A Fully Sound Goal Solving Calculus for the Cooperation of Solvers in the CFLP Scheme

S. Estévez Martín¹, A. J. Fernández²
M.T. Hortalá González¹, M. Rodríguez Artalejo¹
R. del Vado Vírseda¹

¹ Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid
² Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Abstract

The CFLP scheme for Constraint Functional Logic Programming has instances $CFLP(D)$ corresponding to different constraint domains $D$. In this paper, we propose an amalgamated sum construction for building coordination domains $C$, suitable to represent the cooperation among several constraint domains $D_1, \ldots, D_n$ via a mediatorial domain $M$. Moreover, we present a cooperative goal solving calculus for $CFLP(C)$, based on lazy narrowing, invocation of solvers for the different domains $D_i$ involved in the coordination domain $C$, and projection operations for converting $D_i$ constraints into $D_j$ constraints with the aid of mediatorial constraints (so-called bridges) supplied by $M$. Under natural correctness assumptions for the projection operations, the cooperative goal solving calculus can be proved fully sound w.r.t. the declarative semantics of $CFLP(C)$. As a relevant concrete instance of our proposal, we consider the cooperation between Herbrand, real arithmetic and finite domain constraints.

Keywords: Cooperative Goal Solving, Constraints, Functional-Logic Programming, Lazy Narrowing.

1 Introduction

The scheme CFLP for Constraint Functional Logic Programming, recently proposed in [11], continues a long history of attempts to combine the expressive power of functional and logic programming with the improvements in performance provided by domain specific constraint solvers. As the well-known CLP scheme [9], $CFLP$ has many possible instances $CFLP(D)$ corresponding to different specific constraint domains $D$ given as parameters. In spite of the generality of the approach,

---

1 Author partially supported by projects TIN2005-09207-C03-03 and S-0505/TIC0407.
2 Author partially supported by projects TIN2004-7943-C04-01 and TIN2005-08818-C04-01.

1571-0661/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2007.01.026
the use of one fixed domain $D$ is an important limitation, since many practical problems involve more than one domain.

A solution to this practical problem in the CLP context can be found in the concept of solver cooperation [5], an issue that is raising an increasing interest in the constraint community. In general, solver cooperation aims at overcoming two problems: a lack of declarativity of the solutions (i.e., the interaction among solvers makes it easier to express compound problems) and a poor performance of the systems (i.e., the communication among solvers can improve the efficiency of the solving process).

This paper presents a proposal for coordinated programming in the CFLP scheme as described in [11]. We introduce coordination domains as amalgamated sums of the various domains to be coordinated, along with a mediatiorial domain which supplies special communication constraints, called bridges, used to impose equivalences among values of different base types. Building upon previous works [2,10,15], we also describe a coordinated goal solving calculus which combines lazy narrowing with the invocation of the cooperating solvers and two kinds of communication operations, namely the creation of bridges and the projection of constraints between different constraint stores. Projection operations are guided by existing bridges. Using the declarative semantics of CFLP, we have proved a semantic result called full soundness, ensuring soundness and local completeness of the goal solving calculus.

In order to place our proposal for solver cooperation in context, we briefly discuss main differences and similarities with a limited selection of related proposals existing in the literature. E. Monfroy [14] proposed the system BALI (Binding Architecture for Solver Integration) that facilitates the specification of solver cooperation as well as integration of heterogeneous solvers via a number of cooperations primitives. Monfroy’s approach assumes that all the solvers work over a common store, while our present proposal requires communication among different stores. Also, Mircea Marin [12] developed a CFLP scheme that combines Monfroy’s approach to solver cooperation with a higher-order lazy narrowing calculus somewhat similar to [10,15] and the goal solving calculus presented in this paper. In contrast to our proposal, Marin’s approach allows for higher-order unification, which leads both to greater expressivity and to less efficient implementations. Moreover, the instance of CFLP implemented by Marin and others [13] combines four solvers over a constraint domain for algebraic symbolic computation, while the instance we are currently implementing deals with the cooperation among Herbrand, finite domain and real arithmetic constraints. Recently, P. Hofstedt [7,8] proposed a general approach for the combination of various constraint systems and declarative languages into an integrated system of cooperating solvers. In Hofstedt’s proposal, the goal solving procedure of a declarative language is viewed also as a solver, and cooperation of solvers is achieved by two mechanisms: constraint propagation, that submits a constraint belonging to some domain $D$ to its constraint store, say $S_D$; and projection of constraint stores, that consults the contents of a given store $S_D$ and deduces constraints for another domain. Projection, as used in this paper,
differs from Hofstedt’s projection in the creation and use of bridges; while Hofstedt’s propagation corresponds to our goal solving rules for placing constraints in stores and invoking constraint solvers. Hofstedt also proposes the construction of combined computation domains, similar to our coordination domains. The lack of bridges in Hofstedt’s approach corresponds to the lack of mediatiorial domains within her combined domains. In different places along the paper we will include comparisons to Hofstedt’s approach; see especially Table 5 in Section 5.

The structure of the paper is as follows: Section 2 introduces the basic notions of constraint domains and solvers underlying the CFLP scheme. Section 3 describes the constructions needed for coordination in our setting, namely coordination domains, bridges and projections. Programs, goals, the lazy narrowing calculus for cooperative goal solving (with a typical example), and the full soundness result are described in Section 4. Section 5 summarizes conclusions and future work.

2 Constraint Domains and Solvers in the CFLP Scheme

In this section, we recall the essentials of the CFLP(\mathcal{D}) scheme [11], which serves as a logical and semantic framework for lazy Constraint Functional Logic Programming (briefly CFLP) over a parametrically given constraint domain \mathcal{D}. The proper choice of \mathcal{D} for modeling the coordination of several constraint domains will be discussed in Section 3. As a main novelty w.r.t. [11], the current presentation of CFLP(\mathcal{D}) includes now an explicit treatment of a Milner-like polymorphic type system in the line of previous work in Functional Logic Programming [4].

2.1 Signatures and Constraint Domains

We assume a universal signature \Sigma = \langle TC, DC, DF \rangle, where TC = \bigcup_{n \in \mathbb{N}} TC^n, DC = \bigcup_{n \in \mathbb{N}} DC^n and DF = \bigcup_{n \in \mathbb{N}} DF^n are families of countably infinite and mutually disjoint sets of type constructor, data constructor and defined function symbols, respectively. We also assume a countable set T\mathcal{V} of type variables.

Types \tau \in Type_{\Sigma} have the syntax \tau ::= \alpha \mid C \tau_1 \ldots \tau_n \mid (\tau_1, \ldots, \tau_n) \mid \tau \rightarrow \tau’, where \alpha \in T\mathcal{V} and C \in TC^n. By convention, C \rightarrow_n abbreviates C \tau_1 \ldots \tau_n, “\rightarrow” associates to the right, \rightarrow_n \tau abbreviates \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau, and the set of type variables occurring in \tau is written T\mathcal{V}(\tau). A type \tau is called monomorphic iff T\mathcal{V}(\tau) = \emptyset, and polymorphic otherwise. Types C \rightarrow_n, (\tau_1, \ldots, \tau_n) and \tau \rightarrow \tau’ are used to represent constructed values, tuples and functions, respectively. A type without any occurrence of “\rightarrow” is called a datatype. Each n-ary c \in DC^n comes with a principal type declaration c :: \tau_n \rightarrow C \rightarrow \tau_k, where n, k \geq 0, \alpha_1, \ldots, \alpha_k are pairwise different, \tau_i are datatypes, and T\mathcal{V}(\tau_i) \subseteq \{\alpha_1, \ldots, \alpha_k\} for all 1 \leq i \leq n. Also, each n-ary f \in DF^n comes with a principal type declaration f :: \tau_n \rightarrow \tau, where \tau_i, \tau are arbitrary types. For the sake of semantic considerations, we assume a special data constructor (\bot :: \alpha) \in DC^0, intended to represent an undefined data value that belongs to every type. 3

3 In concrete programming languages such as TOY [1] and Curry [6], data constructors and their principal types are introduced by datatype declarations, the principal types of defined functions can be either declared or inferred, and \bot does not textually occur in programs.
Intuitively, a constraint domain provides specific data elements, along with certain primitive functions operating upon them. Following this idea, and extending the formal approach of [11] with a type system, we consider domain specific signatures \( \Gamma = (BT, PF) \) disjoint from \( \Sigma \), where \( BT \) is a family of base types (such as \( \text{int} \) for integer numbers or \( \text{real} \) for real numbers) and \( PF \) is a family of primitive function symbols, each one with an associated principal type declaration \( p : \tau_1 \to \ldots \to \tau_n \to \tau \) (shortly, \( p : \tau_\text{n} \to \tau \)), where \( \tau_1, \ldots, \tau_n \) and \( \tau \) are datatypes. The number \( n \) is called arity of \( p \), and the set of \( n \)-ary symbols in \( PF \) is noted as \( PF^n \).

