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

S. Estévez Martín[a,1] A. J. Fernández[b,2]
M.T. Hortalá González[a,1] M. Rodríguez Artalejo[a,1]
R. del Vado Vírseda[a,1]

[a] *Departamento de Sistemas Informáticos y Computación*
Universidad Complutense de Madrid

[b] *Departamento de Lenguajes y Ciencias de la Computación*
Universidad de Málaga

Abstract

The $CFLP$ scheme for Constraint Functional Logic Programming has instances $CFLP(\mathcal{D})$ corresponding to different constraint domains $\mathcal{D}$. In this paper, we propose an amalgamated sum construction for building coordination domains $\mathcal{C}$, suitable to represent the cooperation among several constraint domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ via a mediatorial domain $\mathcal{M}$. Moreover, we present a cooperative goal solving calculus for $CFLP(\mathcal{C})$, based on lazy narrowing, invocation of solvers for the different domains $\mathcal{D}_i$ involved in the coordination domain $\mathcal{C}$, and projection operations for converting $\mathcal{D}_i$ constraints into $\mathcal{D}_j$ constraints with the aid of mediatorial constraints (so-called bridges) supplied by $\mathcal{M}$. 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 $CFLP(\mathcal{C})$. 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 $CFLP(\mathcal{D})$ corresponding to different specific constraint domains $\mathcal{D}$ given as parameters. In spite of the generality of the approach, the use of one fixed domain $\mathcal{D}$ is an important limitation, since many practical problems involve more than one domain.

[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.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

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 *mediatorial 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 $\mathcal{D}$ to its constraint store, say $S_{\mathcal{D}}$; and projection of constraint stores, that consults the contents of a given store $S_{\mathcal{D}}$ 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 mediatorial 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 $\mathcal{TV}ar$ of *type variables*.

Types $\tau \in Type_\Sigma$ have the syntax $\tau ::= \alpha \mid C\ \tau_1 \ldots \tau_n \mid (\tau_1, \ldots, \tau_n) \mid \tau \to \tau'$, where $\alpha \in \mathcal{TV}ar$ and $C \in TC^n$. By convention, $C\ \overline{\tau}_n$ abbreviates $C\ \tau_1 \ldots \tau_n$, "$\to$" associates to the right, $\overline{\tau}_n \to \tau$ abbreviates $\tau_1 \to \cdots \to \tau_n \to \tau$, and the set of type variables occurring in τ is written $\mathcal{TV}ar(\tau)$. A type τ is called *monomorphic* iff $\mathcal{TV}ar(\tau) = \emptyset$, and *polymorphic* otherwise. Types $C\ \overline{\tau}_n$, $(\tau_1, \ldots, \tau_n)$ and $\tau \to \tau'$ are used to represent constructed values, tuples and functions, respectively. A type without any occurrence of "$\to$" is called a *datatype*. Each n-ary $c \in DC^n$ comes with a principal type declaration $c\ ::\ \overline{\tau}_n \to C\ \overline{\alpha}_k$, where $n, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are pairwise different, τ_i are datatypes, and $\mathcal{TV}ar(\tau_i) \subseteq \{\alpha_1, \ldots, \alpha_k\}$ for all $1 \leq i \leq n$. Also, each n-ary $f \in DF^n$ comes with a principal type declaration $f\ ::\ \overline{\tau}_n \to \tau$, where τ_i, τ 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]

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* $\Gamma = \langle BT, PF\rangle$ disjoint from Σ, where BT is a family of *base types* (such as *int* for integer numbers or *real* for real numbers) and PF is a family of *primitive function* symbols, each one with an associated principal type declaration p ::

[3] In concrete programming languages such as $\mathcal{TOY}$ [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.

$\tau_1 \to \ldots \to \tau_n \to \tau$ (shortly, $p :: \overline{\tau}_n \to \tau$), where τ_1, ..., τ_n and τ 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 Γ (in short, Γ-*domain*) is a structure $\mathcal{D}=\langle\{\mathcal{U}_d^{\mathcal{D}}\}_{d\in BT}, \{p^{\mathcal{D}}\}_{p\in PF}\rangle$, where each $d \in BT$ is interpreted as a non-empty set $\mathcal{U}_d^{\mathcal{D}}$ of *base elements* of type d, as e.g., $\mathbb{Z}=\mathcal{U}_{int}^{\mathcal{D}}$ or $\mathbb{R}=\mathcal{U}_{real}^{\mathcal{D}}$; and interpretations $p^{\mathcal{D}}$ of primitive function symbols behave as explained in Subsection 2.3 below.

2.2 Extended Types, Expressions, Patterns and Substitutions over a Domain $\mathcal{D}$

Given a Γ-domain $\mathcal{D}$, extended types $\tau \in Type_{\Sigma,\Gamma}$ over Γ 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 $\mathcal{Var}$ of *data variables* disjoint from $\mathcal{TVar}$, Σ and Γ, *expressions* $e \in Exp_{\mathcal{D}}$ *over* $\mathcal{D}$ have the syntax $e ::= X \mid u \mid h \mid (e\, e_1)$, where $X \in \mathcal{Var}$, $u \in \mathcal{U}^{\mathcal{D}} =_{def} \bigcup_{d\in BT_\Gamma} \mathcal{U}_d^{\mathcal{D}}$, and $h \in DC_\Sigma \cup DF_\Sigma \cup PF_\Gamma$. Note that $(e\, e_1)$ - not to be confused with the pair (e, e_1) - 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 $(e\, e_1 \ldots e_n)$ as $(e\, \overline{e}_n)$. 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 $\mathcal{D}$ are special expressions $t \in Pat_{\mathcal{D}}$ whose syntax is defined as $t ::= X \mid u \mid (c\, \overline{t}_m) \mid (f\, \overline{t}_m) \mid (p\, \overline{t}_m)$, where $X \in \mathcal{Var}$, $u \in \mathcal{U}^{\mathcal{D}}$, $c \in DC_\Sigma^n$ 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 $\mathcal{D}$ is noted $GPat_{\mathcal{D}}$. The following classification of expressions is also useful: $(X\, \overline{e}_m)$, with $X \in \mathcal{Var}$ and $m \geq 0$, is called a *flexible expression*, while $u \in \mathcal{U}^{\mathcal{D}}$ and $(h\, \overline{e}_m)$ with $h \in DC_\Sigma \cup DF_\Sigma \cup PF_\Gamma$ are called *rigid expressions*. Moreover, a rigid expression $(h\, \overline{e}_m)$ is called *active* iff $h \in DF_\Sigma^n \cup PF_\Gamma^n$ and $m \geq n$, and *passive* otherwise.

We also consider *substitutions* σ, $\theta \in Subs_{\mathcal{D}}$ over $\mathcal{D}$ as mappings from variables to patterns, and by convention, we write ε for the identity substitution, $e\sigma$ instead of $\sigma(e)$ for any $e \in Exp_{\mathcal{D}}$, and $\sigma\theta$ for the composition of σ and θ. A substitution σ such that $\sigma\sigma = \sigma$ is called *idempotent*. The *domain* $\mathcal{V}dom(\sigma) \subseteq \mathcal{Var}$ and *variable range* $\mathcal{V}ran(\sigma) \subseteq \mathcal{Var}$ of σ are defined as usual. For any set of variables $\chi \subseteq \mathcal{Var}$ we define the *restriction* $\sigma\,{\restriction_\chi}$ as the substitution σ' such that $\mathcal{V}dom(\sigma') = \chi$ and $\sigma'(X) = \sigma(X)$ for all $X \in \chi$. Given $\chi \subseteq \mathcal{Var}$, we write $\sigma =_\chi \theta$ to indicate that $\sigma\,{\restriction_\chi} = \theta\,{\restriction_\chi}$, and we abbreviate $\sigma =_{\mathcal{V}\setminus\chi} \theta$ as $\sigma =_{\setminus\chi} \theta$. *Type substitutions* mapping type variables to types can be defined analogously. *Monomorphic instances* τ' of a given type τ can be obtained by applying type substitutions to τ.

