}(\mathcal{U})$ and all weak models $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$. Therefore, we can conclude that $\mathcal{P} \models^{w}_{\mathcal{D}} \varphi$.

Ground completeness is a direct consequence of the equivalence between (a) and (b). When considering arbitrary statements, completeness w.r.t. weak semantics fails in general. As a counterexample, consider the following $CFLP(\mathcal{R})$-program:

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

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

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

Finally, ground canonicity just follows form the equivalence between (c) and (a) and the Conservation Property from Lemma 3.6, as in the proof of Theorem 4.4. Note that $\mathcal{W}_{\mathcal{P}}$ includes also some non-ground c-facts, because all c-interpretations over $\mathcal{D}$ are required to be closed under $cl_{\mathcal{D}}$. Nevertheless, for the purposes of weak semantics, only the ground c-facts are the relevant.□

Using Theorems 4.3 and 4.4 we can now easily obtain two results that were announced at the end of sections 3.2 and 3.3, respectively.

Proposition 4.5 *Strong versus Weak Consequence.*
For any $CFLP(\mathcal{D})$-program $\mathcal{P}$ and any c-fact φ one has: $\mathcal{P} \models^s_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^w_{\mathcal{D}} \varphi$. The reciprocal is false in general.

Proof. Assume that $\mathcal{P} \models^s_{\mathcal{D}} \varphi$. By the Completeness Property in Theorem 4.3, we can conclude that $\mathcal{P} \vdash_{\mathcal{D}} \varphi$, which implies $\mathcal{P} \models^w_{\mathcal{D}} \varphi$ by the Soundness Property in Theorem 4.4.

On the other hand, in the proof of Theorem 4.4 we have seen a $CFLP(\mathcal{R})$-program $\mathcal{P}$ and a non-ground c-statement φ such that $\mathcal{P} \models^w_{\mathcal{R}} \varphi$ and $\mathcal{P} \not\vdash_{\mathcal{R}} \varphi$, which is the same as $\mathcal{P} \not\models^s_{\mathcal{R}} \varphi$ because of Theorem 4.3. □

Proposition 4.6 *Strong versus Weak Least Models.*
For any $CFLP(\mathcal{D})$-program $\mathcal{P}$ one has $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$. The inclusion is strict in general.

Proof. According to Proposition 3.10 and the first item in Theorem 3.15, $\mathcal{S}_{\mathcal{P}}$ is a weak model of $\mathcal{P}$. By the second item of Theorem 3.15, $\mathcal{W}_{\mathcal{P}}$ is the least weak model of $\mathcal{P}$. Therefore, $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$.

As a counterexample for the opposite inclusion, consider an arbitrary cons-

traint domain $\mathcal{D}$ with urelements $\mathcal{U}$, the $CFLP(\mathcal{D})$-program $\mathcal{P}$ consisting of one single program rule $id\ X \rightarrow X$ defining the identity function, and the c-interpretation $\mathcal{I} = cl_{\mathcal{D}}(\{id\ t \rightarrow t \mid t \in D_{\mathcal{U}}\})$. Note that the c-fact $\varphi = (id\ X \rightarrow X)$ does not belong to $\mathcal{I}$, since it is neither a trivial c-fact nor follows by $\mathcal{D}$-entailment from the ground c-facts used for defining $\mathcal{I}$. On the other hand, φ belongs to $\mathcal{S}_{\mathcal{P}}$ by the Canonicity Property in Theorem 4.3, because $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ is obviously true. Therefore, $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{I}$. But $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{I}$ holds by Theorem 3.15, because $\mathcal{I}$ is clearly a weak model of $\mathcal{P}$. From $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{I}$ and $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{I}$ we conclude $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{W}_{\mathcal{P}}$. □

5 Conclusions

We have proposed a new generic scheme $CFLP(\mathcal{D})$ 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 $CRWL(\mathcal{D})$ 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 and not trivial extension of known results on the semantics of success in the $CLP(\mathcal{D})$ scheme for constraint logic programming [49,34]. In comparison to previous work on constraint functional logic programming, we have improved our older $CFLP(\mathcal{D})$ scheme [56,57] 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 [53,5,58] and a solid starting point for a better foundation of our previous work on functional logic programming with multiset constraints [6,7]. Multiset constraints are outside the scope of the present paper because they use algebraic data constructors, while the $CFLP(\mathcal{D})$ scheme presented here assumes free data constructors.

The new scheme $CFLP(\mathcal{D})$ is also planned as a basis for several lines of ongoing and future work, involving other people at our University Department in addition to the authors. The design of a lazy constrained narrowing calculus for goal solving has already started, using ideas and techniques from the narrowing calculi in [36,38,88] as well as a notion of solver inspired in [56,5,63]. After completing this work, we plan to investigate an extension of the $CFLP(\mathcal{D})$ scheme with algebraic data constructors.

Concerning concrete instances of the $CFLP(\mathcal{D})$ scheme, we plan to formali-

ze the work on functional logic programming with finite domain constraints started in [32] and to investigate practical constraint solving methods and applications of the resulting language.

Last but not least, we also plan to extend the work on declarative debugging of functional logic programs started in [15,16,17] to $CFLP(\mathcal{D})$-programs, taking into account existing work on the declarative debugging of constraint logic programs [87] and the finite failure semantics of functional logic programs with disequality constraints [60,61,62,63,64].

References

[1] M. Abengózar-Carneros, P. Arenas-Sánchez, R. Caballer-Roldán, A. Gil-Luezas, J.C. González-Moreno, J. Leach-Albert, F.J. López-Fraguas, N. Martí-Oliet, J.M. Molina-Bravo, E. Pimentel-Sánchez, M. Rodríguez-Artalejo, M.M. Roldán-García, J.J. Ruz-Ortiz and J. Sánchez-Hernández. *TOY: A Multiparadigm Declarative Language. Version 2.0.* Technical Report, Dpto. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, February 2002.