A constraint domain over a specific signature \( \Gamma \) (in short, \( \Gamma \)-domain) is a structure \( D := (\{U_d\}_{d \in BT}, \{p^D\}_{p \in PF}) \), where each \( d \in BT \) is interpreted as a non-empty set \( U_d \) of base elements of type \( d \), as e.g., \( \mathbb{Z} = U_{\text{int}} \) or \( \mathbb{R} = U_{\text{real}} \); and interpretations \( p^D \) of primitive function symbols behave as explained in Subsection 2.3 below.

2.2 Extended Types, Expressions, Patterns and Substitutions over a Domain \( D \)

Given a \( \Gamma \)-domain \( D \), extended types \( \tau \in Type_{\Sigma, \Gamma} \) over \( \Gamma \) have the syntax \( \tau := \alpha \mid d \mid C \tau_1 \ldots \tau_n \mid \tau \to \tau' \mid (\tau_1, \ldots, \tau_n) \), where \( d \in BT_\Gamma \). Obviously, \( Type_{\Sigma} \subseteq Type_{\Sigma, \Gamma} \).

Given a countable infinite set \( \mathfrak{V} \) of data variables disjoint from \( \mathcal{T} \mathfrak{V} \), \( \Sigma \) and \( \Gamma \), expressions \( e \in Exp_D \) over \( D \) have the syntax \( e := X \mid u \mid h \mid (e_1 e_2) \), where \( X \in \mathfrak{V} \), \( u \in U_D = \bigcup_{d \in BT_\Gamma} U_d \), and \( h \in DC_{\Sigma} \cup DF_{\Sigma} \cup PF_\Gamma \). Note that \((e_1 e_2)\) - not to be confused with the pair \((e_1, e_2)\) - stands for the application operation which applies the function denoted by \( e \) to the argument denoted by \( e_1 \). Following usual conventions, we assume that application associates to the left, and we abbreviate \((e_1 \ldots e_n)\) as \((e \tau_n)\). Expressions without repeated variable occurrences are called linear, variable-free expressions are called ground and expressions without any occurrence of \( \bot \) are called total. Patterns over \( D \) are special expressions \( t \in Pat_D \) whose syntax is defined as \( t := X \mid u \mid (c \tau_m) \mid (f \tau_m) \mid (p \tau_m) \), where \( X \in \mathfrak{V} \), \( u \in U_D \), \( c \in DC_{\Sigma}^n \) with \( m \leq n \), \( f \in DF_{\Sigma}^n \) with \( m < n \), and \( p \in PF_{\Gamma}^n \) with \( m < n \). The set of all ground patterns over \( D \) is noted \( GPat_D \). The following classification of expressions is also useful: \((X \tau_m)\), with \( X \in \mathfrak{V} \), and \( m \geq 0 \), is called a flexible expression, while \( u \in U_D \) and \((h \tau_m)\) with \( h \in DC_{\Sigma} \cup DF_{\Sigma} \cup PF_\Gamma \) are called rigid expressions. Moreover, a rigid expression \((h \tau_m)\) is called active iff \( h \in DF_{\Sigma}^n \cup PF_{\Gamma}^n \) and \( m \geq n \), and passive otherwise.

We also consider substitutions \( \sigma, \theta \in Subs_D \) over \( D \) as mappings from variables to patterns, and by convention, we write \( \varepsilon \) for the identity substitution, \( \sigma \varepsilon \) instead of \( \sigma(e) \) for any \( e \in Exp_D \), and \( \sigma \theta \) for the composition of \( \sigma \) and \( \theta \). A substitution \( \sigma \) such that \( \sigma \sigma = \sigma \) is called idempotent. The domain \( \text{Vdom}(\sigma) \subseteq \mathfrak{V} \) and variable range \( \text{Vran}(\sigma) \subseteq \mathfrak{V} \) of \( \sigma \) are defined as usual. For any set of variables \( \chi \subseteq \mathfrak{V} \) we define the restriction \( \sigma \mid \chi \) as the substitution \( \sigma' \) such that \( \text{Vdom}(\sigma') = \chi \) and \( \sigma'(X) = \sigma(X) \) for all \( X \in \chi \). Given \( \chi \subseteq \mathfrak{V} \), we write \( \sigma =_\chi \theta \) to indicate that \( \sigma \mid \chi = \theta \mid \chi \), and we abbreviate \( \sigma =_\mathfrak{V} \chi \theta \) as \( \sigma =_\chi \theta \). Type substitutions mapping type variables to types can be defined analogously. Monomorphic instances \( \tau' \) of a given type \( \tau \) can be obtained by applying type substitutions to \( \tau \).

Finally, we define the information ordering \( \sqsubseteq_D \) as the least partial ordering over \( Exp_D \) such that \( \bot \sqsubseteq_D e \) for all \( e \in Exp_D \) and \( (e e_1) \sqsubseteq_D (e' e'_1) \) whenever \( e \sqsubseteq_D e' \) and \( e_1 \sqsubseteq_D e'_1 \). The information ordering is useful for semantic considerations.
2.3 Interpreting Primitive Function Symbols

Assume a specific signature $\Gamma = (BT, PF)$ and a $\Gamma$-domain $D$. We define the carrier set $D_D$ of $D$ as the set $GPat_D$ of all the ground patterns over $D$. For each $p \in PF^n$ whose declared principal type in $\Gamma$ is $p:\tau_n \rightarrow \tau$, the interpretation of $p$ must be a set of tuples $p^D \subseteq D^n_{\tau+1}$. By convention, we write $p^D \tilde{t}_n \rightarrow t$ to indicate $(\tilde{t}_n, t) \in p^D$. Moreover, $p^D$ is required to satisfy three conditions:

(i) **Polarity**: For all $\tilde{t}_n, t, t' \in D_D$, if $p^D \tilde{t}_n \rightarrow t, \tilde{t}_n \subseteq D \tilde{t}_n$ and $t \sqsupseteq D$ then $p^D \tilde{t}_n \rightarrow t'$ (i.e., monotonicity w.r.t. arguments and antimonotonicity w.r.t. result).

(ii) **Radicality**: For all $\tilde{t}_n, t \in D_D$, if $p^D \tilde{t}_n \rightarrow t$ then $t = \bot$ or else there is some total $t' \in D_D$ such that $t' \sqsupseteq D t$, and $p^D \tilde{t}_n \rightarrow t'$.

(iii) **Well-Typedness**: For all monomorphic $\tilde{\tau}_n, \tau' \in Type_{\Sigma, \Gamma}$ and all $\tilde{t}_n, t \in D_D$, if $\tilde{\tau}_n \rightarrow \tau'$ is a monomorphic instance of $\tau_n \rightarrow \tau$, $D \vdash_MT \tilde{t}_n :: \tilde{\tau}_n$ and $p^D \tilde{t}_n \rightarrow t$ then $D \vdash_MT t :: \tau'$ (where $D \vdash_MT \tilde{t}_n :: \tilde{\tau}_n$ abbreviates $D \vdash_MT t_1 :: \tilde{\tau}_1, \ldots, D \vdash_MT t_n :: \tilde{\tau}_n$).

Type judgements of the form $D \vdash_MT t :: \tau'$ as used in item (iii) above mean that $\tau'$ is a monomorphic instance of $e$’s principal type, and can be derived by well-known type inference rules, see e.g. [4].

2.4 Constraint Solutions and Constraint Solvers

Constraints over a given $\Gamma$-domain $D$ are logical statements built from atomic constraints by means of logical conjunction $\wedge$ and existential quantification $\exists$. Atomic constraints can have the form $\Diamond$ (standing for truth), $\lozenge$ (standing for falsity), or $p \bar{\tau}_n \rightarrow !t$ with $p \in PF_n$, $\bar{\tau}_n \in Exp_D$ and $t \in Pat_D$ total. Atomic primitive constraints have the form $\Diamond$, $\lozenge$ or $p \bar{\tau}_n \rightarrow !t$ with $\bar{\tau}_n \in Pat_D$. In the sequel, the set of all primitive constraints (resp. atomic primitive constraints) over $D$ is noted $PCon(D)$, (resp. $APCon(D)$). Three concrete constraint domains considered in this paper are:

- The **Herbrand domain** $\mathcal{H}$, with no specific base type, which supports syntactic equality and disequality constraints $\text{seq} e_1 e_2 \rightarrow !t$ (abbreviated as $e_1 = e_2$ resp. $e_1 /= e_2$ when $t$ is true resp. false) over elements of any type. See [11] for details.
- $\mathcal{F}_D$, with specific base type $\text{int}$, which supports finite domain constraints over $U_{\text{int}}^{\mathcal{F}_D} = \mathbb{Z}$ and the primitive functions described in [3] and summarized in Table 1.
- $\mathcal{R}_D$, with specific base type $\text{real}$, which supports real arithmetic constraints over $U_{\text{real}}^{\mathcal{R}_D} = \mathbb{R}$ and the primitive functions described in [11] and summarized in Table 2.