Finally, we define the *information ordering* $\sqsubseteq_{\mathcal{D}}$ as the least partial ordering over $Exp_{\mathcal{D}}$ such that $\bot \sqsubseteq_{\mathcal{D}} e$ for all $e \in Exp_{\mathcal{D}}$ and $(e\ e_1) \sqsubseteq_{\mathcal{D}} (e'\ e_1')$ whenever $e \sqsubseteq_{\mathcal{D}} e'$ and $e_1 \sqsubseteq_{\mathcal{D}} e_1'$. The information ordering is useful for semantic considerations.

2.3 Interpreting Primitive Function Symbols

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

(i) **Polarity**: For all $\bar{t}_n, \overline{t'}_n, t, t' \in D_\mathcal{D}$, if $p^\mathcal{D}\ \bar{t}_n \to t, \bar{t}_n \sqsubseteq_\mathcal{D} \overline{t'}_n$ and $t \sqsupseteq_\mathcal{D} t'$ then $p^\mathcal{D}\overline{t'}_n \to t'$ (i.e., monotonicity w.r.t. arguments and antimonotonicity w.r.t. result).

(ii) **Radicality**: For all $\bar{t}_n, t \in D_\mathcal{D}$, if $p^\mathcal{D}\bar{t}_n \to t$ then $t = \bot$ or else there is some total $t' \in D_\mathcal{D}$ such that $t' \sqsupseteq_\mathcal{D} t$, and $p^\mathcal{D}\bar{t}_n \to t'$.

(iii) **Well-Typedness**: For all monomorphic $\overline{\tau'}_n, \tau' \in Type_{\Sigma,\Gamma}$ and all $\bar{t}_n, t \in D_\mathcal{D}$, if $\overline{\tau'}_n \to \tau'$ is a monomorphic instance of $\overline{\tau}_n \to \tau$, $\mathcal{D} \vdash_{MT} \bar{t}_n :: \overline{\tau'}_n$ and $p^\mathcal{D}\bar{t}_n \to t$ then $\mathcal{D} \vdash_{MT} t :: \tau'$ (where $\mathcal{D} \vdash_{MT} \bar{t}_n :: \overline{\tau'}_n$ abbreviates $\mathcal{D} \vdash_{MT} t_1 :: \tau'_1, \ldots, \mathcal{D} \vdash_{MT} t_n :: \tau'_n$).

Type judgements of the form $\mathcal{D} \vdash_{MT} t :: \tau'$ as used in item (iii) above mean that τ' 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 Γ-domain $\mathcal{D}$ 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), $\blacklozenge$ (standing for falsity), or $p\ \bar{e}_n \to !t$ with $p \in PF^n_\Gamma$, $\bar{e}_n \in Exp_\mathcal{D}$ and $t \in Pat_\mathcal{D}$ total. *Atomic primitive constraints* have the form $\Diamond$, $\blacklozenge$ or $p\ \bar{t}_n \to !t$ with $\bar{t}_n \in Pat_\mathcal{D}$. In the sequel, the set of all primitive constraints (resp. atomic primitive constraints) over $\mathcal{D}$ is noted $PCon(\mathcal{D})$, (resp. $APCon(\mathcal{D})$). 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 *seq* $e_1 e_2 \to !t$ (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.
- $\mathcal{FD}$, with specific base type *int*, which supports *finite domain* constraints over $\mathcal{U}^{\mathcal{FD}}_{int} = \mathbb{Z}$ and the primitive functions described in [3] and summarized in Table 1.
- $\mathcal{R}$, with specific base type *real*, which supports *real arithmetic* constraints over $\mathcal{U}^{\mathcal{R}}_{real} = \mathbb{R}$ and the primitive functions described in[11] and summarized in Table 2.

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

For any constraint domain $\mathcal{D}$ we postulate a *constraint solver* given as a function $solve^\mathcal{D}$ such that for any finite $\Pi \subseteq APCon(\mathcal{D})$, $solve^\mathcal{D}(\Pi)$ returns a finite disjunction $\bigvee_{j=1}^k \exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j)$ fulfilling the following correctness conditions:
• For all $1 \leq j \leq k$: $\overline{Y}_j$ are new variables, $\Pi_j \subseteq APCon(\mathcal{D})$ finite, σ_j idempotent substitution such that $\mathcal{V}dom(\sigma_j) \subseteq var(\Pi)$, $\mathcal{V}ran(\sigma_j) \subseteq \overline{Y}_j$, and $solve^\mathcal{D}(\Pi_j) = \Pi_j \sqcap \varepsilon$.
• $Sol_\mathcal{D}(\Pi) = \bigcup_{j=1}^k Sol_\mathcal{D}(\exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j))$ (where $\sqcap$ is interpreted as conjunction).
Π is called a *solved form* iff $solve^\mathcal{D}(\Pi) = \Pi \sqcap \varepsilon$. In the sequel, we will use the following notations:
• $\Pi \Vdash_{solve^\mathcal{D}} \exists \overline{Y'}.\ (\Pi' \sqcap \sigma')$ to indicate that $\exists \overline{Y'}.\ (\Pi' \sqcap \sigma')$ is $\exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j)$ for some $1 \leq j \leq k$ (successful solving step).
• $\Pi \Vdash_{solve^\mathcal{D}} \blacklozenge$ to indicate that $k = 0$ (failing solving step; in this case, $Sol_\mathcal{D}(\Pi) = \emptyset$).

A solving step $\Pi \Vdash_{solve^{\mathcal{D}}} \exists\overline{Y'}.\ (\Pi' \square \sigma')$ is called *admissible* w.r.t. a set of variables $\overline{U}$ iff the two following conditions hold:
• $\overline{U}\sigma'$ is a set of pairwise variable-disjoint linear patterns.
• Either $\overline{U} \cap var(\Pi') = \emptyset$ or else some variable in $\overline{U}$ is demanded by Π'.
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 $\mathcal{C}$ built from various domains $\mathcal{D}_i$, intended to cooperate, and a mediatorial domain $\mathcal{M}$, which supplies special communication constraints called *bridges*. Instances $CFLP(\mathcal{C})$, where $\mathcal{C}$ is a coordination domain, provide a declarative semantic framework for cooperative $CFLP$ programming and goal solving.