[2] H. Aït-Kaci and A. Podelski. *A feature constraint system for logic programming with entailment.* Theoretical Computer Science 122, pp. 263–283, 1994.

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

[4] K.R. Apt and M. Gabbrielli. *Declarative Interpretations Reconsidered.* Proc. Int. Conf. on Logic Programming (ICLP'94), Santa Margherita Ligure, the MIT Press, pp. 74–89, 1994.

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

[6] P. Arenas-Sánchez, F.J. López-Fraguas and M. Rodríguez-Artalejo. *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.

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

[8] P. Arenas-Sánchez and M. Rodríguez-Artalejo. *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.

[9] P. Arenas-Sánchez and M. Rodríguez-Artalejo. *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.

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

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

[12] R. Backofen. *A Complete Axiomatization of a Theory with Feature and Arity Constraints.* Journal of Logic Programming 24(1&2), pp. 37–71, 1995.

[13] R. Backofen and G. Smolka. *A complete and recursive feature theory*. Theoretical Computer Science 146, pp. 243–268, 1995.

[14] A. Bossi, M. Gabbrielli, G. Levi and M. Martelli. *The s-Semantics Approach: Theory and Applications*. Journal of Logic Programming 19&20, pp. 149–197, 1994.

[15] R. Caballero, F.J. López-Fraguas and M. Rodríguez-Artalejo. *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.

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

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

[18] K.L. Clark. *Predicate logic as a computational formalism*. Research Report DOC 79/59, Imperial College, Department of Computing, London 1979.

[19] A. Colmerauer. *Prolog and Infinite Trees*. In K.L. Clark and S.A. Tärnlud (eds.), *Logic Programming*, Academic Press, pp. 153–172, 1982.

[20] A. Colmerauer. *Equations and Inequations on Finite and Infinite Trees*. Proc. of the 2nd International Conference on Fifth Generation Computer Systems, pp. 85–89, 1984.

[21] L. Damas and R. Milner. *Principal Type Schemes for Functional Programs*. Proc. ACM Symp. on Principles of Programming Languages (POPL'82), ACM Press, pp. 207–212, 1982.

[22] J. Darlington and Y.K. Guo. *Constraint Functional Programming*. Technical Report, Imperial College, November 1989.

[23] J. Darlington and Y.K. Guo. *Constraint Equational Deduction*. Proc. of 2nd Int. Workshop on Conditional and Typed Rewriting Systems (CTRS'90), Springer LNCS 516, pp. 11–14, 1991.

[24] J. Darlington, Y.K. Guo and H. Pull. *Introducing Constraint Functional Logic Programming*. PHOENIX Seminar and Workshop on Declarative Programming (DP'91), Springer Workshops in Computing, pp. 20–34, 1992.

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

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

[27] N. Dershowitz and J.P. Jouannaud. *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.

[28] N. Dershowitz and M. Okada. *A Rationale for Conditional Equational Programming*. Theoretical Computer Science 75, pp. 111–138, 1990.

[29] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi. *Declarative Modeling of the Operational Behavior of Logic Languages*. Theoretical Computer Science 69(3). pp. 289–318, 1989.

[30] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi. *A Model-theoretic Reconstruction of the Operational Semantics of Logic Programs*. Information and Computation 102(1). pp. 86-113, 1993.

[31] M.J. Fay. *First-Order Unification in an Equational Theory*. Proc. Workshop on Automated Deduction (CADE'79), Academic Press, pp. 161–177, 1979.

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

[33] M. Gabbrielli and G. Levi. *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.

[34] M. Gabbrielli, G.M. Dore and G. Levi. *Observable Semantics for Constraint Logic Programs*. Journal of Logic and Computation 5 (2), pp. 133–171, 1995.

[35] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas and M. Rodríguez-Artalejo. *A Rewriting Logic for Declarative Programming*. Proc. European Symp. on Programming (ESOP'96), Springer LNCS 1058, pp. 156–172, 1996.

[36] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas and M. Rodríguez-Artalejo. *An Approach to Declarative Programming Based on a Rewriting Logic*. Journal of Logic Programming 40(1), pp. 47–87, 1999.

[37] J.C. González-Moreno, M.T. Hortalá-González and M. Rodríguez-Artalejo. *A Higher Order Rewriting Logic for Functional Logic Programming*. Proc. Int. Conf. on Logic Programming, The MIT Press, pp. 153–167, 1997.

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

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

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

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

[42] M. Henz, G. Smolka and J. Würtz. *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.

[43] J.M. Hullot. *Canonical Forms and Unification*. Proc. Conf. on Automated Deduction (CADE'80), Springer LNCS 87, pp. 318–334, 1980.

[44] H. Hussmann. *Nichtdeterministische Algebraische Spezifikationen*. Ph. D. Thesis, University of Passau, 1988. (In German)

[45] H. Hussmann. *Nondeterministic Algebraic Specifications and Nonconfluent Term Rewriting*. Journal of Logic Programming 12, pp. 237–255, 1992.

[46] H. Hussmann. *Non-determinism in Algebraic Specifications and Algebraic Programs*. Birkhäuser Verlag, 1993.

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

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

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

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

[51] C. Kirchner, H. Kirchner and M. Rusinowitch. *Deduction with Symbolic Constraints*. Revue Française d'Intelligence Artificielle, 4 (3) pp. 9–52, 1990.

[52] J.W. Klop. *Term Rewriting Systems*. In S. Abramsky, D.M. Gabbay and T.S.E. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. 2, pp. 2–116, Oxford University Press, 1992.