Ground substitutions $\eta$ over $D$ are called valuations. The set of all valuations over $D$ is denoted $Val_D$. For any $\pi \in PCon(D)$, $Sol_D(\pi) = \{\eta \in Val_D | \eta \text{ satisfies } \pi\}$ can be defined in a natural way; see [11] for details. Moreover, the set of solutions of $\Pi \subseteq PCon(D)$ is defined as $Sol_D(\Pi) = \bigcap_{\pi \in \Pi} Sol_D(\pi)$. Therefore, sets of constraints are interpreted as conjunctions. A variable $X \in \text{var}(\Pi)$ such that $\eta(X) \neq \bot$ for all $\eta \in Sol_D(\Pi)$ is said to be demanded by $\Pi$. In practical constraint domains, the set of variables demanded by $\Pi$ is expected to be decidable.

For any constraint domain $D$ we postulate a constraint solver given as a function $solve^D$ such that for any finite $\Pi \subseteq APCon(D)$, $solve^D(\Pi)$ returns a finite disjunction $\bigvee_{j=1}^k \exists \bar{y}_j$. $(\Pi_j \sqcap \sigma_j)$ fulfilling the following correctness conditions:
For all $1 \leq j \leq k$: $Y_j$ are new variables, $\Pi_j \subseteq \text{APCon}(\mathcal{D})$ finite, $\sigma_j$ idempotent substitution such that $\forall \text{dom}(\sigma_j) \subseteq \text{var}(\Pi)$, $\forall \text{ran}(\sigma_j) \subseteq Y_j$, and $\text{solve}^D(\Pi_j) = \Pi_j \cap \varepsilon$.

$\text{Sol}_D(\Pi) = \bigcup_{j=1}^k \text{Sol}_D(\exists Y_j \cdot (\Pi_j \sqcap \sigma_j))$ (where $\sqcap$ is interpreted as conjunction).

$\Pi$ is called a solved form iff $\text{solve}^D(\Pi) = \Pi \cap \varepsilon$. In the sequel, we will use the following notations:

- $\Pi \vdash_{\text{solve}} \exists Y_j$. ($\Pi' \sqcap \sigma'$) to indicate that $\exists Y_j$. ($\Pi_j \sqcap \sigma_j$) for some $1 \leq j \leq k$ (successful solving step).
- $\Pi \vdash_{\text{solve}} \blacksquare$ to indicate that $k = 0$ (failing solving step; in this case, $\text{Sol}_D(\Pi) = \emptyset$).

A solving step $\Pi \vdash_{\text{solve}} \exists Y_j$. ($\Pi' \sqcap \sigma'$) is called admissible w.r.t. a set of variables $\mathcal{U}$ iff the two following conditions hold:

- $\mathcal{U} \sigma'$ is a set of pairwise variable-disjoint linear patterns.
- Either $\mathcal{U} \cap \text{var}(\Pi') = \emptyset$ or else some variable in $\mathcal{U}$ is demanded by $\Pi'$.

This notion will be used in the goal solving calculus presented in Section 4.

## 3 Coordination of Domains in the CFLP Scheme

In this section, we describe the construction of the coordination domain $C$ built from various domains $\mathcal{D}_i$, intended to cooperate, and a mediatorial domain $\mathcal{M}$, which supplies special communication constraints called bridges. Instances $\text{CFLP}(C)$, where $C$ is a coordination domain, provide a declarative semantic framework for cooperative CFLP programming and goal solving.

### 3.1 Mediatorial and Coordination Domains

Assume a $\Gamma$-domain $\mathcal{D}$ and a $\Gamma'$-domain $\mathcal{D}'$ with specific signatures $\Gamma = \langle \mathcal{B}T, PF \rangle$ and $\Gamma' = \langle \mathcal{B}T', PF' \rangle$. $\mathcal{D}$ and $\mathcal{D}'$ are called joinable iff $PF \cap PF' = \emptyset$ and $\mathcal{U}^D_d = \mathcal{U}^D_{d'}$ for all $d \in BT \cap BT'$. The amalgamated sum $\mathcal{D} \oplus \mathcal{D}'$ of two joinable domains $\mathcal{D}$ and $\mathcal{D}'$ is a new domain with specific signature $\Gamma'' = \langle BT'', PF'' \rangle$ where $BT'' = BT \cup BT'$, $PF'' = PF \cup PF'$, and is constructed as follows:

- For all $d \in BT$, $\mathcal{U}^D_d'' = \mathcal{U}^D_d$, and for all $d \in BT'$, $\mathcal{U}^{D'}_d'' = \mathcal{U}^{D'}_d$.
- For all $p \in PF$ and all $\bar{t}_n, t \in D_{\mathcal{D}'}$: $p^{\mathcal{D}''} \bar{t}_n \rightarrow t \Leftarrow_{\text{def}}$ either $\bar{t}_n \in D_{\mathcal{D}}$, $t \in D_{\mathcal{D}}$ and $p^{\mathcal{D}} \bar{t}_n \rightarrow t$, or else $t = \bot$.
- For all $p \in PF'$ and all $\bar{t}_n, t \in D_{\mathcal{D}'}$: $p^{\mathcal{D}''} \bar{t}_n \rightarrow t \Leftarrow_{\text{def}}$ either $\bar{t}_n \in D_{\mathcal{D}'}, t \in D_{\mathcal{D}'}$ and $p^{\mathcal{D}} \bar{t}_n \rightarrow t$, or else $t = \bot$.

The amalgamated sum of $n$ pairwise joinable domains can be defined analogously.

Assume $n$ pairwise joinable domains $\mathcal{D}_i$ with specific signatures $\Gamma_i = \langle BT_i, PF_i \rangle$ ($1 \leq i \leq n$) and another domain $\mathcal{M}$ with specific signature $\Gamma_0 = \langle BT_0, PF_0 \rangle$. $\mathcal{M}$ is called a mediatorial domain for $\mathcal{D}_1, \ldots, \mathcal{D}_n$ iff

- $BT_0 = \bigcup_{i=1}^n BT_i$, and for all $1 \leq i \leq n$: $PF_0 \cap PF_i = \emptyset$.
- For each $p \in PF_0$ there exists $1 \leq i, j \leq n$, $d_i \in BT_i$ and $d_j \in BT_j$ such that $p$ is an equivalence primitive $\text{equiv}_{d_i, d_j} : d_i \rightarrow d_j \rightarrow \text{bool}$ and there is an injective partial mapping $\text{inj}_{d_i, d_j} : \mathcal{U}^D_{d_i} \rightarrow \mathcal{U}^D_{d_j}$ such that, for all $t_1, t_2, t \in D_M$: $\text{equiv}_{d_i, d_j}^M t_1 t_2 \rightarrow t \Leftarrow_{\text{def}}$ $t_1 \in \text{dom}(\text{inj}_{d_i, d_j})$, $t_2 = \text{inj}_{d_i, d_j}(t_1)$ and true $\sqsubseteq_M t$, or else $t_1 \in \text{dom}(\text{inj}_{d_i, d_j})$, $t_2 \in \mathcal{U}^D_{d_j}$, $t_2 \neq \text{inj}_{d_i, d_j}(t_1)$ and false $\sqsubseteq_M t$, or else $t = \bot$.

We note that, for fixed $i, j, 1 \leq i, j \leq n$:
• If \( d \in BT_i \cap BT_j \), an equivalence primitive \( \equiv_{d,d} : d \rightarrow d \rightarrow \text{bool} \) can be defined if wished, whose interpretation \( \equiv_{d,d}^M \) is based on the identity function \( \text{id} : U_d^D_i \rightarrow U_d^D_j \) (due to the joinability requirements). The primitive \( \equiv_{d,d} \) may be useful for communication purposes in case that \( D_i \) and \( D_j \) have different primitives involving the common base type \( d \).

• There can be none, one, or more than one possibilities of choosing base types \( d_i \in BT_i \), \( d_j \in BT_j \) such that an equivalence primitive \( \equiv_{d_i,d_j} : d_i \rightarrow d_j \rightarrow \text{bool} \) is available in \( \mathcal{M} \). An equivalence primitive is called redundant iff there is some other equivalence primitive whose interpretation is based on the same partial injection or its inverse. We assume that no redundant equivalence primitives are available in \( \mathcal{M} \). If \( \equiv_{d_i,d_j} \) is available in \( \mathcal{M} \) for some \( d_i \in BT_i \), \( d_j \in BT_j \), we say that \( D_i \) and \( D_j \) are comparable.

Assume now \( n \) given pairwise joinable domains \( D_1, \ldots, D_n \) with specific signatures \( \Gamma_1, \ldots, \Gamma_n \) and a mediatorial domain \( \mathcal{M} \) for \( D_1, \ldots, D_n \). Then, the \( n+1 \) domains \( \mathcal{M}, D_1, \ldots, D_n \) are pairwise joinable, and the amalgamated sum \( \mathcal{C} = \mathcal{M} \oplus D_1 \oplus \ldots \oplus D_n \) can be built. In the sequel, we assume that the Herbrand domain \( \mathcal{H} \) is taken as one of the \( D_i \), and thus \( \mathcal{C} = \mathcal{M} \oplus \mathcal{H} \oplus D_1 \oplus \ldots \oplus D_n \). Such a \( \mathcal{C} \) is called a coordination domain, because CFLP(\( \mathcal{C} \)) supports coordinated CFLP programming, using bridge constraints of the form \( e_1 \equiv_{d_i,d_j} e_2 = \text{def} \equiv_{d_i,d_j} e_1 e_2 \rightarrow \text{true} \) for communication between \( D_i \) and \( D_j \) (this will work for all the equivalence primitives available in the mediatorial domain \( \mathcal{M} \)).