3.1 Mediatorial and Coordination Domains

Assume a Γ-domain $\mathcal{D}$ and a Γ'-domain $\mathcal{D}'$ with specific signatures $\Gamma{=}\langle BT, PF\rangle$ and $\Gamma'{=}\langle BT', PF'\rangle$. $\mathcal{D}$ and $\mathcal{D}'$ are called *joinable* iff $PF{\cap}PF'{=}\emptyset$ and $\mathcal{U}_d^{\mathcal{D}}{=}\mathcal{U}_d^{\mathcal{D}'}$ 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 $d{\in}BT$, $\mathcal{U}_d^{\mathcal{D}''}{=}\mathcal{U}_d^{\mathcal{D}}$, and for all $d{\in}BT'$, $\mathcal{U}_d^{\mathcal{D}''}{=}\mathcal{U}_d^{\mathcal{D}'}$.
• For all $p{\in}PF$ and all $\bar{t}_n, t{\in}D_{\mathcal{D}''}$: $p^{\mathcal{D}''}\bar{t}_n{\to}t \Leftrightarrow_{def}$ either $\bar{t}_n{\in}D_{\mathcal{D}}$, $t{\in}D_{\mathcal{D}}$ and $p^{\mathcal{D}}\bar{t}_n{\to}t$, or else $t{=}\bot$.
• For all $p{\in}PF'$ and all $\bar{t}_n, t{\in}D_{\mathcal{D}''}$: $p^{\mathcal{D}''}\bar{t}_n{\to}t \Leftrightarrow_{def}$ either $\bar{t}_n{\in}D_{\mathcal{D}'}$, $t{\in}D_{\mathcal{D}'}$ and $p^{\mathcal{D}'}\bar{t}_n{\to}t$, 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
• $BT_0 \subseteq \bigcup_{i=1}^{n} BT_i$, 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 $equiv_{d_i,d_j} :: d_i \to d_j \to bool$ and there is an injective partial mapping $inj_{d_i,d_j} :: \mathcal{U}_{d_i}^{\mathcal{D}_i} {-\!\!\to} \mathcal{U}_{d_j}^{\mathcal{D}_j}$ such that, for all $t_1, t_2, t \in D_{\mathcal{M}}$: $equiv_{d_i,d_j}^{\mathcal{M}}\, t_1\ t_2 \to t$ $\Leftrightarrow_{def}$ $t_1 \in dom(inj_{d_i,d_j})$, $t_2 = inj_{d_i,d_j}(t_1)$ and $true \sqsupseteq_{\mathcal{M}} t$, or else $t_1 \in dom(inj_{d_i,d_j})$, $t_2 \in \mathcal{U}_{d_j}^{\mathcal{D}_j}$, $t_2 \neq inj_{d_i,d_j}(t_1)$ and $false \sqsupseteq_{\mathcal{M}} t$, 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 $equiv_{d,d} :: d \to d \to bool$ can be defined if wished, whose interpretation $equiv_{d,d}^{\mathcal{M}}$ is based on the identity function $inj_{d,d} = id : \mathcal{U}_d^{\mathcal{D}_i} \to \mathcal{U}_d^{\mathcal{D}_j}$ ($\mathcal{U}_d^{\mathcal{D}_i} = \mathcal{U}_d^{\mathcal{D}_j}$ due to the joinability requirements). The primitive $equiv_{d,d}$ may be useful for communication purposes in case that $\mathcal{D}_i$ and $\mathcal{D}_j$ 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 $equiv_{d_i,d_j} :: d_i \to d_j \to bool$ 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 $\mathcal{D}_i$ and $\mathcal{D}_j$ are *comparable*.

Assume now n given pairwise joinable domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ with specific signatures $\Gamma_1, \ldots, \Gamma_n$ and a mediatorial domain $\mathcal{M}$ for $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Then, the $n+1$ domains $\mathcal{M}, \mathcal{D}_1, \ldots, \mathcal{D}_n$ are pairwise joinable, and the amalgamated sum $\mathcal{C} = \mathcal{M} \oplus \mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_n$ can be built. In the sequel, we assume that the Herbrand domain $\mathcal{H}$ is taken as one of the $\mathcal{D}_i$, and thus $\mathcal{C} = \mathcal{M} \oplus \mathcal{H} \oplus \mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_n$. Such a $\mathcal{C}$ is called a *coordination domain*, because $CFLP(\mathcal{C})$ supports coordinated $CFLP$ programming, using *bridge constraints* of the form $e_1 \#==_{d_i,d_j} e_2 =_{def} equiv_{d_i,d_j} e_1 e_2 \rightarrow !true$ for communication between $\mathcal{D}_i$ and $\mathcal{D}_j$ (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 M and uses them for enabling cooperation between different solvers. More precisely, bridge constraints of the form $e_1 \#==_{d_i,d_j} e_2$ 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 $\mathcal{D}_j$'s store from the constraints available in $\mathcal{D}_i$'s store and the relevant bridges available in M. This enables each solver to take advantage of the computations performed by other solvers. For every pair i, j such that $\mathcal{D}_i$ and $\mathcal{D}_j$ are comparable, we postulate a projection function $projections^{\mathcal{D}_i \rightarrow \mathcal{D}_j}$ such that for any $\pi \in APCon(\mathcal{D}_i)$ and any finite set M of bridge constraints, $projections^{\mathcal{D}_i \rightarrow \mathcal{D}_j}(\pi, M)$ returns a finite disjunction $\bigvee_{k=1}^{l} \exists \overline{Y}_k.\ \Pi'_k$ fulfilling the following *safety conditions*:

- For all $1 \leq k \leq l$: $\overline{Y}_k$ are new variables, and $\Pi'_k \subseteq APCon(\mathcal{D}_j)$ is finite.
- $Sol_{\mathcal{C}}(\pi \wedge M) \subseteq \bigcup_{k=1}^{l} Sol_{\mathcal{C}}(\exists \overline{Y}_k.\ (\pi \wedge \Pi'_k \wedge M))$ (where M and Π'_k are interpreted as conjunctions).

In the sequel, we use the notation $(\pi, M) \vdash_{projections^{\mathcal{D}_i \rightarrow \mathcal{D}_j}} \exists \overline{Y'}.\ \Pi'$ to indicate that $\exists \overline{Y'}.\ \Pi'$ is $\exists \overline{Y}_k.\ \Pi'_k$ 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. [4] 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 $\mathcal{D}_i$ and $\mathcal{D}_j$ are comparable a function $bridges^{\mathcal{D}_i \rightarrow \mathcal{D}_j}$ such that for any $\pi \in APCon(\mathcal{D}_i)$ and any finite set M of bridge constraints, $bridges^{\mathcal{D}_i \rightarrow \mathcal{D}_j}(\pi, M)$ returns a finite set M' of new bridge constraints involving new variables $\overline{V}$, so that the following *safety condition* holds: $Sol_{\mathcal{C}}(\pi \wedge M) \subseteq Sol_{\mathcal{C}}(\exists \overline{V}.\ (\pi \wedge M \wedge M'))$ (where M and M' are interpreted as conjunctions).

[4] Projections in [8] depend on the set of variables common to the stores of $\mathcal{D}_i$ and $\mathcal{D}_j$. In our $CFLP$ framework, well-typing usually prevents the occurrence of one and the same variable in two different stores.