[53] H. Kuchen, F.J. López-Fraguas, J.J. Moreno-Navarro and M. Rodríguez-Artalejo. *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.

[54] D.S. Lankford. *Canonical inference*. Technical Report ATP-32, Department of Mathematics and Computer Science, University of Texas at Austin, 1975.

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

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

[57] F.J. López-Fraguas. *Programación Funcional y Lógica con Restricciones*. Ph.D. Thesis, Univ. Complutense Madrid, 1994. (In Spanish)

[58] F.J.López Fraguas, J. Sánchez Hernández. *Disequalities May Help to Narrow*. Proc. APPIA-GULP-PRODE'99, pp. 89–104, 1999.

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

[60] F.J. López-Fraguas and J. Sánchez-Hernández. *Proving Failure in Functional Logic Programs*. Proc. Int. Conf. on Computational Logic (CL'2000), Springer LNCS 1861, pp. 179–193, 2000.

[61] F.J. López-Fraguas and J. Sánchez-Hernández. *Functional Logic Programming with Failure: A Set-Oriented View*. Proc. Int. Conf. on Logic Programming and Automated Reasoning (LPAR'2001), Springer LNCS 2250, pp. 455–469, 2001.

[62] F.J. López-Fraguas and J. Sánchez-Hernández. *Narrowing Failure in Functional Logic Programming*. Proc. 6th Int. Symp. on Functional and Logic Programming (FLOPS'2002), Springer LNCS 2441, pp. 212–227, 2002.

[63] F.J. López-Fraguas and J. Sánchez-Hernández. *Failure and equality in functional logic programming*. Electronic Notes in Theoretical Computer Science 86(3), 21 pages, 2003.

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

[65] M.J. Maher. *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.

[66] L. Mandel. *Constrained lambda calculus*. Aachen Verlag Shaker, 1995.

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

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

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

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

[71] J. Meseguer. *Conditional Rewriting Logic as a Unified Model of Concurrency*. Theoretical Computer Science 96, pp. 73–155, 1992.

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

[73] R. Milner. *A Theory of Type Polymorphism in Programming*. Journal of Computer and Systems Sciences, 17, pp. 348–375, 1978.

[74] B. Möller. *On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types*. Acta Informatica 22, pp. 537–578, 1985.

[75] A. Mück, T. Streicher. *A Tiny Constrain Functional Logic Language and Its Continuation Semantics*. Proc. European Symp. on Programming (ESOP'94), Springer LNCS 788, pp. 439-453, 1994.

[76] M. Palomino Tarjuelo. *Comparing Meseguer's Rewriting Logic with the Logic CRWL*. Electronic Notes in Theoretical Computer Science 64, 22 pages, 2002.

[77] M. Palomino Tarjuelo. *A Comparison between Meseguer's Rewriting Logic and the Logic CRWL*. Under consideration for publication in Theory and Practice of Logic Programming.

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

[79] M. Rodríguez-Artalejo. *Functional and Constraint Logic Programming*. in H. Comon, C. Marché and R. Treinen (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.

[80] V. Saraswat. *Concurrent Constraint Programming Languages*. PhD Thesis, Carnegie Mellon University, 1989. In ACM distinguished dissertattion series. The MIT press, 1993.

[81] V. Saraswat and M. Rinard. *Concurrent Constraint Programming*. Proc. of the 17th Annual Symposium on Principles of Programming Languages (POPL'90), ACM Computer Society Press, pp. 232–245, 1990.

[82] V. Saraswat, M. Rinard and P. Panangaden. *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.

[83] D.S. Scott. *Domains for Denotational Semantics*. Proc. ICALP'82, Springer LNCS 140, pp. 577–613, 1982.

[84] J.R. Slagle. *Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity*. Journal of the ACM 21(4), pp. 622–642, 1974.

[85] G. Smolka and R. Treinen. *Records for Logic Programming*. Journal of Logic Programming 18, pp. 229–258, 1994.

[86] A. Tarski. *A lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics 5, pp. 285–309, 1955.

[87] A. Tessier and G. Ferrand. *Declarative Diagnosis in the CLP Scheme*. In P. Deransart, M. Hermenegildo and J. Małuszynski (eds.), *Analysis and Visualization Tools for Constraint Programming*, Chapter 5, pp. 151–174. Springer LNCS 1870, 2000.

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

[89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series, The MIT Press, 1989.

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

[91] P. Van Hentenryck, H. Simonis and M. Dincbas. *Constraint satisfaction using constraint logic programming*. Artificial Intelligence 58, pp. 113–159, 1994.

[92] P. Van Hentenryck, V. Saraswat and Y. Deville. *Design, implementation and evaluation of the constraint language* `cc(FD)`. Journal of Logic Programming 37, pp. 139–164, 1998.

[93] G. Winskel. *On Powerdomains and Modality*. Theoretical Computer Science 36, pp. 127–137, 1985.

6 Small sample of $CFLP(\mathcal{D})$-programming in $\mathcal{TOY}$

6.1 Permutation sort in $CFLP(\mathcal{H}_{seq})$ 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 -> check Test (Generate Input)

check :: (Solution -> bool) -> Solution -> Solution
check Test Candidate = Candidate <== Test Candidate

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

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

% The generator computes permutations:

permute :: [A] -> [A]
permute []     -> []
permute [X|Xs] -> insert X (permute Xs)

insert :: A -> [A] -> [A]
insert X []     -> [X]
insert X [Y|Ys] -> [X,Y|Ys] // [Y|insert X Ys]

% Binary choice function.

infixr 20 //

(//) :: A -> A -> A
X // Y = X
X // Y = Y

% 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

% Auxiliary funtion for tests:
```

```
downFrom :: int -> [int]
downFrom N = if N > 0 then [N|downFrom (N-1)] else [N]