The instance CRWL(\( \mathcal{C} \)) of the Constraint ReWriting Logic CRWL presented in [11] provides a declarative semantics for CFLP(\( \mathcal{C} \)) programming, whose usefulness for correctness results will be seen in Subsection 4.4.

### 3.2 Bridges and Projections for Cooperative Goal Solving

The cooperative goal solving calculus for CFLP(\( \mathcal{C} \)) described in Section 4 below, stores bridge constraints in a special store \( \mathfrak{M} \) and uses them for enabling cooperation between different solvers. More precisely, bridge constraints of the form \( e_1 \equiv_{d_i,d_j} e_2 \) can be used either for binding or projection purposes. Binding simply instantiates a variable occurring at one end of a bridge whenever the other end of the bridge becomes a primitive value. Projection is a more complex operation which infers constraints to be placed in \( D_j \)'s store from the constraints available in \( D_i \)'s store and the relevant bridges available in \( \mathfrak{M} \). This enables each solver to take advantage of the computations performed by other solvers. For every pair \( i,j \) such that \( D_i \) and \( D_j \) are comparable, we postulate a projection function \( \text{projections}^{D_i \rightarrow D_j} \) such that for any \( \pi \in APCon(D_i) \) and any finite set \( M \) of bridge constraints, \( \text{projections}^{D_i \rightarrow D_j}(\pi, M) \) returns a finite disjunction \( \bigvee_{k=1}^l \exists Y_k \). \( \Pi'_k \) fulfilling the following safety conditions:

• For all \( 1 \leq k \leq l \): \( Y_k \) are new variables, and \( \Pi'_k \subseteq APCon(D_j) \) is finite.

• \( \text{Sol}_C(\pi \wedge M) \subseteq \bigcup_{k=1}^l \text{Sol}_C(\exists Y_k \wedge (\pi \wedge \Pi'_k \wedge M)) \) (where \( M \) and \( \Pi'_k \) are interpreted as conjunctions).

In the sequel, we use the notation \( (\pi, M) \not\models_{\text{projections}^{D_i \rightarrow D_j}} \exists Y' \). \( \Pi' \) to indicate that \( \exists Y' \). \( \Pi' \) is \( \exists Y_k \). \( \Pi'_k \) for some \( 1 \leq k \leq l \) (successful projection step). Our projections
are inspired by those of [7,8], but our proposal of bridge constraints is a novelty. Following the terminology of [8], we say that a projection returning \( k \) alternatives is strong if \( k > 1 \) and weak otherwise.

In order to maximize the opportunities for projection, we postulate for each pair \( i, j \) such that \( D_i \) and \( D_j \) are comparable a function \( \text{bridges}^{D_i \to D_j} \) such that for any \( \pi \in \text{APCon}(D_i) \) and any finite set \( M \) of bridge constraints, \( \text{bridges}^{D_i \to D_j}(\pi, M) \) returns a finite set \( M' \) of new bridge constraints involving new variables \( \overline{V} \), so that the following safety condition holds: \( \text{Sol}_C(\pi \land M) \subseteq \text{Sol}_C(\exists \overline{V}. (\pi \land M \land M')) \) (where \( M \) and \( M' \) are interpreted as conjunctions).

<table>
<thead>
<tr>
<th>( \pi \in \text{APCon}(\mathcal{F}D) )</th>
<th>( \text{bridges}^{\mathcal{F}D \to \mathcal{R}}(\pi, M) )</th>
<th>( \text{projections}^{\mathcal{F}D \to \mathcal{R}}(\pi, M) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{domain}[X_1,\ldots,X_n] a b</td>
<td>{X_i#==RX_i \mid 1\leq i\leq n, X_i \text{ has no bridge in } M, RX_i \text{ new} }</td>
<td>{a \leq RX_i, RX_i \leq b \mid 1\leq i\leq n, (X_i#==RX_i) \in M }</td>
</tr>
<tr>
<td>\text{belongs } X [a_1,\ldots,a_n]</td>
<td>{X#==RX \mid X \text{ has no bridge in } M, RX \text{ new} }</td>
<td>{\min(a_1,\ldots,a_n)\leq RX, RX\leq\max(a_1,\ldots,a_n) \mid 1\leq i\leq n, (X#==RX) \in M }</td>
</tr>
<tr>
<td>( t_1 #&lt;t_2 ) (analogously #&lt;,#,#&gt;,#==,#==)</td>
<td>{X_i#==RX_i \mid 1\leq i\leq 2, t_i \text{ is a variable } X_i \text{ with no bridge in } M, RX_i \text{ new} }</td>
<td>{t^R_1 &lt; t^R_2 \mid \text{For } 1\leq i\leq 2: \text{either } t_i \text{ is an integer constant and } t^R_i \text{ is } n, \text{ or else } t_i \text{ is a variable } X_i, (X_i#==RX_i) \in M, \text{and } t^R_i \text{ is } RX_i }</td>
</tr>
<tr>
<td>( t_1 #+t_2 \rightarrow t_3 ) (analogously #-,#*)</td>
<td>{X_i#==RX_i</td>
<td>1\leq i\leq 3,t_i \text{ is a variable } X_i \text{ with no bridge in } M,RX_i \text{ new} }</td>
</tr>
</tbody>
</table>

Table 1

Bridge Constraints and Projections from \( \mathcal{F}D \) to \( \mathcal{R} \)

As a concrete example, Table 1 and Table 2 show a partial description of the functions \( \text{bridges} \) and \( \text{projections} \) between the comparable domains \( \mathcal{F}D \) and \( \mathcal{R} \), where bridge constraints written as \( u\#==v \) are based on an equivalence primitive \( \text{equiv} :: \text{int} \rightarrow \text{real} \rightarrow \text{bool} \). The tables do not show all possible cases due to lack of space. Some cases omitted here can be found in [2].

4 Coordinated CFLP Programming

In this section, we discuss the syntax of \( \text{CFLP}(\mathcal{C}) \)-programs and admissible goals for programs, in order to set the basis for coordinated programming in the \( \text{CFLP} \) scheme using lazy narrowing with cooperation of constraint solvers.

4.1 Structure of Program Rules and Goals

\( \text{CFLP}(\mathcal{C}) \)-programs are sets of constrained rewriting rules that define the behavior of possibly higher-order and/or non-deterministic lazy functions over \( \mathcal{C} \), called program rules. More precisely, a program rule for a defined function symbol \( f \in \)

---

4 Projections in [8] depend on the set of variables common to the stores of \( D_i \) and \( D_j \). In our \( \text{CFLP} \) framework, well-typing usually prevents the occurrence of one and the same variable in two different stores.
$DF^n_3$ with principal type $\tau_n \rightarrow \tau$ has the form $f \tau_n = r \Leftarrow C$, where $f \in DF^n_3$, $\tau_n$ is a linear sequence of patterns, $r$ is an expression and $C$ is a finite conjunction $\delta_1, \ldots, \delta_m$ of atomic constraints $\delta_i$ for each $1 \leq i \leq m$, possibly including occurrences of defined function symbols. Program rules are required to be well-typed.  

As an example for the rest of the paper, we consider the following program fragment adapted from [8] and written in $TOY$ syntax [1]. Function $rc$ computes the capacity of circuits built from a set of resistors with given capacities by means of sequential and parallel composition. The program rules involve typical constraints over the domains $\mathcal{FD}$ and $\mathcal{R}$, as well as cooperation via communication bridges $X \#== C$ with $X :: \mathit{int}$ and $C :: \mathit{real}$.