$\pi \in APCon(\mathcal{FD})$	$bridges^{\mathcal{FD}\to\mathcal{R}}(\pi, M)$	$projections^{\mathcal{FD}\to\mathcal{R}}(\pi, M)$
domain$[X_1,...,X_n]$ a b	$\{X_i\#==RX_i \mid 1\leq i\leq n,\ X_i$ has no bridge in M, RX_i new$\}$	$\{a \leq RX_i,\ RX_i \leq b \mid 1\leq i\leq n,\ (X_i\#==RX_i) \in M\}$
belongs X $[a_1,...,a_n]$	$\{X\#==RX \mid X$ has no bridge in M, RX new$\}$	$\{min(a_1,..,a_n)\leq RX, RX\leq max(a_1,.,a_n) \mid 1\leq i\leq n,\ (X\#==RX) \in M\}$
$t_1\#<t_2$ (analogously $\#<=,\#>,\#=>,\#=$)	$\{X_i\#==RX_i \mid 1\leq i\leq 2,\ t_i$ is a variable X_i with no bridge in M, RX_i new$\}$	$\{t_1^{\mathcal{R}} < t_2^{\mathcal{R}} \mid$ For $1\leq i\leq 2$: either t_i is an integer constant n and $t_i^{\mathcal{R}}$ is n, or else t_i is a variable X_i, $(X_i\#==RX_i) \in M$, and $t_i^{\mathcal{R}}$ is $RX_i\}$
$t_1\#+t_2 \to! t_3$ (analogously $\#-$, $\#*$)	$\{X_i\#==RX_i \mid 1\leq i\leq 3, t_i$ is a variable X_i with no bridge in M, RX_i new$\}$	$\{t_1^{\mathcal{R}} + t_2^{\mathcal{R}} \to! t_3^{\mathcal{R}} \mid$ For $1\leq i\leq 3$: $t_i^{\mathcal{R}}$ is determined as in the previous case$\}$

Table 1
Bridge Constraints and Projections from $\mathcal{FD}$ to $\mathcal{R}$

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

$\pi \in APCon(\mathcal{R})$	$bridges^{\mathcal{R}\to\mathcal{FD}}(\pi, M)$	$projections^{\mathcal{R}\to\mathcal{FD}}(\pi, M)$
$RX <= RY$	$\emptyset$ (no bridges are created)	$\{X\#<=Y \mid (X\#==RX),(Y\#==RY)\in M\}$
$RX <= a$	$\emptyset$ (no bridges are created)	$\{X\#<=\lfloor a\rfloor \mid a\in\mathbb{R},\ (X\#==RX)\in M\}$
$t_1 + t_2 \to! t_3$ (analogously for $-$, $*$)	$\{X\#==RX \mid t_3$ 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 $(X_i\#==RX_i) \in M\}$	$\{t_1^{\mathcal{FD}} \#+ t_2^{\mathcal{FD}} \to! t_3^{\mathcal{FD}} \mid$ For $1\leq i\leq 3$: $t_i^{\mathcal{FD}}$ is determined as in the previous case$\}$

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

4 Coordinated $CFLP$ Programming

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

4.1 Structure of Program Rules and Goals

$CFLP(\mathcal{C})$-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 $f \in$

DF^n_Σ with principal type $\overline{\tau}_n \to \tau$ has the form $f\,\overline{t}_n = r \Leftarrow C$, where $f \in DF^n_\Sigma$, $\overline{t}_n$ is a linear sequence of patterns, r is an expression and C is a finite conjunction $\delta_1, \ldots, \delta_m$ of atomic constraints δ_i for each $1 \leq i \leq m$, possibly including occurrences of defined function symbols. Program rules are required to be well-typed. [5]

As an example for the rest of the paper, we consider the following program fragment adapted from [8] and written in $\mathcal{TOY}$ 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 $X :: int$ and $C :: real$.

```
data resistor = res real | seq resistor resistor | par resistor resistor
type capacity ::  real
rc ::  resistor -> capacity
rc (res C) = C <== X #== C, belongs X [300,600,900,...,2700,3000], labeling [] [X]
rc (seq R1 R2) = rc R1 + rc R2
rc (par R1 R2) = 1/((1/rc R1) + (1/rc R2))
```

In the sequel, we consider $CFLP(\mathcal{C})$-*goals* in the general form $G \equiv \exists\overline{U}.\ P \,\square\, C \,\square\, M \,\square\, H \,\square\, S_1 \,\square \ldots \square\, S_n$, 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 $e_1 \to t_1, \ldots, e_m \to t_m$, where $e_i \in Exp_\mathcal{D}$ and $t_i \in Pat_\mathcal{D}$ for all $1 \leq i \leq m$. [6] The set of *produced variables* of G is defined as the set $pvar(P)$ 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 following forms: $X\#==_{d_i,d_j}X'$ or $u\#==_{d_i,d_j}X'$ or $X\#==_{d_i,d_j}u'$ or $u\#==_{d_i,d_j}u'$, where $1 \leq i, j \leq n$ are such that $\mathcal{D}_i$ and $\mathcal{D}_j$ are comparable, X, X' are variables, $u \in \mathcal{U}_{d_i}^{\mathcal{D}_i}$ and $u' \in \mathcal{U}_{d_j}^{\mathcal{D}_j}$.
- H is the so-called *Herbrand store*, including a finite set Π of atomic primitive $\mathcal{H}$-constraints and an answer substitution θ with variable bindings. We use the notation $(\Pi \,\square\, \theta)$ to represent the store H.
- S_i $(1 \leq i \leq n)$ is a $\mathcal{D}_i$ *constraint store* associated to the domain $\mathcal{D}_i$, including a finite set $\Pi_i \subseteq PCon(\mathcal{D}_i)$ of atomic primitive $\mathcal{D}_i$-constraints and an answer substitution θ_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

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

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

demanded by the $\mathcal{FD}$ constraint $X \#=> 3$, but not demanded by the $\mathcal{H}$ constraint $suc\,X \mathrel{/=} zero$, where suc and $zero$ 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 σ and also adding σ to the corresponding store H or S_i ($1 \leq i \leq n$):
• $(P \square C \square M \square H \square S_1 \square \ldots \square S_n)@_H\sigma =_{def} (P\sigma \square C\sigma \square M\sigma \square\ H \upharpoonright \sigma \square S_1\sigma \square \ldots \square S_n\sigma)$, where $H \upharpoonright \sigma \equiv (\Pi \square \theta) \upharpoonright \sigma =_{def} \Pi\sigma \square \theta\sigma$.
• $(P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_n)@_{S_i}\sigma =_{def} (P\sigma \square C\sigma \square M\sigma \square H\sigma \square S_1\sigma \square \ldots \square S_i \upharpoonright \sigma \square \ldots \square S_n\sigma)$, where $S_i \upharpoonright \sigma \equiv (\Pi_i \square \theta_i) \upharpoonright \sigma =_{def} \Pi_i\sigma \square \theta_i\sigma$.

4.2 A Lazy Narrowing Calculus for Cooperative Goal Solving

The *Cooperative Constrained Lazy Narrowing Calculus* $CCLNC(\mathcal{C})$ presented in this section generalizes [2] to cooperative goal solving in $CFLP(\mathcal{C})$ for any coordination domain $\mathcal{C}$ and has been proved fully sound w.r.t. $CRWL(\mathcal{C})$ 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, $CCLNC(\mathcal{C})$ 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 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 $\mathcal{D}_i$ and $\mathcal{D}_j$.

SB Set Bridges

$$\exists \overline{U}.\ P \square \pi,\ C \square M \square H \square S_1 \square \ldots \square S_n \vdash_{\mathbf{SB}} \exists \overline{V},\ \overline{U}.\ P \square \pi,\ C \square M',\ M \square H \square S_1 \square \ldots \square S_n$$

If $\pi \in APCon(\mathcal{D}_i)$, $M' \equiv bridges^{\mathcal{D}_i \to \mathcal{D}_j}(\pi, M) \neq \emptyset$, and $\overline{V} = var(M') \setminus var(M)$ are the new variables occurring in the new bridge constraints.