% Goal:      permSort (downFrom 99) == Xs
% Solution:  Xs = [0,1,2, ..., 99]
```

6.2 *List difference in* $CFLP(\mathcal{H}_{seq})$ *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
```

6.3 Computing a mortgage in $CFLP(\mathcal{R})$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                            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)
```

7 Proofs of the main results

7.1 Proofs of the main results from section 3

Proof of Lemma 3.6

Proof. (1) Assume a given proof tree T for $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. Reasoning by induction on $\|T\|$ we prove the existence of some finite subset $\mathcal{I}_0 \subseteq \mathcal{I}$ and a proof tree T' such that $T' : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$. 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 $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ is a derivation from the trivial c-interpretation $cl_{\mathcal{D}}(\emptyset)$. Therefore, T and T' are the same easy proof tree for $cl_{\mathcal{D}}(\emptyset) \Vdash_{\mathcal{D}} \varphi$, and of course, for $cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$ with $\mathcal{I}_0 \subseteq \mathcal{I}$ every finite subset. In other case, using the induction hypothesis and the fact that we only use almost one c-fact of $\mathcal{I}$ 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 $T = \mathbf{DF}_{\mathcal{I}}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ for some c-fact $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ such that $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ with $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $\|T_s\| < \|T\|$, by induction hypothesis we obtain $T'_i : cl_{\mathcal{D}}(\mathcal{I}_i) \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ for some finite subset $\mathcal{I}_i \subseteq \mathcal{I}$ $(1 \leq i \leq n)$ and $T'_s : cl_{\mathcal{D}}(\mathcal{I}_s) \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ for some finite subset $\mathcal{I}_s \subseteq \mathcal{I}$. Then, we can define the finite subset $\mathcal{I}_0 =_{def} \bigcup_{i=1}^{m} \mathcal{I}_i \cup \mathcal{I}_s \cup \{f\,\overline{t}_n \to s \Leftarrow \Pi\}$. We note that $\mathcal{I}_0 \subseteq \mathcal{I}$ and $cl_{\mathcal{D}}(\mathcal{I}_0) = \bigcup_{i=1}^{m} cl_{\mathcal{D}}(\mathcal{I}_i) \cup cl_{\mathcal{D}}(\mathcal{I}_s) \cup cl_{\mathcal{D}}(\{f\,\overline{t}_n \to s \Leftarrow \Pi\})$. Moreover, we have $T'_i : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$, $T'_s : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ and $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in cl_{\mathcal{D}}(\mathcal{I}_0)$. Hence, $T' =_{def} \mathbf{DF}_{cl_{\mathcal{D}}(\mathcal{I}_0)}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T'_1, \ldots, T'_n, T'_s])$.
(2) Assume a given proof tree T for $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. Reasoning by induction on $\|T\|$, we prove the existence of a proof tree T' for $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$. First, if T is an easy proof tree then the property holds trivially because $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ is a derivation from the trivial c-interpretation $cl_{\mathcal{D}}(\emptyset)$. Therefore, T and T' are the same easy proof tree for $cl_{\mathcal{D}}(\emptyset) \Vdash_{\mathcal{D}} \varphi$, and of course, for $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$. In other case, using the induction hypothesis and the fact that $\mathcal{I} \subseteq \mathcal{I}'$ if $\mathcal{I}$ 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 $T = \mathbf{DF}_{\mathcal{I}}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ for some c-fact $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ such that $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ with $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $\|T_s\| < \|T\|$, by induction hypothesis we obtain $T'_i : \mathcal{I}' \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ and $T'_s : \mathcal{I}' \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$. Moreover, since $\mathcal{I} \subseteq \mathcal{I}'$, we also have $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}'$. Hence, $T' =_{def} \mathbf{DF}_{\mathcal{I}'}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T'_1, \ldots, T'_n, T'_s])$, which verifies $T' : \mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.

(3) In case that $Sol_{\mathcal{D}}(\Pi) = \emptyset$, $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ is trivially true and $T : \Vdash_{\mathcal{D}}$

$e \to t \Leftarrow \Pi$ with just one **TI** inference. In the rest of this proof we can assume $Sol_{\mathcal{D}}(\Pi) \neq \emptyset$ and reason by induction on the syntactic size of e. We distinguish cases for t:

- $t = \bot$. In this case, $\Pi \models_{\mathcal{D}} e \sqsupseteq \bot$ is trivially true and $T : \Vdash_{\mathcal{D}} e \to \bot \Leftarrow \Pi$ with just one **TI** inference.

- $t = u \in \mathcal{U}$. We consider several subcases for e. If $e = u$ then $\Pi \models_{\mathcal{D}} u \sqsupseteq u$ is true and $T : \Vdash_{\mathcal{D}} u \to u \Leftarrow \Pi$ with just one **RR** inference. If $e = X \in \mathcal{V}$ and $\Pi \models_{\mathcal{D}} X \sqsupseteq u$ then $T : \Vdash_{\mathcal{D}} X \to u \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathcal{D}} X \sqsupseteq u$ then $\not\Vdash_{\mathcal{D}} X \to u \Leftarrow \Pi$ (since no inference rule is applicable). Finally, if e is neither u nor a variable then $\Pi \not\models_{\mathcal{D}} e \sqsupseteq u$. Assume a proof tree $T : \Vdash_{\mathcal{D}} e \to u \Leftarrow \Pi$ (if there is no proof tree, then we are done). Since the c-interpretation is $cl_{\mathcal{D}}(\emptyset)$ and $e \neq u$, $e \notin \mathcal{V}$, the inference rule applied at the root of T must be either **PF** or **AC**. In either case, T is not easy.