<table>
<thead>
<tr>
<th>$\pi \in \mathsf{APCon}(\mathcal{R})$</th>
<th>bridges$^{\mathcal{R}} \rightarrow \mathcal{FD}(\pi, M)$</th>
<th>projections$^{\mathcal{R}} \rightarrow \mathcal{FD}(\pi, M)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RX &lt;= RY$</td>
<td>$\emptyset$ (no bridges are created)</td>
<td>{X$#== Y</td>
</tr>
<tr>
<td>$RX &lt;= a$</td>
<td>$\emptyset$ (no bridges are created)</td>
<td>{X$#== [a]</td>
</tr>
<tr>
<td>$t_1 + t_2 \rightarrow ! t_3$ (analogously for $-$, $\ast$)</td>
<td>{X$#== RX</td>
<td>t_3$ is a variable RX with no bridge in M, X new, for $1 \leq i \leq 2$, $t_i$ is either an integer constant or a variable RX$^i$ with bridge (X$i$ $#== RX$) $\in M$}</td>
</tr>
</tbody>
</table>

Table 2
Bridge Constraints and Projections from $\mathcal{R}$ to $\mathcal{FD}$

In the sequel, we consider $CFLP(\mathcal{C})$-goals in the general form $G \equiv \exists \overline{X}. P \square C \square M \square H \square S_1 \square \ldots \square S_n$, in order to represent a generic state of the computation with cooperation of solvers over the coordination domain $\mathcal{C} = \mathcal{M} \oplus \mathcal{H} \oplus \mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_n$. The symbol $\square$ is interpreted as conjunction and,

- $\overline{U}$ is a finite set of so-called existential variables, intended to represent local variables in the computation.
- $P$ is a set of so-called productions of the form $e_1 \rightarrow t_1, \ldots, e_m \rightarrow t_m$, where $e_i \in \mathit{Exp}_{\mathcal{FD}}$ and $t_i \in \mathit{Pat}_{\mathcal{FD}}$ for all $1 \leq i \leq m$.  
  The set of produced variables of $G$ is defined as the set $\mathit{pvar}(P)$ of variables occurring in $t_1, \ldots, t_m$.
- $C$ is a finite set of constraints to be solved, possibly including active occurrences of defined functions symbols.
- $M$ is the so-called mediatorial store including bridge constraints of one of the four

---

5 The notion of well-typed $CFLP$ program can be formalized by an easy extension of [4].

6 A production $e_i \rightarrow t_i$ can be viewed as a suspension. It is solved by evaluating $e_i$ by lazy narrowing and unifying the result with $t_i$. 

---
following forms: \( X\#==d_i,d_jX' \) or \( u\#==d_i,d_ju' \) or \( X\#==d_i,d_jX' \) or \( u\#==d_i,d_ju' \), where \( 1 \leq i,j \leq n \) are such that \( D_i \) and \( D_j \) are comparable, \( X,X' \) are variables, \( u \in U_{d_i}^D \) and \( u' \in U_{d_j}^D \).

- \( H \) is the so-called Herbrand store, including a finite set \( \Pi \) of atomic primitive \( \mathcal{H} \)-constraints and an answer substitution \( \theta \) with variable bindings. We use the notation \((\Pi \square \theta)\) to represent the store \( H \).

- \( S_i \) (\( 1 \leq i \leq n \)) is a \( D_i \) constraint store associated to the domain \( D_i \), including a finite set \( \Pi_i \subseteq P\text{Con}(D_i) \) of atomic primitive \( D_i \)-constraints and an answer substitution \( \theta_i \) with variable bindings. We use the notation \((\Pi_i \square \theta_i)\) to represent the structure of the store \( S_i \).

We work with admissible goals \( G \) satisfying the goal invariants given in [10,15]. We also write \( \blacksquare \) to denote an inconsistent goal. Moreover, we say that a variable \( X \) is a demanded variable in a goal \( G \) iff \( X \) is demanded by some of the constraint stores occurring in \( G \) in the sense explained in Subsection 2.4. For example, \( X \) is demanded by the \( \mathcal{F} \mathcal{D} \) constraint \( X \#==3 \), but not demanded by the \( \mathcal{H} \) constraint \( \text{suc}X \#= \text{zero} \), where \( \text{suc} \) and \( \text{zero} \) are constructor symbols.

Two special kinds of admissible goals are useful. Initial goals, consisting just of a finite conjunction \( C \) of constraints and without any existential variables; and solved goals (also called solved forms), consisting of a conjunction of constraint stores in solved form (\( H, M \) and \( S_i \), for each \( 1 \leq i \leq n \)) and empty \( P \) and \( C \) parts, possibly with existential variables.

In the sequel, we use the following notations in order to indicate the transformation of a goal by applying a substitution \( \sigma \) and also adding \( \sigma \) to the corresponding store \( H \) or \( S_i \) (\( 1 \leq i \leq n \)):

- \( (P\square C\square M\square H\square S_1\square \ldots \square S_n)@H:\sigma=_{\text{def}}(P\sigma\square C\sigma\square M\sigma\square H \upharpoonright \sigma \square S_1\sigma\square \ldots \square S_n\sigma) \), where \( H \upharpoonright \sigma \equiv (\Pi \square \theta) \upharpoonright \sigma =_{\text{def}} \Pi\sigma \square \theta\sigma \).

- \( (P\square C\square M\square H\square S_1\square \ldots \square S_i\square \ldots \square S_n)@S_i:\sigma=_{\text{def}}(P\sigma\square C\sigma\square M\sigma\square H\sigma\square S_1\sigma\square \ldots \square S_i \upharpoonright \sigma \square \ldots \square S_n\sigma) \), where \( S_i \upharpoonright \sigma \equiv (\Pi_i \square \theta_i) \upharpoonright \sigma =_{\text{def}} \Pi_i\sigma \square \theta_i\sigma \).

4.2 A Lazy Narrowing Calculus for Cooperative Goal Solving

The Cooperative Constrained Lazy Narrowing Calculus \( \text{CCLNC}(C) \) presented in this section generalizes [2] to cooperative goal solving in \( \text{CFLP}(C) \) for any coordination domain \( C \) and has been proved fully sound w.r.t. \( \text{CRWL}(C) \) semantics, as shown in Subsection 4.4. Moreover, projections (as understood in this paper and [8]) can operate over the constraints included in the constraint stores of the current goal, while the propagations used in [2] can only operate over constraints in the \( C \) part of the current goal, that are not yet placed in any particular store. Due to this difference, projections are computationally more powerful and more difficult to implement than propagations.

As in the case of related calculi, \( \text{CCLNC}(C) \) is based on goal transformation rules intended to transform a given initial goal into solved form. The presentation below distinguishes two kinds of goal transformation rules: rules for constrained lazy narrowing with sharing, relying on the productions (these rules are easily adapted from [10,15]; see Table 3), and new rules for cooperative constraint solving, relying
DC Decomposition
\[ \exists \overline{U}. \ h \overline{\tau}_m \rightarrow h \overline{\tau}_m, \ P \square C \square M \triangleleft H \triangleleft S_1 \square \ldots \square S_n \vdash_{\text{DC}} \exists \overline{U}. \overline{c}_m \rightarrow \overline{t}_m, \ P \square C \square M \triangleleft H \triangleleft S_1 \square \ldots \square S_n \]

CF Conflict Failure
\[ \exists \overline{U}. \ e \rightarrow t, \ P \boxdot C \square M \square H \square S_1 \square \ldots \square S_n \vdash_{\text{CF}} \]

if \( e \) is rigid and passive, \( t \not\in \forall \text{ar}, e \) and \( t \) have conflicting roots.

SP Simple Production
\[ \exists \overline{U}. \ s \rightarrow t, \ P \boxdot C \square M \square H \square S_1 \square \ldots \square S_n \vdash_{\text{SP}} \exists \overline{U}. \ (P \boxdot C \square M \square H \square S_1 \square \ldots \square S_n)@H \sigma \]

if \( s \equiv X \in \forall \text{ar}, t \not\in \forall \text{ar}, \sigma = \{ X \rightarrow t \} \) or \( s \in \text{Pat}_P, t \equiv X \in \forall \text{ar}, \sigma = \{ X \rightarrow s \}; \overline{U} \equiv \overline{U} \setminus \{ X \} \).

IM Imitation
\[ \exists X, \overline{U}. \ h \overline{\tau}_m \rightarrow X, \ P \triangleleft C \triangleleft M \triangleleft H \triangleleft S_1 \triangleleft \ldots \triangleleft S_n \vdash_{\text{IM}} \exists \overline{X}, \overline{U}. \overline{e}_m \rightarrow \overline{X}_m, \ P \triangleleft C \triangleleft M \triangleleft H \triangleleft S_1 \triangleleft \ldots \triangleleft S_n \sigma \]

if \( h \overline{\tau}_m \not\in \text{Pat}_P \) is passive, \( X \) is a demanded variable, \( \sigma = \{ X \rightarrow h \overline{X}_m \} \), and \( \overline{X}_m \) are new variables.

EL Elimination
\[ \exists X, \overline{U}. e \rightarrow X, \ P \boxdot C \boxdot M \boxdot H \boxdot S_1 \boxdot \ldots \boxdot S_n \vdash_{\text{EL}} \exists \overline{U}. \ P \boxdot C \boxdot M \boxdot H \boxdot S_1 \boxdot \ldots \boxdot S_n \]

if \( X \) does not occur in the rest of the goal.

DF Defined Function
\[ \exists \overline{U}. \ f \overline{e}_n \overline{\pi}_k \rightarrow t, \ P \square C \square M \square H \square S_1 \square \ldots \square S_n \vdash_{\text{DF}} \]

\[ \exists X, \overline{U}, \overline{e}_m \rightarrow \overline{t}_m, \ r \rightarrow X, \overline{X} \overline{\pi}_k \rightarrow t, \ P \square C', \ C \square M \square H \square S_1 \square \ldots \square S_n \]

if \( f \in \text{DF}^2_{\Sigma} \) \((k \geq 0)\), \( t \not\in \forall \text{ar} \) or \( t \) is a demanded variable and \( R : f \overline{\tau}_n = r \leftrightarrow C' \) is a fresh variant of a rule in \( P \), with \( \overline{Y} = \text{var}(R) \) and \( X \) are new variables (if \( k = 0 \) we can omit \( X \)).