PR Projection

$$\exists \overline{U}.\ P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S_j \square \ldots \square S_n \vdash_{\mathbf{PR}} \exists \overline{Y'}, \overline{U}.\ P \square C \square M \square H \square S_1 \square \ldots \square S_i \square \ldots \square S'_j \square \ldots \square S_n$$

Where $S_i \equiv (\pi \wedge \Pi_i \square \theta_i)$ is the $\mathcal{D}_i$-store, $S_j \equiv (\Pi_j \square \theta_j)$ is the $\mathcal{D}_j$-store, and $(\pi, M) \vdash_{projections^{\mathcal{D}_i \to \mathcal{D}_j}} \exists \overline{Y'}.\ \Pi'$, with $S'_j \equiv (\Pi' \wedge \Pi_j \square \theta_j)$.

DC Decomposition

$\exists \overline{U}.\ h\ \overline{e}_m \to h\ \overline{t}_m,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{DC}} \exists \overline{U}.\ \overline{e_m \to t_m},\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n$

CF Conflict Failure $\exists \overline{U}.\ e \to t, P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{CF}} \blacksquare$

if e is rigid and passive, $t \notin \mathcal{V}ar$, e and t have conflicting roots.

SP Simple Production

$\exists \overline{U}.\ s \to t,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{SP}} \exists \overline{U'}.\ (P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n)@_H\sigma$

if $s \equiv X \in \mathcal{V}ar$, $t \notin \mathcal{V}ar$, $\sigma = \{X \mapsto t\}$ or $s \in Pat_{\mathcal{D}}$, $t \equiv X \in \mathcal{V}ar$, $\sigma = \{X \mapsto s\}$; $\overline{U'} \equiv \overline{U} \setminus \{X\}$.

IM Imitation

$\exists X, \overline{U}.\ h\ \overline{e}_m \to X,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \sqcap \vdash_{\mathbf{IM}} \exists \overline{X}_m, \overline{U}.(\overline{e_m \to X_m},\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n)\sigma$

if $h\ \overline{e}_m \notin Pat_{\mathcal{D}}$ is passive, X is a demanded variable, $\sigma = \{X \mapsto h\ \overline{X}_m\}$, and $\overline{X}_m$ are new variables.

EL Elimination $\exists X, \overline{U}.e \to X,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{EL}} \exists \overline{U}.P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n$

if X does not occur in the rest of the goal.

DF Defined Function

$\exists \overline{U}.\ f\ \overline{e}_n \overline{a}_k \to t,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{DF}}$

$\exists X, \overline{Y}, \overline{U}.\ \overline{e_n \to t_n},\ r \to X,\ X\ \overline{a}_k \to t,\ P \sqcap C',\ C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n$

if $f \in DF^n_\Sigma$ $(k \geq 0)$, $t \notin \mathcal{V}ar$ or t is a demanded variable and $R : f\ \overline{t}_n = r \Leftarrow C'$ is a fresh variant of a rule in $\mathcal{P}$, with $\overline{Y} = var(R)$ and X are new variables (if $k = 0$ we can omit X).

PC Place Constraint

$\exists \overline{U}.\ p\ \overline{e}_n \to t,\ P \sqcap C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{PC}} \exists \overline{U}.P \sqcap p\ \overline{e}_n \to !t,\ C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n$

if $p \in PF^n_\Gamma$, $t \notin \mathcal{V}ar$ or t is a demanded variable.

FC Flatten Constraint

$\exists \overline{U}.\ P \sqcap p\ \overline{e}_n \to !\ t,\ C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n \vdash_{\mathbf{FC}}$

$\exists \overline{V}_m, \overline{U}.\ \overline{a_m \to V_m},\ P \sqcap p\ \overline{t}_n \to !\ t,\ C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_n$

if some $e_i \notin Pat_{\mathcal{D}}$, $\overline{a}_m$ are those e_i which are not patterns, $\overline{V}_m$ are new variables, $p\ \overline{t}_n$ is obtained from $p\ \overline{e}_n$ by replacing each e_i which is not a pattern by V_i.

SC Submit Constraints

$\exists \overline{U}.P \sqcap p\ \overline{t}_n \to !\ t,\ C \sqcap M \sqcap H \sqcap S_1 \sqcap \ldots \sqcap S_i \sqcap \ldots S_n \vdash_{\mathbf{SC}} \exists \overline{U}.P \sqcap C \sqcap M' \sqcap H' \sqcap S_1 \sqcap \ldots \sqcap S'_i \sqcap \ldots \sqcap S_n$

If **SB** cannot be used to set new bridges, and one of the following cases applies:

- If $p\ \overline{t}_n \to !\ t$ is a bridge $t_1\ \#\!\!=\!\!=\ t_2$ then $M' \equiv (t_1\ \#\!\!=\!\!=\ t_2 \wedge M)$, $H' \equiv H$, and $S'_i \equiv S_i$.
- If $p\ \overline{t}_n \to !\ t$ is $seq\ t_1\ t_2 \to !\ t$ then $M' \equiv M$, $H' \equiv (seq\ t_1\ t_2 \to !\ t \wedge \Pi \sqcap \theta)$, and $S'_i \equiv S_i$.
- If $p\ \overline{t}_n \to !\ t$ is a primitive constraint $\pi \in PCon(\mathcal{D}_i)$ then $M' \equiv M$, $H' \equiv H$, and $S'_i \equiv (\pi \wedge \Pi_i \sqcap \theta_i)$.

Table 3
Rules for Constrained Lazy Narrowing

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).

MS M-Solver

- $\exists\overline{U}.P\Box C\Box\ X\ \#{=}{=}_{d_i,d_j}u',\ M\Box H\Box S_1\Box\ldots\Box S_n \vdash_{\mathbf{MS_1}} \exists\overline{U'}.(P\Box C\Box M\Box H\Box S_1\Box\ldots\Box S_n)@_{S_i}\sigma$

If $X\notin pvar(P)$, $u'\in\mathcal{U}_{d_j}^{\mathcal{D}_j}$, $\sigma=\{X\mapsto u\}$ with $u\in\mathcal{U}_{d_i}^{\mathcal{D}_i}$ such that $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u'\rightarrow true$ and $\overline{U'}=\overline{U}\setminus\{X\}$.

- $\exists\overline{U}.P\Box C\Box\ u\ \#{=}{=}_{d_i,d_j}X,\ M\ \Box H\Box S_1\Box\ldots\Box S_n \vdash_{\mathbf{MS_2}} \exists\overline{U'}.(P\Box C\Box M\Box H\Box S_1\Box\ldots\Box S_n)@_{S_j}\sigma$

If $X\notin pvar(P)$, $u\in\mathcal{U}_{d_i}^{\mathcal{D}_i}$, $\sigma=\{X\mapsto u'\}$ with $u'\in\mathcal{U}_{d_j}^{\mathcal{D}_j}$ such that $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u'\rightarrow true$ and $\overline{U'}=\overline{U}\setminus\{X\}$.

- $\exists\overline{U}.P\Box C\Box u\ \#{=}{=}_{d_i,d_j}u',\ M\ \Box H\Box S_1\Box\ldots\Box S_n \vdash_{\mathbf{MS_3}} \exists\overline{U}.\ P\Box C\Box M\Box H\Box S_1\Box\ldots\Box S_n$

If $u\in\mathcal{U}_{d_i}^{\mathcal{D}_i}$, $u'\in\mathcal{U}_{d_j}^{\mathcal{D}_j}$, and $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u'\rightarrow true$.