- $t = X \in \mathcal{V}$. We consider several subcases for e. If $e = X$ then $\Pi \models_{\mathcal{D}} X \sqsupseteq X$ and $T : \Vdash_{\mathcal{D}} X \to X \Leftarrow \Pi$ with just one **RR** inference. If e is a pattern $s \neq X$ and $\Pi \models_{\mathcal{D}} s \sqsupseteq X$ then $T : \Vdash_{\mathcal{D}} s \to X \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathcal{D}} s \sqsupseteq X$ then $\not\Vdash_{\mathcal{D}} s \to X \Leftarrow \Pi$ (since no inference rule is applicable). Finally, if e is not a pattern, we consider any $\mu \in Sol_{\mathcal{D}}(\Pi)$ such that $X\mu$ is a total pattern. Then $e\mu \sqsupseteq X\mu$ is not true and therefore $\Pi \not\models_{\mathcal{D}} e \sqsupseteq X$. Assume a proof tree $T : \Vdash_{\mathcal{D}} e \to X \Leftarrow \Pi$ (if there is no proof tree, then we are done). Since the c-interpretation is $cl_{\mathcal{D}}(\emptyset)$ 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 $T = \mathbf{IR}(h\,\overline{e}_m \to X \Leftarrow \Pi, [T_1, \ldots, T_m])$, and for each $1 \leq i \leq m$, $T_i : \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ such that $\Pi \models_{\mathcal{D}} h\,\overline{t}_m \sqsupseteq X$. Then, for some $1 \leq i \leq m$, $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$. Otherwise we would have $\Pi \models_{\mathcal{D}} e_i \sqsupseteq t_i$ for all $1 \leq i \leq m$, and then $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ and $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq X$, which is not the case. Fix any $1 \leq i \leq m$ such that $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$. By induction hypothesis (note that the size of e_i is smaller than the size of $h\,\overline{e}_m$), 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 $e = X \in \mathcal{V}$ and $\Pi \models_{\mathcal{D}} X \sqsupseteq h\,\overline{t}_m$ then $T : \Vdash_{\mathcal{D}} X \to h\,\overline{t}_m \Leftarrow \Pi$ with just one **SP** inference, and if $\Pi \not\models_{\mathcal{D}} X \sqsupseteq h\,\overline{t}_m$ then $\not\Vdash_{\mathcal{D}} X \to h\,\overline{t}_m \Leftarrow \Pi$

(since no inference rule is applicable). If $e = h\,\overline{e_m}$ and $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ then $\Pi \models_{\mathcal{D}} e_i \sqsupseteq t_i$ for all $1 \leq i \leq m$. Hence, by induction hypothesis (the size of e_i is smaller than the size of $h\,\overline{e}_m$), $\Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ with an easy proof tree T_i for all $1 \leq i \leq m$ and $T : \Vdash_{\mathcal{D}} h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi$ with $T = \mathbf{DC}(h\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$ is an easy proof true. Moreover, if $\Pi \not\models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ then $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$ 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 $\Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ exists. Therefore, no easy proof tree T exists for $\Vdash_{\mathcal{D}} h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi$ (**DC** doesn't work and no other inference rule applies). Finally, if e is neither a variable nor of the form $h\,\overline{e}_m$ then we consider any total $\mu \in Sol_{\mathcal{D}}(\Pi)$. Clearly $e\mu \sqsupseteq (h\,\overline{t}_m)\mu$ is not true. Hence, $\Pi \not\models_{\mathcal{D}} e \sqsupseteq h\,\overline{t}_m$. If $\not\Vdash_{\mathcal{D}} e \to h\,\overline{t}_m \Leftarrow \Pi$ we are done. If there is some proof tree $T : \Vdash_{\mathcal{D}} e \to h\,\overline{t}_m \Leftarrow \Pi$, the inference rule applied at the root must be either **PF** or **AC** ($\mathbf{DF}_{\mathcal{I}}$ cannot be used with the trivial c-interpretation $cl_{\mathcal{D}}(\emptyset)$). In any case, T is not easy.

(4) Let φ be a c-fact of the form $f\,\overline{t}_n \to t \Leftarrow \Pi$. We prove first the "if" part. Taking into account that $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t_i \Leftarrow \Pi$ 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 $t_i \in Pat_{\bot}(\mathcal{U})$ and therefore $\Pi \models_{\mathcal{D}} t_i \sqsupseteq t_i$ also holds for all $1 \leq i \leq n$), we can also suppose that $t \neq \bot$ (the case $\mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{t}_n \to \bot \Leftarrow \Pi$ is trivial by **TI**) and build directly the deduction $\mathbf{DF}_{\mathcal{I}}(f\overline{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$ using the initial hypothesis $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$. The "only if" part. First, if we suppose that $t = \bot$ or $Sol_{\mathcal{D}}(\Pi) = \emptyset$, directly $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ by definition of c-interpretation. Otherwise, by initial hypothesis $T : \mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{t}_n \to t \Leftarrow \Pi$ must have the form $T = \mathbf{DF}_{\mathcal{I}}(f\overline{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$, where $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ are easy proof trees such that $(f\,\overline{t'}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ with $t'_1, \ldots, t'_n \in Pat_{\bot}(\mathcal{U})$. In this setting, we obtain $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq t_i$ for all $1 \leq i \leq n$ using the Approximation Property. It follows that $(f\,\overline{t'}_n \to t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (f\,\overline{t}_n \to t \Leftarrow \Pi)$ and consequently $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ because $\mathcal{I}$ is closed under entailment by definition of c-interpretation.
(5) The "only if" part. By initial hypothesis, a proof tree T for $\mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{t}_n \to !\,t \Leftarrow \Pi$ must have the form $T = \mathbf{AC}(p\,\overline{t}_n \to !\,t \Leftarrow \Pi, [T_1, \ldots, T_n])$, where $\Pi \models_{\mathcal{D}} p\,\overline{t'}_n \to !\,t$ for some $t'_1, \ldots, t'_n \in Pat_{\bot}(\mathcal{U})$ and $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ are easy proof trees. Moreover, $\Pi \models_{\mathcal{D}} t_i \sqsupseteq t'_i$ $(1 \leq i \leq n)$ follows using the Approximation Property, and then $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to !\,t$. Now, the "if" part. Since $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to !\,t$ by initial hypothesis and $T_i : \Vdash_{\mathcal{D}} t_i \to t_i \Leftarrow \Pi$ are easy proof trees for all $1 \leq i \leq n$ using the Approximation Property, we can build a proof tree $T =_{def} \mathbf{AC}(p\,\overline{t}_n \to !\,t \Leftarrow \Pi, [T_1, \ldots, T_n])$ for