PC Place Constraint
\[ \exists \overline{U}. \ p \overline{e}_n \rightarrow t, \ P \triangleleft C \triangleleft M \triangleleft H \triangleleft S_1 \triangleleft \ldots \triangleleft S_n \vdash_{\text{PC}} \exists \overline{U}. \ P \triangleleft p \overline{e}_n \rightarrow \overline{t}_n, \ C \triangleleft M \triangleleft H \triangleleft S_1 \triangleleft \ldots \triangleleft S_n \]

if \( p \in \text{PF}^\sharp_{\Pi} \), \( t \not\in \forall \text{ar} \) or \( t \) is a demanded variable.

FC Flatten Constraint
\[ \exists \overline{U}. \ P \square p \overline{e}_n \rightarrow ! t, \ C \square M \square H \square S_1 \square \ldots \square S_n \vdash_{\text{FC}} \exists \overline{U}. \ P \triangleleft p \overline{e}_n \rightarrow ! t, \ C \square M \square H \square S_1 \square \ldots \square S_n \]

if some \( e_i \not\in \text{Pat}_P, \overline{e}_m \) are those \( e_i \) which are not patterns, \( \overline{V}_m \) are new variables, \( p \overline{e}_n \) is obtained from \( p \overline{e}_n \) by replacing each \( e_i \) which is not a pattern by \( V_i \).

SC Submit Constraints
\[ \exists \overline{U}. \ P \triangleleft p \overline{e}_n \rightarrow ! t, \ C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_n \vdash_{\text{SC}} \exists \overline{U}. \ P \triangleleft C \triangleleft M' \triangleleft H' \triangleleft S_1 \triangleleft \ldots \triangleleft S_i \triangleleft \ldots \triangleleft S_n \]

If \( \text{SB} \) cannot be used to set new bridges, and one of the following cases applies:

- If \( p \overline{e}_n \rightarrow ! t \) is a bridge \( t_1 \equiv t_2 \) then \( M' \equiv (t_1 \equiv t_2 \land M), H' \equiv H, \text{ and } S'_i \equiv S_i \).
- If \( p \overline{e}_n \rightarrow ! t \) is \( \text{seq} \ t_1 \ t_2 \rightarrow ! t \) then \( M' \equiv M, H' \equiv (\text{seq} t_1 \ t_2 \rightarrow ! t \land \Pi \square \theta), \text{ and } S'_i \equiv S_i \).
- If \( p \overline{e}_n \rightarrow ! t \) is a primitive constraint \( \pi \in \text{PCon}(D_i) \) then \( M' \equiv M, H' \equiv H, \text{ and } S'_i \equiv (\pi \land \Pi \square \theta) \).

Table 3

Rules for Constrained Lazy Narrowing

on bridges and projections. The following two rules describe the creation of new bridge constraints stored in \( M \) with the aim of enabling projections, and the actual projection of constraints via bridges between any pair of constraint stores \( S_i \) and \( S_j \) \((1 \leq i, j \leq n)\) corresponding to comparable domains \( D_i \) and \( D_j \).
SB Set Bridges
\[ \exists \overline{U}. \ P \square \pi, \ C \square M \square H \square S_1 \square \ldots \square S_n \models_{SB} \exists \overline{V}, \ \overline{U}. \ P \square \pi, \ C \square M', \ M \square H \square S_1 \square \ldots \square S_n \]
If \( \pi \in APCon(D_i) \), \( M' \equiv \text{bridges}_{D_i \rightarrow D_j}(\pi, M) \neq \emptyset \), and \( \overline{V} = \text{var}(M') \setminus \text{var}(M) \) are the new variables occurring in the new bridge constraints.

PR Projection
\[ \exists \overline{U}. \ P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_j \square \ldots \square S_n \models_{PR} \exists \overline{V'}, \ \overline{U}. \ P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_j' \square \ldots \square S_n \]
Where \( S_i \equiv (\pi \land \Pi_i \square \theta_i) \) is the \( D_i \)-store, \( S_j \equiv (\Pi_j \square \theta_j) \) is the \( D_j \)-store, and \( (\pi, M) \models_{\text{projections}_{D_i \rightarrow D_j}} \exists \overline{V'}, \Pi' \), with \( S_j' \equiv (\Pi' \land \Pi_j \square \theta_j) \).

### Table 4

<table>
<thead>
<tr>
<th>MS M-Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists \overline{U}.P \square C \square X \equiv_{d_i,d_j} u', \ M \square H \square S_1 \square \ldots \square S_n \models_{MS_1} \exists \overline{U'}.(P \square C \square M \square H \square S_1 \square \ldots \square S_n)@S_i \sigma )</td>
</tr>
<tr>
<td>If ( X \notin \text{pvar}(P) ), ( u' \in U'<em>{d_i,j} ), ( \sigma = {X \mapsto u} ) with ( u \in U</em>{d_i} ) such that ( \text{equiv}^{M}_{d_i,d_j} u \ u' \rightarrow \text{true} ) and ( \overline{U'} = \overline{U} \setminus {X} ).</td>
</tr>
<tr>
<td>( \exists \overline{U}.P \square C \square u \equiv_{d_i,d_j} X, \ M \square H \square S_1 \square \ldots \square S_n \models_{MS_2} \exists \overline{U'}.(P \square C \square M \square H \square S_1 \square \ldots \square S_n)@S_j \sigma )</td>
</tr>
<tr>
<td>If ( X \notin \text{pvar}(P) ), ( u \in U'<em>{d_i} ), ( \sigma = {X \mapsto u'} ) with ( u' \in U'</em>{d_j} ) such that ( \text{equiv}^{M}_{d_i,d_j} u \ u' \rightarrow \text{true} ) and ( \overline{U'} = \overline{U} \setminus {X} ).</td>
</tr>
<tr>
<td>( \exists \overline{U}.P \square C \square u \equiv_{d_i,d_j} u', \ M \square H \square S_1 \square \ldots \square S_n \models_{MS_3} \exists \overline{U'}.P \square C \square M \square H \square S_1 \square \ldots \square S_n )</td>
</tr>
<tr>
<td>If ( u \in U'<em>{d_i} ), ( u' \in U'</em>{d_j} ), and ( \text{equiv}^{M}_{d_i,d_j} u \ u' \rightarrow \text{true} ) .</td>
</tr>
<tr>
<td>( \exists \overline{U}.P \square C \square u \equiv_{d_i,d_j} u', \ M \square H \square S_1 \square \ldots \square S_n \models_{MS_4} \exists \overline{U'}.P \square C \square M \square H \square S_1 \square \ldots \square S_n )</td>
</tr>
<tr>
<td>If ( u \in U'<em>{d_i} ), ( u' \in U'</em>{d_j} ), and ( \text{equiv}^{M}_{d_i,d_j} u \ u' \rightarrow \text{false} ) .</td>
</tr>
</tbody>
</table>

| HS H-Solver |
| \( \exists \overline{U}.P \square C \square M \square H \square S_1 \square \ldots \square S_n \models_{HS} \exists \overline{V'}.U.(P \square C \square M \square (\Pi' \square \sigma') \square S_1 \square \ldots \square S_n)@H \sigma' \) |
| If \( H = (\Pi \square \sigma) \), and the \( H \)-solving step \( \Pi \models_{solve} \exists \overline{V'}.(\Pi' \square \sigma') \) is admissible w.r.t. \( \text{pvar}(P) \). |

| SS S_i-Solver |
| \( \exists \overline{U}.P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_n \models_{SS} \exists \overline{V'}.U.(P \square C \square M \square H \square S_1 \square \ldots \square (\Pi' \square \sigma_i) \square \ldots \square S_n)@S_i \sigma_i' \) |
| If \( S_i = (\Pi_i \square \sigma_i) \), and the \( D_i \)-solving step \( \Pi_i \models_{solve} \exists \overline{V'}.(\Pi' \square \sigma_i') \) is admissible w.r.t. \( \text{pvar}(P) \). |

| SF Solving Failure |
| \( \exists \overline{U}.P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_n \models_{SF} \) |
| If \( K \models_{solve} D \square \), where \( D \) is the constraint domain \( H \) or \( D_i \) (\( 1 \leq i \leq n \)) and \( K \) is the set of constraints included in \( D \)'s store. |

The four rules in Table 4 describe the process of constraint solving by means of the application of a constraint solver over the corresponding stores (\( M, H \) or \( S_i \)). Note that the constraint solving rules impose certain technical conditions to the variable bindings produced by solvers. These conditions are needed for ensuring the admissibility of goals (see [10,15] for more details).
4.3 An Example of Cooperative Goal Solving