- $\exists\overline{U}.\ P\ \Box\ C\ \Box\ u\ \#{=}{=}_{d_i,d_j}u',\ M\ \Box\ H\ \Box\ S_1\ \Box\ \ldots\ \Box\ S_n \vdash_{\mathbf{MS_4}} \blacksquare$

If $u\in\mathcal{U}_{d_i}^{\mathcal{D}_i}$, $u'\in\mathcal{U}_{d_j}^{\mathcal{D}_j}$, and $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u'\rightarrow false$.

HS H-Solver $\exists\overline{U}.P\Box C\Box M\Box H\Box S_1\Box\ldots\Box S_n \vdash_{\mathbf{HS}} \exists\overline{Y'},\overline{U}.(P\Box C\Box M\Box(\Pi'\Box\sigma)\Box S_1\ \Box\ldots\Box S_n)@_H\sigma'$

If $H=(\Pi\Box\sigma)$, and the $\mathcal{H}$-solving step $\Pi\vdash_{solve^{\mathcal{H}}}\exists\overline{Y'}.\ (\Pi'\Box\sigma')$ is admissible w.r.t. $pvar(P)$.

S$_i$S S_i-Solver

$\exists\overline{U}.P\Box C\Box M\Box H\Box S_1\Box\ldots\ \Box S_i\Box\ldots\Box S_n \vdash_{\mathbf{S_iS}} \exists\overline{Y'},\overline{U}.(P\Box C\Box M\Box H\Box S_1\Box\ldots\ \Box(\Pi_i'\Box\sigma_i)\Box\ldots\Box S_n)@_{S_i}\sigma_i'$

If $S_i=(\Pi_i\Box\sigma_i)$, and the $\mathcal{D}_i$-solving step $\Pi_i\vdash_{solve^{\mathcal{D}_i}}\exists\overline{Y'}.\ (\Pi_i'\Box\sigma_i')$ is admissible w.r.t. $pvar(P)$.

SF Solving Failure $\exists\overline{U}.\ P\ \Box\ C\ \Box\ M\ \Box\ H\ \Box\ S_1\ \Box\ \ldots\ \Box\ S_i\ \Box\ \ldots\ \Box\ S_n \vdash_{\mathbf{SF}} \blacksquare$

If $K\vdash_{solve^{\mathcal{D}}}\blacklozenge$, where $\mathcal{D}$ is the constraint domain $\mathcal{H}$ or $\mathcal{D}_i$ $(1\leq i\leq n)$ and K is the set of constraints included in $\mathcal{D}'$s store.

Table 4
Rules for Constraint Solving

4.3 An Example of Cooperative Goal Solving

In order to illustrate the behavior of $CCLNC(\mathcal{C})$, let us discuss a goal solving example inspired by [8] and involving cooperation among the domains $\mathcal{H}$, $\mathcal{FD}$ and $\mathcal{R}$. We compute all the solved forms from the constraint `rc (par RA RB) == 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].

□ rc(par RA RB)==200 □□□□ $\vdash_{FC}$ rc(par RA RB)→C □ C==200 □□□□ $\vdash_{PC}$
rc(par RA RB)→C□□□□ C==200 □□ $\vdash_{HS}$ rc(par RA RB)→200□□□□□□ $\vdash_{DF_{rc.3},DC,SP^2}$
1/(1/rc(RA)+1/rc(RB))→200 □□□□□□ $\vdash_{PC}$ □ 1/(1/rc(RA)+1/rc(RB))→!200 □□□□□ $\vdash_{FC}$
1/rc(RA)+1/rc(RB)→C1 □ 1/C1→!200 □□□□□ $\vdash_{PC,SC}$
□ 1/rc(RA)+1/rc(RB)→!C1 □□□□□ 1/C1→!200 $\vdash_{FC,PC^2,SC}$
□ 1/rc(RA)→!C2, 1/rc(RB)→!C3 □□□□□ C2+C3→!C1, 1/C1→!200 $\vdash_{FC^2,SC^2}$
rc(RA)→C4,rc(RB)→C5□□□□1/C4→!C2,1/C5→!C3,C2+C3→!C1,1/C1→!200$\vdash_{RS}$
rc(RA)→C4, rc(RB)→C5 □□□□ (1/C4)+(1/C5)==1/200 $\vdash_{DF_{rc.1}}$
RA→res C6, C6→C4, rc(RB)→C5 □ X6#==C6,
belongs X6 [300,600,900,1200,2700,3000], labeling [] [X6]□□□(1/C4)+(1/C5)==1/200$\vdash^2_{SP}$
rc(RB)→C5 □ X6#==C6,belongs X6 [300,..,3000], labeling [] [X6] □ RA↦res C6 □□
(1/C6)+(1/C5)==1/200$\vdash^2_{SC}$
rc(RB)→C5 □□ X6#==C6 □ RA↦res C6 □ belongs X6 [300,..,3000], labeling [] [X6] □
(1/C6)+(1/C5)==1/200$\vdash^*_{DF_{rc.1}}$
□□ X7#==C7,X6#==C6 □ RB↦res C7,RA↦res C6 □ belongs X7 [300,..,3000], labeling [] [X7],
belongs X6 [300,..,3000], labeling [] [X6] □ (1/C6)+(1/C7)==1/200$\vdash^2_{PR^{\mathcal{FD}\rightarrow\mathcal{R}}}$

□□ X7#==C7,X6#==C6 □ RB↦res C7,RA↦res C6 □ belongs X7 [300,..,3000], labeling [] [X7], belongs X6 [300,..,3000], labeling [] [X6] □ 300≤C7,C7≤3000,300≤C6,C6≤3000, (1/C6)+(1/C7)==1/200⊩$_{RS}$

□□ X7#==C7,X6#==C6 □ RB↦res C7,RA↦res C6 □ belongs X7 [300,..,3000], labeling [] [X7], belongs X6 [300,..,3000], labeling [] [X6] □ 300≤C7,C7≤600,300≤C6,C6≤600, (1/C6)+(1/C7)==1/200⊩$^{4}_{PR^{\mathcal{R}\to\mathcal{FD}}}$

□□ X7#==C7,X6#==C6 □ RB↦res C7, RA ↦res C6 □ 300#≤X7, X7#≤600, 300#≤X6,X6#≤600, belongs X7 [300,..,3000], labeling [] [X7], belongs X6 [300,..,3000], labeling [] [X6] □ 300≤C7, C7≤600, 300≤C6, C6≤600, (1/C6)+(1/C7)==1/200 ⊩$_{FS}$

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

$G_1 \equiv$ □□ 300#==C7,300#==C6 □ RB↦res C7,RA↦res C6 □ X7↦300,X6↦300 □ 300≤C7,C7≤600, 300≤C6,C6≤ 600,(1/C6)+(1/C7)==1/200 ⊩$^{2}_{MS}$ □□□ RB↦res 300,RA↦res 300 □□ 300≤300, 300≤600,300≤300,300≤600,(1/300)+(1/300)==1/200⊩$_{RS}$ ■

$G_2 \equiv$ □□ 300#==C7,600#==C6 □ RB↦res C7,RA↦res C6 □ X7↦300,X6↦600□ 300≤C7,C7≤600,300≤C6, C6≤ 600,(1/C6)+(1/C7)==1/200 ⊩$^{2}_{MS}$ □□□ RB↦res 300,RA↦res 600 □□ 300≤300,300≤600, 300≤600,600≤600,(1/600)+(1/300)==1/200 ⊩$_{RS}$ □□□ RB↦res 300,RA↦res 600 □□