$\mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{t}_n \rightarrow\!!\, t \Leftarrow \Pi$.

(6) Assume $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ and a substitution σ which relates φ and φ' as expected by the entailment relation (see definition 3.3). We can also assume $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$, otherwise $T' : \mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ 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 $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. Reasoning by induction on $\|T\|$ we prove the existence of a proof tree T' for $\mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ such that $\mid T \mid \geq \mid T' \mid$. We distinguish various possible cases:

- T is an easy proof tree and $\varphi = e \rightarrow t \Leftarrow \Pi$. 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, $\mathbf{DF}_{\mathcal{I}}$ is not used. Therefore, $T : \Vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$. By the Approximation Property, $\Pi \models_{\mathcal{D}} e \sqsupseteq t$. This implies $\Pi\sigma \models_{\mathcal{D}} e\sigma \sqsupseteq t\sigma$. Since $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, we know that $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq e\sigma$ and $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$. We can conclude $\Pi' \models_{\mathcal{D}} e' \sqsupseteq t'$. By the Approximation Property again, there is some easy $T' : \Vdash_{\mathcal{D}} e' \rightarrow t' \Leftarrow \Pi'$, and of course, $T' : \mathcal{I} \Vdash_{\mathcal{D}} e' \rightarrow t' \Leftarrow \Pi'$. Since T and T' are both easy, $\mid T \mid = 0 \geq 0 = \mid T' \mid$.

- $T = \mathbf{DC}(h\,\overline{e}_m \rightarrow h\,\overline{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$. In this case, we know $\varphi = h\,\overline{e}_m \rightarrow h\,\overline{t}_m \Leftarrow \Pi$ with $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq m)$ and $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (h\,\overline{e}_m)\sigma$, $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq t'$. We can assume that T is not easy; otherwise we could reason as in the previous case. Since T is not easy, $h\,\overline{e}_m$ is not a pattern. Then it must be the case that $e' = h\,\overline{e'_m}$ with $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq m$ (otherwise, any total $\mu \in Sol_{\mathcal{D}}(\Pi')$ would be such that $e'\mu \sqsupseteq (h\,\overline{e}_m)\sigma\mu$ is not true). Moreover, $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq t'$ and $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$ leave only two possible cases for t'. First, $t' = h\,\overline{t'_m}$ with $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t'_i$ for all $1 \leq i \leq m$. In this case, for each $1 \leq i \leq m$ we have $(e_i \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \rightarrow t'_i \Leftarrow \Pi')$ because $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$, $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t'_i$. By induction hypothesis, we can assume proof trees $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \rightarrow t'_i \Leftarrow \Pi'$ with $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq m)$. Therefore, $T' =_{def} \mathbf{DC}(h\,\overline{e'}_m \rightarrow h\,\overline{t'}_m \Leftarrow \Pi', [T'_1, \ldots, T'_m])$ verifies that $T' : \mathcal{I} \Vdash_{\mathcal{D}} h\,\overline{e'}_m \rightarrow h\,\overline{t'}_m \Leftarrow \Pi'$ and $\mid T \mid = \sum_{i=1}^{m} \mid T_i \mid \geq \sum_{i=1}^{m} \mid T'_i \mid = \mid T' \mid$. Second, if $t' = X \in \mathcal{V}$ with $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X$. In this case, for each $1 \leq i \leq m$ we have $(e_i \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \rightarrow t_i\sigma \Leftarrow \Pi')$ because $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$, $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t_i\sigma$. By induction hypothesis, we can assume proof trees $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \rightarrow t_i\sigma \Leftarrow \Pi'$ with $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq m)$. Since $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X$, we can build the proof tree $T' =_{def} \mathbf{IR}(h\,\overline{e'}_m \rightarrow X \Leftarrow \Pi', [T'_1, \ldots, T'_m])$, which verifies

$T' : \mathcal{I} \Vdash_{\mathcal{D}} h\,\overline{e'}_m \to X \Leftarrow \Pi'$ and $|\,T\,| = \sum_{i=1}^{m} |\,T_i\,| \geq \sum_{i=1}^{m} |\,T'_i\,| = |\,T'\,|$.

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

- $T = \mathbf{DF}_{\mathcal{I}}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$. Assume $k > 0$ (the case $k = 0$ is analogous and easier). In this case, we know $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ with $t \neq \bot$, and there are some c-fact $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ and some partial pattern $s \neq \bot$ such that $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ and $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$, $\|T_s\| < \|T\|$. Since $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, we know $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$ and $t' \neq \bot$ (if $t' = \bot$ then T' consists of just one **TI** step and $|\,T\,| > 0 = |\,T'\,|$). From $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, it follows that $e' = f\,\overline{e'}_n\overline{a'}_k$ with $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$ and $\Pi' \models_{\mathcal{D}} a'_j \sqsupseteq a_j\sigma$ for all $1 \leq j \leq k$ (otherwise, for any total $\mu \in Sol_{\mathcal{D}}(\Pi')$ 