In order to illustrate the behavior of CCLNC(C), let us discuss a goal solving example inspired by [8] and involving cooperation among the domains $H$, $FD$ and $R$. We compute all the solved forms from the constraint $rc$ (par RA RB) $\equiv$ 200 and the program rules given in Subsection 4.1. At each goal transformation step, we underline which subgoal is selected. For the sake of readability, we omit explicit quantification of existential variables. See Section 5 for a comparison between the computations below and those sketched in [8].

\[
\begin{align*}
\text{rc(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\text{RC(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\text{RC(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\text{RC(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\text{RC(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\text{RC(par RA RB)} & \equiv 200 \quad \because \quad X7, X7 \leq 300,600,900,1200,2700,3000, \text{ labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\end{align*}
\]

At this point there are four possible continuations of the computation:

\[
\begin{align*}
G_1 & \equiv [300,600,900,1200,2700,3000] \text{, labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
G_2 & \equiv [300,600,900,1200,2700,3000] \text{, labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
G_3 & \equiv [300,600,900,1200,2700,3000] \text{, labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
G_4 & \equiv [300,600,900,1200,2700,3000] \text{, labeling } [] \quad [X7], \quad [X6] \quad RA \rightarrow res C6 \text{ } \square \\
\end{align*}
\]

4.4 Full Soundness of the Cooperative Goal Solving Calculus

This section presents the main semantic result of the paper, namely full soundness of the cooperative goal solving calculus w.r.t. the declarative semantics of CFLP(C), formalized by means of the constraint rewriting logic $CRWL(C)$. We define the notion of solution for an admissible goal $G \equiv \exists U$. P $\Box$ C $\Box M$ $\Box H$ $\Box S_1$ $\Box \ldots$ $\Box S_n$ and a given CFLP(C)-program $P$ as a valuation $\mu \in Val(C)$ such that there
exists some other valuation $\mu' = (\tau) \mu$ fulfilling the following two conditions: $\mu'$ is a solution of $(P \sqcap C)$ (which means, by definition, $P \vdash_{CRWL(C)} (P \sqcap C) \mu'$) and $\mu' \in Sol(M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n)$ (which can be proved equivalent to $\mu' \in Sol(M) \cap Sol(H) \cap Sol(D_1(S_1)) \cap \ldots \cap Sol(D_n(S_n))$). We write $Sol_P(G)$ for the set of all solutions for $G$. It is easy to check that $Sol_P(S) = Sol_C(S)$ for any solved goal $S$.

The following theorem proves that the goal transformation rules preserve the solutions of admissible goals and fail only in case of inconsistent goals. The proof (given in Appendix A) essentially relies on the correctness conditions for solvers and the safety conditions for bridges and projections, as required in Subsections 2.4 and 3.2.

**Theorem 4.1 (Full Soundness)** Assume an admissible goal $G$ not in solved form for a given CFLP($C$)-program $P$. For any $CCLNC(C)$-rule $RL$ applicable to $G$, there exist $l$ admissible goals $G_k$ such that $G \vdash_{RL,P} G_k$ for each $1 \leq k \leq l$ and $Sol_P(G) = \bigcup_{k=1}^l Sol_P(G_k)$. Moreover, the transformation steps fail only in case of inconsistent goals (i.e., if $G \vdash_{RL,P} \square$ then $Sol_P(G) = \emptyset$).

The soundness of the calculus follows easily from Theorem 4.1. It ensures that the solved forms obtained as computed answers for an initial goal using the rules of the cooperative goal solving calculus are indeed semantically valid answers of $G$.

**Corollary 4.2 (Soundness)** Let $G$ an initial admissible goal and $P$ a CFLP($C$)-program such that $G \vdash^*_P S$, where $S$ is a solved goal. Then, $Sol_C(S) \subseteq Sol_P(G)$.

**Proof.** As an obvious consequence of Theorem 4.1, one gets $Sol_P(G') \subseteq Sol_P(G)$ for any $G'$ such that $G \vdash_P G'$. From this, an easy induction shows that $Sol_P(S) \subseteq Sol_P(G)$ holds for each solved form $S$ such that $G \vdash^*_P S$. Since $Sol_P(S) = Sol_C(S)$, the corollary is proved. $\square$

5 Conclusions and Future Work

This paper contributes to the investigation of cooperation among solvers in declarative programming languages. A small survey of related work has been presented in the introduction. We have focused on coordinated goal solving techniques suitable for constraint functional logic languages such as TOY and Curry. Extending the particular proposal given in [2] to a quite general setting, we have presented coordination domains $C$ and a cooperative goal solving calculus $CCLNC(C)$, thus showing that the $CFLP(C)$ instances of the $CFLP$ scheme [11] provide a fully sound formal framework for functional logic programming with cooperating solvers over various constraint domains. The computation model embodied in $CCLNC(C)$ combines lazy narrowing with the coordinated action of various domain specific solvers.

Inspired by [7,8], we have used projection operations for communication among different solvers. As a novelty w.r.t. Hofstedt’s work, projections in our setting are guided by so-called bridge constraints, provided by a mediatorial domain, which

---

7 See [11] for more details about deductions in the $CRWL(D)$ constrained rewriting logic, which works in particular when $D$ is a coordination domain $C$. 

Polymorphic types |
--- |
Monomorphic types (sorts) |

Coordination domain $C$ (with mediatorial domain) |
--- |
Combined computation domain (without mediatorial domain) |

$H$ within $C$ |
--- |
"The FLP Store" |

Placing and solving constraints as independent actions |
--- |
Placing and solving constraints as simultaneous actions (function $tell$) |

Projections guided by bridge constraints (provided by a mediatorial domain) |
--- |
Projections guided by common variables |

Resistors example: Weak projections suffice (solvers can generate alternatives) |
--- |
Resistors example: Strong projections claimed to be necessary |

Declarative programming systems as goal solving calculi on top of solvers for primitive constraints (Motivation: obtaining precise description of goal solving procedures and strong semantic results) |
--- |
Declarative programming systems viewed as solvers (Motivation: modeling the combination of programming languages as combination of solvers)

<table>
<thead>
<tr>
<th>$CFLP(C)$</th>
<th>Petra Hofstedt’s Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polymorphic types</td>
<td>Monomorphic types (sorts)</td>
</tr>
<tr>
<td>Coordination domain $C$ (with mediatorial domain)</td>
<td>Combined computation domain (without mediatorial domain)</td>
</tr>
<tr>
<td>$H$ within $C$</td>
<td>&quot;The FLP Store&quot;</td>
</tr>
<tr>
<td>Placing and solving constraints as independent actions</td>
<td>Placing and solving constraints as simultaneous actions (function $tell$)</td>
</tr>
<tr>
<td>Projections guided by bridge constraints (provided by a mediatorial domain)</td>
<td>Projections guided by common variables</td>
</tr>
<tr>
<td>Resistors example: Weak projections suffice (solvers can generate alternatives)</td>
<td>Resistors example: Strong projections claimed to be necessary</td>
</tr>
<tr>
<td>Declarative programming systems as goal solving calculi on top of solvers for primitive constraints (Motivation: obtaining precise description of goal solving procedures and strong semantic results)</td>
<td>Declarative programming systems viewed as solvers (Motivation: modeling the combination of programming languages as combination of solvers)</td>
</tr>
</tbody>
</table>

Table 5
Comparison to Petra Hofstedt’s Approach

can be used to express equivalences between values of different base types. A comparison between the $CCLNC(C)$ computations for the resistors example shown in Subsection 4.3 above and the computations for the same example given in Section 3.1 of [8] reveals some differences between Hofstedt’s work and our approach, as summarized in Table 5. In particular, note that the $CCLNC(C)$ computations can solve the resistors problem without resorting to the strong projections used for the same example in [8]. In our opinion, weak projections suffice for the cooperation between $FD$ and $R$, since the generation of alternatives can be handled (at least in this particular but typical example) by the solvers.

As future work, we plan to implement cooperative goal solving with bridges and projections for $CFLP(M \oplus H \oplus FD \oplus R)$ in the $TOY$ system, by extending the implementation reported in [2]. As mentioned in Subsection 4.2, this implementation already supports bridges and a particular kind of projections, called propagations. On the other hand, we also plan to investigate completeness results for $CCLNC(C)$. Obviously, the full soundness theorem 4.1 implies completeness under the additional hypothesis of a finite search space. We aim at stronger completeness results that hold under less restrictive hypotheses, like those found in [10,15] and other related papers. Finally, we plan to investigate the behavior of iterated goal solving and projection operations under different strategies, which should be useful both for implemented systems and as a guide for completeness proofs.
A Appendix: Proof of Theorem 4.1

We present in this appendix the proof of Theorem 4.1. For each goal transformation rule $\text{RL}$ of the $CCLNC(C)$ calculus, we prove the equality $\text{Sol}(G) = \bigcup_{k=1}^{l} \text{Sol}(G'_k)$ under the assumption that $G'_k$ ($1 \leq k \leq l$) are the $l$ admissible goals such that $G \vdash_{\text{RL},P} G'_k$. Here, we restrict ourselves to the cases $\text{RL} = \text{SB}$, $\text{RL} = \text{PR}$ and $\text{RL} = \text{MS}$, corresponding to the goal transformation rules which are new w.r.t. \cite{10}. In each case, we assume that $G$ and $G'_k$ ($1 \leq k \leq l$) are exactly as they appear in the presentation of the corresponding rule $\text{RL}$ in Subsection 4.2. Proofs for the other cases follow easily from the results in \cite{10}.
Rule SB:

In this case, $k = 1$. Let us write $G'$ for $G'_1$. Assume that $\mu \in \text{Sol}_P(G')$. There exists $\mu' = \sqrt[\Upsilon]{\emptyset} \mu$ such that $\mu'$ is a solution for the result of dropping the existential prefix $\exists \overline{V}, \overline{U}$ of $G'$. In particular, $\mu' \in \text{Sol}_C(M)$. Since $\overline{V} = \text{var}(M') \setminus \text{var}(M)$ are new variables not occurring in $G$ and the logical conditions occurring in $G$ under the existential prefix $\exists \overline{U}$ are the same as those occurring in $G'$ except for the bridges in $M'$, we conclude that $\mu \in \text{Sol}_P(G)$. This proves $\text{Sol}_P(G) \supseteq \text{Sol}_P(G')$.