$G_3 \equiv$ □□ 600#==C7,300#==C6 □ RB↦res C7,RA↦res C6 □ X7↦600,X6↦300 □ 300≤C7,C7≤600,300≤C6, C6≤600,(1/C6)+(1/C7)==1/200 ⊩$_{MS^2,RS}$ □□□ RB↦res 600, RA↦res 300 □□

$G_4 \equiv$ □□ 600#==C7,600#==C6 □ RB↦res C7,RA↦res C6 □ X7↦600,X6↦600 □ 300≤C7,C7≤600,300≤C6, C6≤600,(1/C6)+(1/C7)==1/200 ⊩$_{MS^2,RS}$ ■

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 $CFLP(\mathcal{C})$, formalized by means of the constraint rewriting logic $CRWL(\mathcal{C})$. We define the notion of *solution* for an admissible goal $G \equiv \exists\overline{U}.\ P \square C \square M \square H \square S_1 \square \ldots \square S_n$ and a given $CFLP(\mathcal{C})$-program $\mathcal{P}$ as a valuation $\mu \in Val(\mathcal{C})$ such that there exists some other valuation $\mu' =_{\setminus\overline{U}} \mu$ fulfilling the following two conditions: μ' is a solution of $(P \square C)$ (which means, by definition, $\mathcal{P} \vdash_{CRWL(\mathcal{C})} (P \square C)\mu'$ [7]) and $\mu' \in Sol_{\mathcal{C}}(M \square H \square S_1 \square \ldots \square S_n)$ (which can be proved equivalent to $\mu' \in Sol_{\mathcal{M}}(M) \cap Sol_{\mathcal{H}}(H) \cap Sol_{\mathcal{D}_1}(S_1) \cap \ldots \cap Sol_{\mathcal{D}_n}(S_n)$). We write $Sol_{\mathcal{P}}(G)$ for the set of all solutions for G. It is easy to check that $Sol_{\mathcal{P}}(S) = Sol_{\mathcal{C}}(S)$ 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 $CFLP(\mathcal{C})$-program $\mathcal{P}$. For any $CCLNC(\mathcal{C})$-rule* **RL** *applicable to G, there exist l admissible goals G_k such that $G \vdash_{\mathbf{RL},\mathcal{P}} G_k$ for each $1 \leq k \leq l$ and $Sol_{\mathcal{P}}(G) = \bigcup_{k=1}^{l} Sol_{\mathcal{P}}(G_k)$. Moreover, the transformation steps fail only in case of inconsistent goals (i.e., if $G \vdash_{\mathbf{RL},\mathcal{P}}$ ■ then $Sol_{\mathcal{P}}(G) = \emptyset$).*

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.

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

Corollary 4.2 (*Soundness*) *Let G an initial admissible goal and $\mathcal{P}$ a $CFLP(\mathcal{C})$-program such that $G \vdash_{\mathcal{P}}^{*} S$, where S is a solved goal. Then, $Sol_{\mathcal{C}}(S) \subseteq Sol_{\mathcal{P}}(G)$.*

Proof. As an obvious consequence of Theorem 4.1, one gets $Sol_{\mathcal{P}}(G') \subseteq Sol_{\mathcal{P}}(G)$ for any G' such that $G \vdash_{\mathcal{P}} G'$. From this, an easy induction shows that $Sol_{\mathcal{P}}(S) \subseteq Sol_{\mathcal{P}}(G)$ holds for each solved form S such that $G \vdash_{\mathcal{P}}^{*} S$. Since $Sol_{\mathcal{P}}(S) = Sol_{\mathcal{C}}(S)$, the corollary is proved. □

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 $\mathcal{TOY}$ and *Curry*. Extending the particular proposal given in [2] to a quite general setting, we have presented coordination domains $\mathcal{C}$ and a cooperative goal solving calculus $CCLNC(\mathcal{C})$, thus showing that the $CFLP(\mathcal{C})$ 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 $CCLNC(\mathcal{C})$ combines lazy narrowing with the coordinated action of various domain specific solvers.

$CFLP(\mathcal{C})$	*Petra Hofstedt's Approach*
Polymorphic types	Monomorphic types (sorts)
Coordination domain $\mathcal{C}$ (with mediatorial domain)	Combined computation domain (without mediatorial domain)
$\mathcal{H}$ within $\mathcal{C}$	"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 5
Comparison to Petra Hofstedt's Approach

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 can be used to express equivalences between values of different base types. A comparison between the $CCLNC(\mathcal{C})$ 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 $CCLNC(\mathcal{C})$ 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 $\mathcal{FD}$ and $\mathcal{R}$, 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 $CFLP(\mathcal{M} \oplus \mathcal{H} \oplus \mathcal{FD} \oplus \mathcal{R})$ in the $\mathcal{TOY}$ 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 $CCLNC(\mathcal{C})$. 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.

References

[1] P. Arenas, F.J. López-Fraguas, M. Rodríguez-Artalejo. *$\mathcal{TOY}$. A Multiparadigm Declarative Language. Version* 2.2.2 (2006), R. Caballero and J. Sánchez (Eds.), Available at `http://toy.sourceforge.net`.

[2] S. Estévez Martín, A.J. Fernández, M.T. Hortalá-González, M. Rodríguez-Artalejo, F. Sáenz-Pérez and R. del Vado-Vírseda. *A Proposal for the Cooperation of Solvers in Constraint Functional Logic Programming*. Proceedings of PROLE'06, 14 pp. To appear in Electronic Notes on Theoretical Computer Science, 2007.

[3] A.J. Fernández, M.T. Hortalá-González, F. Sáenz-Pérez and R. del Vado-Vírseda. *Constraint functional logic programming over finite domains*. To appear in the Journal of Theory and Practice of Logic Programming, volume 7(3), 2006. Available on-line in http://arXiv.org/abs/cs/0601071.

[4] 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`.

[5] L. Granvilliers, E. Monfroy, and F. Benhamou. *Cooperative solvers in constraint programming: a short introduction. ALP Newsletter*, 14(2), May 2001.

[6] M. Hanus. *Curry: an Integrated Functional Logic Language*, Version 0.8.2, March 28, (2006). `http://www-i2.informatik.uni-kiel.de/~curry/`.

[7] P. Hofstedt. *Cooperation and Coordination of Constraint Solvers*. Ph.D. thesis, Shaker Verlag, Aachen, 2001.

[8] P. Hofstedt and P. Pepper. *Integration of Declarative and Constraint Programming*. Theory and Practice of Logic Programming, 2006.

[9] J. Jaffar and M. Maher. *Constraint logic programming: a survey. The Journal of Logic Programming*, 19-20:503–581, 1994.

[10] F.J. López-Fraguas, M. Rodríguez-Artalejo and R. del Vado-Vírseda. *A Lazy Narrowing Calculus for Declarative Constraint Programming*. In Proc. ACM SIGPLAN of Int. Conf. on Principles and Practice of Declarative Programming (PPDP'04), ACM Press, pp. 43–54, 2004.

[11] F.J. López-Fraguas, M. Rodríguez-Artalejo and R. del Vado-Vírseda. *A New Generic Scheme for Functional Logic Programming with Constraints*. To appear in the Journal of *Higher-Order and Symbolic Computation*, volume 20(1/2), 2007.

[12] M. Marin. *Functional Logic Programming with Distributed Constraint Solving*. PhD thesis, Johannes Kepler Universität Linz, 2000.

[13] M. Marin, T. Ida, and W. Schreiner. *CFLP: a Mathematica Implementation of a Distributed Constraint Solving System*. In *Third International Mathematica Symposium (IMS'99)*, Hagenberg, Austria, August 23–25 1999. Computational Mechanics Publications, WIT Press, Southampton, UK.