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 $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ for all $1 \leq i \leq n$ and $(s\,\overline{a}_k \to t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi')$. By induction hypothesis (applied to T_i, T_s), we get $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $|\,T_i\,| \geq |\,T'_i\,|$ $(1 \leq i \leq n)$ and $T'_s : \mathcal{I} \Vdash_{\mathcal{D}} s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi'$, $|\,T_s\,| \geq |\,T'_s\,|$. Since $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ and $(f\,\overline{t}_n \to s \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (f\,\overline{t_n\sigma} \to s\sigma \Leftarrow \Pi')$, it implies that $(f\,\overline{t_n\sigma} \to s\sigma \Leftarrow \Pi') \in \mathcal{I}$ 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 $\mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $k > 0$ and $t \neq \bot$). We can build the proof tree $T' =_{def} \mathbf{DF}_{\mathcal{I}}(f\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi', [T'_1, \ldots, T'_n, T'_s])$, which verifies $T' : \mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{e'}_n\overline{a'}_k \to t' \Leftarrow \Pi'$ and $|\,T\,| = 1 + \sum_{i=1}^{m} |\,T_i\,| + |\,T_s\,| \geq 1 + \sum_{i=1}^{m} |\,T'_i\,| + |\,T'_s\,| = |\,T'\,|$.

- $T = \mathbf{PF}(p\,\overline{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$. In this case, we know $\varphi = p\,\overline{e}_n \to t \Leftarrow \Pi$ and $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ with $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to t$. Since $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ and $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$, it must be the case that $\varphi' = p\,\overline{e'}_n \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$, $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$. From the previous conditions, we can deduce $\Pi\sigma \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t\sigma$ and hence also $\Pi' \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t'$. We also

observe that $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ $(1 \leq i \leq n)$. By induction hypothesis, we obtain proof trees $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $\mid T_i \mid \geq \mid T'_i \mid (1 \leq i \leq n)$. Since $\Pi' \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t'$, 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 $T' : \mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{e'}_n \to t' \Leftarrow \Pi'$ with $\mid T \mid = 1 + \sum_{i=1}^{m} \mid T_i \mid \geq 1 + \sum_{i=1}^{m} \mid T'_i \mid = \mid T' \mid$.

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

□

Proof of Proposition 3.14

Proof. We prove only the properties of the strong interpretation transformer $ST_{\mathcal{P}}$; the corresponding properties of $WT_{\mathcal{P}}$ can be proved similarly. In the rest of the proof we use the notation $preST_{\mathcal{P}}(\mathcal{I})$ for the set of c-facts.

$$\{(f\,\overline{t}_n)\theta \to t \Leftarrow \Pi \mid (f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}, \theta \in Sub_{\perp}(\mathcal{U}), \Pi \subseteq PCon_{\perp}(\mathcal{D}), t \in Pat_{\perp}(\mathcal{U}), \mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi, \mathcal{I} \Vdash_{\mathcal{D}} r\theta \to t \Leftarrow \Pi\}$$

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

A careful inspection of Definition 3.9 reveals that $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$ iff $preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. On the other hand, $preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ iff $ST_{\mathcal{P}}(\mathcal{I}) = cl_{\mathcal{D}}(preST_{\mathcal{P}}(\mathcal{I})) \subseteq \mathcal{I}$, because $\mathcal{I}$ is closed under $cl_{\mathcal{D}}$. Therefore, $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$ iff $ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, i.e., the strong models of $\mathcal{P}$ are the pre-fixpoints of $ST_{\mathcal{P}}$.

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

(i) $ST_{\mathcal{P}}$ is monotonic:

Assume $\mathcal{I}$, $\mathcal{J} \in \mathbb{I}_{\mathcal{D}}$ such that $\mathcal{I} \subseteq \mathcal{J}$. Then, $preST_{\mathcal{P}}(\mathcal{I}) \subseteq preST_{\mathcal{P}}(\mathcal{J})$ is an easy consequence of the Extension Property from Lemma 3.6, and we can conclude $ST_{\mathcal{P}}(\mathcal{I}) \subseteq ST_{\mathcal{P}}(\mathcal{J})$.

(ii) $ST_{\mathcal{P}}$ preserves the lubs of non-empty directed sets:

Assuming a non-empty directed set $\mathfrak{I} \subseteq \mathbb{I}_{\mathcal{D}}$, we must prove $ST_{\mathcal{P}}(\sqcup\,\mathfrak{I}) = \sqcup\, ST_{\mathcal{P}}(\mathfrak{I})$. The inclusion $ST_{\mathcal{P}}(\sqcup\,\mathfrak{I}) \supseteq \sqcup\, ST_{\mathcal{P}}(\mathfrak{I})$ holds because $ST_{\mathcal{P}}$ is monotonic. Since $\mathfrak{I}$ is not empty, the opposite inclusion can be rewritten as $ST_{\mathcal{P}}(\bigcup \mathfrak{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$, which is obviously a consequence of - $preST_{\mathcal{P}}(\bigcup \mathfrak{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$. In order to prove this last inclusion, assume an arbitrarily fixed c-fact φ belonging to the set $preST_{\mathcal{P}}(\bigcup \mathfrak{I})$. 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 $\bigcup \mathfrak{I} \Vdash_{\mathcal{D}} \varphi_i$. Therefore, by

the Compactness Property in Lemma 3.6, there must be some finite set of c-facts $\mathcal{I}_0 \subseteq \bigcup \mathfrak{I}$ such that $\varphi \in preST_{\mathcal{P}}(cl_{\mathcal{D}}(\mathcal{I}_0))$. Since $\mathfrak{I}$ is directed, there must be also some $\mathcal{I} \in \mathfrak{I}$ such that $\mathcal{I}_0 \subseteq \mathcal{I}$. Since $\mathcal{I}$ is closed under $cl_{\mathcal{D}}$, we obtain $cl_{\mathcal{D}}(\mathcal{I}_0) \subseteq \mathcal{I}$, and therefore (using the Extension Property from Lemma 3.6) $\varphi \in preST_{\mathcal{P}}(\mathcal{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$.

□