Assume now that $\mu \in \text{Sol}_P(G)$. There exists $\mu' = \sqrt[\Upsilon]{\emptyset} \mu$ such that $\mu'$ is a solution for the result of dropping the existential prefix $\exists \overline{U}$ of $G$. In particular, $\mu' \in \text{Sol}_P(\exists \overline{V}. (\pi \wedge M \wedge M'))$ (where $M$ and $M'$ are interpreted as conjunctions). Therefore, there exists $\mu'' = \sqrt[\overline{V}]{\emptyset} \mu'$ such that $\mu'' \in \text{Sol}_C(\pi \wedge M \wedge M')$. Since the variables in $\overline{V}$ are new and did not occur in $G$, we can conclude that $\mu''$ is a solution of all the logical conditions occurring in $G'$ under the existential prefix $\exists \overline{V}, \overline{U}$. Therefore, we can conclude that $\mu \in \text{Sol}_P(G')$. This proves $\text{Sol}_P(G) \subseteq \text{Sol}_P(G')$.

Rule PR:

Assume that $\mu \in \bigcup_{k=1}^{j} \text{Sol}_P(G'_k)$. Then, $\mu \in \text{Sol}_P(G'_k)$ for some $1 \leq k \leq l$, and there exists $\mu' = \sqrt[\overline{V}, \overline{U}]{\emptyset} \mu$ such that $\mu'$ is a solution for the result of dropping the existential prefix $\exists \overline{V}_k, \overline{U}$ of $G'_k$. In particular, $\mu' \in \text{Sol}_P(S'_j) = \text{Sol}_C(\Pi_j \wedge \Pi_j \sqcap \theta_j) \subseteq \text{Sol}_C(\Pi_j \sqcap \theta_j) = \text{Sol}_C(S_j)$. Since $\overline{V} = \text{var}(M') \setminus \text{var}(M)$ are new variables not occurring in $G$, and the logical conditions occurring in $G$ under the existential prefix $\exists \overline{U}$ are the same as those occurring in $G'$ except for $S'_j$, we conclude that $\mu \in \text{Sol}_P(G)$. This proves $\text{Sol}_P(G) \supseteq \bigcup_{k=1}^{j} \text{Sol}_P(G'_k)$.

Assume now that $\mu \in \text{Sol}_P(G)$. There exists $\mu' = \sqrt[\overline{U}]{\emptyset} \mu$ such that $\mu'$ is a solution for the result of dropping the existential prefix $\exists \overline{U}$ of $G$. In particular, $\mu' \in \text{Sol}_P(\exists \overline{V}. (\pi \wedge \Pi' \wedge M))$ (where $M$ and $\Pi'$ are interpreted as conjunctions). Therefore, there exists $\mu'' = \sqrt[\overline{V}]{\emptyset} \mu'$ such that $\mu'' \in \text{Sol}_C(\pi \wedge \Pi' \wedge M)$. Choose a value of $k$ such that $1 \leq k \leq l$ and $G \vdash_{PR,P} G'_k$ using precisely the projection step $(\pi, M) \vdash_{\text{projections}^{\Pi_i \rightarrow \Pi_j}, \overline{V}'} (\pi \wedge \Pi' \wedge M)$. Since the variables in $\overline{V}'$ are new and did not occur in $G$, we can conclude that $\mu''$ is a solution of all the logical conditions occurring in $G'_k$ under the existential prefix $\exists \overline{V}', \overline{U}$. Therefore, we can conclude that $\mu \in \text{Sol}_P(G'_k)$. This proves $\text{Sol}_P(G) \subseteq \bigcup_{k=1}^{j} \text{Sol}_P(G'_k)$.

Rule MS:

We consider the four subcases of this rule one by one.

Case MS$_1$:

In this case $k = 1$. Let us write $G'$ for $G'_1$. Assume that $\mu \in \text{Sol}_P(G')$. There exists $\mu' = \sqrt[\overline{U}]{\emptyset} \mu$ such that $\mu'$ is a solution for the result of dropping the existential
prefix $\exists \overline{U}^\text{f}$ of $G'$. Then $\mu' \in \text{Sol}(\sigma)$ (i.e., $X \mu' = u = u\mu'$) and thus $\sigma \mu' = \mu'$.

Moreover, $\mu' \in \text{Sol}_C(X \#==d_i,d_j u')$ because $X \mu' = u \in \mathcal{U}^{D_i}_{d_i}$, $u' \mu' = u' \in \mathcal{U}^{D_j}_{d_j}$ and $\text{equiv}^M_{d_i,d_j} u u' \rightarrow \text{true}$ by the applicability conditions of $\text{MS}_1$. Therefore, $\mu' \in \text{Sol}_P(P \sigma \square C \sigma \square M \sigma \square H \sigma \square S_1 \sigma \square \ldots \square (\Pi_i \sigma \square \theta_i \sigma) \square \ldots \square S_n \sigma)$. Since $\sigma \mu' = \mu'$, we have $\mu' \in \text{Sol}_P(P \square C \square X \#==d_i,d_j u',M \square H \square S_1 \square \ldots \square S_n)$. Since $S_i \equiv \Pi_i \square \theta_i$ and $\mu' \in \text{Sol}_C(X \#==d_i,d_j u')$, we also have $\mu' \in \text{Sol}_P(P \square C \square X \#==d_i,d_j u',M \square H \square S_1 \square \ldots \square S_n)$. Since $\overline{U}^f = \overline{U}$ if $X \notin \overline{U}$ and $\overline{U}^f = \overline{U} \setminus \{X\}$ otherwise, we deduce that $\mu' = \overline{U} \mu$ and then $\mu \in \text{Sol}_P(G)$. This proves $\text{Sol}_P(G) \supseteq \text{Sol}_P(G')$.

Assume now that $\mu \in \text{Sol}_P(G)$. There exists $\mu' = \overline{U} \mu$ such that $\mu'$ is a solution for the result of dropping the existential prefix $\exists \overline{U}$ of $G$. Then, $\mu' \in \text{Sol}_C(X \#==d_i,d_j u')$, and thus $\text{equiv}^M_{d_i,d_j} X \mu' u' \rightarrow \text{true}$. According to the applicability conditions of $\text{MS}_1$, $X \mu' = u$ with $u \in \mathcal{U}^{D_i}_{d_i}$. Since $\sigma = \{X \mapsto u\}$, it follows that $\sigma \mu' = \mu'$. Then, $\mu' \in \text{Sol}_P(P \square C \square X \#==d_i,d_j u',M \square H \square S_1 \square \ldots \square S_n) \subseteq \text{Sol}_P(P \square C \square M \square H \square S_1 \square \ldots \square S_n)$. Since $\mu' = \sigma \mu'$, we also have $\sigma \mu' \in \text{Sol}_P(P \square C \square M \square H \square S_1 \square \ldots \square S_n)$. Therefore, $\mu' \in \text{Sol}_P((P \square C \square M \square H \square S_1 \square \ldots \square S_n)@S_i \sigma)$. By definition of $\overline{U}^f$, we deduce that $\mu' = \overline{U} \mu$ and then $\mu \in \text{Sol}_P(G')$. This proves $\text{Sol}_P(G) \subseteq \text{Sol}_P(G')$.

Cases $\text{MS}_2$ and $\text{MS}_3$: Analogous to the case $\text{MS}_1$.

Case $\text{MS}_4$: In this case $k = 0$ and we must prove that $\text{Sol}_P(G) = \emptyset$. Indeed, this is by definition of $\text{Sol}_P(G)$, because $\text{equiv}^M_{d_i,d_j} u u' \rightarrow \text{false}$ with $u \in \mathcal{U}^{D_i}_{d_i}$ and $u' \in \mathcal{U}^{D_j}_{d_j}$ by the applicability conditions of $\text{MS}_4$, and then $\text{Sol}_C(u \#==d_i,d_j u') = \emptyset$. $\square$