7.2 Proofs of the main results from section 4

Proof of Lemma 4.2

Proof. This lemma is proved in a similar way to the proof of 3.6. In (1) the property holds trivially because we only use almost one program rule instance of $\mathcal{P}$ in each step of the derivation. (2) is obvious using the fact that $\mathcal{P} \subseteq \mathcal{P}'$ if some program rule instance of $\mathcal{P}$ 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 $\mathcal{P} \vdash_{\mathcal{D}} \varphi'$ such that $\mid T \mid \geq \mid T' \mid$. Now, the only different case is in the application of the rule $\mathbf{DF}_{\mathcal{P}}$. If $T = \mathbf{DF}_{\mathcal{P}}(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, \overline{T}, T_r, T_s])$ 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 $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_\bot$ and some partial pattern $s \neq \bot$ such that $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$, $\overline{T} : \mathcal{P} \vdash_{\mathcal{D}} P \,\square\, \Delta \Leftarrow \Pi$, $\|\overline{T}\| < \|T\|$, $T_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$, $\|T_r\| < \|T\|$ and $T_s : \mathcal{P} \vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$, $\|T_s\| < \|T\|$. Since $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, we know $\varphi' = e' \to t' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$ and $t' \neq \bot$ (if $t' = \bot$ then T' consists of just one **TI** step and $\mid T \mid > 0 = \mid T' \mid$). From $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, it follows that $e' = f\,\overline{e'}_n\overline{a'}_k$ with $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ for all $1 \leq i \leq n$ and $\Pi' \models_{\mathcal{D}} a'_j \sqsupseteq a_j\sigma$ for all $1 \leq j \leq k$ (otherwise, for any total $\mu \in Sol_{\mathcal{D}}(\Pi')$ 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 $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ for all $1 \leq i \leq n$, $(P \,\square\, \Delta \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, \Delta)\sigma \Leftarrow \Pi')$, $(r \to s \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (r\sigma \to s\sigma \Leftarrow \Pi')$ and $(s\,\overline{a}_k \to t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi')$. By induction hypothesis (applied to T_i, $\overline{T}$, T_r, T_s), we get $T'_i : \mathcal{P} \vdash_{\mathcal{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$, $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq n)$, $\overline{T'} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, \Delta)\sigma \Leftarrow \Pi'$, $\mid \overline{T} \mid \geq \mid \overline{T'} \mid$, $T'_r : \mathcal{P} \vdash_{\mathcal{D}} r\sigma \to s\sigma \Leftarrow \Pi'$, $\mid T_r \mid \geq \mid T'_r \mid$, and $T'_s : \mathcal{P} \vdash_{\mathcal{D}} s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi'$, $\mid T_s \mid \geq \mid T'_s \mid$. Since $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta)\sigma \in [\mathcal{P}]_\bot$ 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 $\mathcal{P} \vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ with $k > 0$ and $t \neq \bot$), we can build the proof tree $T' =_{def} \mathbf{DF}_{\mathcal{P}}(f\overline{e'}_n\overline{a'}_k \to t' \Leftarrow$

Π', $[T'_1, \ldots, T'_n, \overline{T'}, T'_r, T'_s]$), which verifies $T' : \mathcal{P} \vdash_{\mathcal{D}} f\,\overline{e'}_n\,\overline{a'}_k \to t' \Leftarrow \Pi'$ 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 4.3

Proof. The stated result follows from the following three implications:

(i) $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^s_{\mathcal{D}} \varphi$:
Assume $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$. Consider any strong model $\mathcal{I} \models^s \mathcal{P}$. We prove $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ 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 $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$.

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 $CRWL(\mathcal{D})$, and using the induction hypothesis for $T_1, \ldots, T_n$.

The interesting case is that of the rule $\mathbf{DF}_{\mathcal{P}}$ applied at the root step. We consider the first variant of the rule $\mathbf{DF}_{\mathcal{P}}$ (the reasoning for the second one is similar).

The tree T would have the form

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

with $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\overline{T} : \mathcal{P} \vdash_{\mathcal{D}} P \,\square\, \Delta \Leftarrow \Pi$, $T_r : \mathcal{P} \vdash_{\mathcal{D}} r \to t \Leftarrow \Pi$, where $f \in DF^n$, $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_{\perp}$.

By the induction hypothesis applied to $\overline{T}, T_r$, there must exist $\overline{T'}, T'_r$ such that $\overline{T'} : \mathcal{I} \Vdash_{\mathcal{D}} P \square \Delta \Leftarrow \Pi$ and $T'_r : \mathcal{I} \Vdash_{\mathcal{D}} r \to t \Leftarrow \Pi$. This, together with the fact that $\mathcal{I}$ is a strong model of $\mathcal{P}$, ensures that $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$. Combining this with the rest of the induction hypothesis which ensures the existence of trees $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, 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 $\mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{e}_n \to t$.

(ii) $\mathcal{P} \models^s_{\mathcal{D}} \varphi \Rightarrow \mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$:
This holds simply because $\mathcal{S}_{\mathcal{P}} \models^s \mathcal{P}$, as proved in Theorem 3.15.

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

prove the following:

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

We prove that by induction on k.

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

A similar reasoning holds for the second case of the **DF**$_{\mathcal{I}}$ rule.

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

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

A similar reasoning holds for the second case of the **DF**$_{\mathcal{I}}$ rule.

□