[14] E. Monfroy. *Solver collaboration for constraint logic programming*. PhD thesis, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine, November 1996.

[15] R. del Vado-Vírseda. *Declarative Constraint Programming with Definitional Trees*. In Proc. of the 5th International Conference on *Frontiers of Combining Systems* (FroCoS'05), volume 3717 of LNCS, pages 184-199, Springer, 2005.

A Appendix: Proof of Theorem 4.1

We present in this appendix the proof of Theorem 4.1. For each goal transformation rule **RL** of the $CCLNC(\mathcal{C})$ calculus, we prove the equality $Sol_{\mathcal{P}}(G) = \bigcup_{k=1}^{l} Sol_{\mathcal{P}}(G'_k)$ under the assumption that G'_k $(1 \leq k \leq l)$ are the l admissible goals such that $G \vdash_{\mathbf{RL},\mathcal{P}} G'_k$. Here, we restrict ourselves to the cases **RL** = **SB**, **RL** = **PR** and **RL** = **MS**, corresponding to the goal transformation rules which are new w.r.t. [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 **RL** in Subsection 4.2. Proofs for the other cases follow easily from the results in [10].

Rule **SB**:

In this case, $k = 1$. Let us write G' for G'_1. Assume that $\mu \in Sol_{\mathcal{P}}(G')$. There exists $\mu' =_{\setminus \overline{V},\overline{U}} \mu$ such that μ' is a solution for the result of dropping the existential prefix $\exists \overline{V}, \overline{U}$ of G'. In particular, $\mu' \in Sol_{\mathcal{C}}(M)$. Since $\overline{V} = var(M') \setminus var(M)$ 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 $\mu \in Sol_{\mathcal{P}}(G)$. This proves $Sol_{\mathcal{P}}(G) \supseteq Sol_{\mathcal{P}}(G')$.

Assume now that $\mu \in Sol_{\mathcal{P}}(G)$. There exists $\mu' =_{\setminus \overline{U}} \mu$ such that μ' is a solution for the result of dropping the existential prefix $\exists \overline{U}$ of G. In particular, $\mu' \in Sol_{\mathcal{D}_i}(\pi)$ and $\mu' \in Sol_{\mathcal{C}}(M)$. By the *safety condition* postulated for $bridges^{\mathcal{D}_i \to \mathcal{D}_j}$ (see Subsection 3.2), it follows that $\mu' \in Sol_{\mathcal{C}}(\exists \overline{V}.\ (\pi \wedge M \wedge M'))$ (where M and M' are interpreted as conjunctions). Therefore, there exists $\mu'' =_{\setminus \overline{V}} \mu'$ such that $\mu'' \in Sol_{\mathcal{C}}(\pi \wedge M \wedge M')$. Since the variables in $\overline{V}$ are new and did not occur in G, we can conclude that μ'' 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 $\mu \in Sol_{\mathcal{P}}(G')$. This proves $Sol_{\mathcal{P}}(G) \subseteq Sol_{\mathcal{P}}(G')$.

Rule **PR**:

Assume that $\mu \in \bigcup_{k=1}^{l} Sol_{\mathcal{P}}(G'_k)$. Then, $\mu \in Sol_{\mathcal{P}}(G'_k)$ for some $1 \leq k \leq l$, and there exists $\mu' =_{\setminus \overline{V}_k,\overline{U}} \mu$ such that μ' is a solution for the result of dropping the existential prefix $\exists \overline{V}_k, \overline{U}$ of G'_k. In particular, $\mu' \in Sol_{\mathcal{C}}(S'_j) = Sol_{\mathcal{C}}(\Pi' \wedge \Pi_j \square \theta_j) \subseteq Sol_{\mathcal{C}}(\Pi_j \square \theta_j) = Sol_{\mathcal{C}}(S_j)$. Since $\overline{V} = var(M') \setminus var(M)$ 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 $\mu \in Sol_{\mathcal{P}}(G)$. This proves $Sol_{\mathcal{P}}(G) \supseteq \bigcup_{k=1}^{l} Sol_{\mathcal{P}}(G'_k)$.

Assume now that $\mu \in Sol_{\mathcal{P}}(G)$. There exists $\mu' =_{\setminus \overline{U}} \mu$ such that μ' is a solution for the result of dropping the existential prefix $\exists \overline{U}$ of G. In particular, $\mu' \in Sol_{\mathcal{C}}(\pi)$

and $\mu' \in Sol_{\mathcal{C}}(M)$. By the *safety conditions* postulated for $projections^{\mathcal{D}_i \to \mathcal{D}_j}$ (see Subsection 3.2), there must be some projection step $(\pi, M) \vdash_{projections^{\mathcal{D}_i \to \mathcal{D}_j}} \exists \overline{Y'}.\Pi'$ such that $\mu' \in Sol_{\mathcal{C}}(\exists \overline{Y'}.\ (\pi \wedge \Pi' \wedge M))$ (where M and Π' are interpreted as conjunctions). Therefore, there exists $\mu'' =_{\setminus \overline{Y'}} \mu'$ such that $\mu'' \in Sol_{\mathcal{C}}(\pi \wedge \Pi' \wedge M)$. Choose a value of k such that $1 \leq k \leq l$ and $G \vdash_{\mathbf{PR},\mathcal{P}} G'_k$ using precisely the projection step $(\pi, M) \vdash_{projections^{\mathcal{D}_i \to \mathcal{D}_j}} \exists \overline{Y'}.\ \Pi'$. Since the variables in $\overline{Y'}$ are new and did not occur in G, we can conclude that μ'' is a solution of all the logical conditions occurring in G'_k under the existential prefix $\exists \overline{Y'}, \overline{U}$. Therefore, we can conclude that $\mu \in Sol_{\mathcal{P}}(G'_k)$. This proves $Sol_{\mathcal{P}}(G) \subseteq \bigcup_{k=1}^{l} Sol_{\mathcal{P}}(G'_k)$.

Rule **MS**: We consider the four subcases of this rule one by one.

Case $\mathbf{MS}_1$:

In this case $k = 1$. Let us write G' for G'_1. Assume that $\mu \in Sol_{\mathcal{P}}(G')$. There exists $\mu' =_{\setminus \overline{U'}} \mu$ such that μ' is a solution for the result of dropping the existential prefix $\exists \overline{U'}$ of G'. Then $\mu' \in Sol(\sigma)$ (i.e., $X\mu' = u = u\mu'$) and thus $\sigma\mu' = \mu'$. Moreover, $\mu' \in Sol_{\mathcal{C}}(X \ \#==_{d_i,d_j} \ u')$ because $X\mu' = u \in \mathcal{U}_{d_i}^{\mathcal{D}_i}$, $u'\mu' = u' \in \mathcal{U}_{d_j}^{\mathcal{D}_j}$ and $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u' \to true$ by the applicability conditions of $\mathbf{MS}_1$. Therefore, $\mu' \in Sol_{\mathcal{P}}(P\sigma \ \square \ C\sigma \ \square \ M\sigma \ \square \ H\sigma \ \square \ S_1\sigma \ \square \ \ldots \ \square \ (\Pi_i\sigma \ \square \ \theta_i\sigma) \ \square \ \ldots \ \square \ S_n\sigma)$. Since $\sigma\mu' = \mu'$, we have $\mu' \in Sol_{\mathcal{P}}(P \ \square \ C \ \square \ M \ \square \ H \ \square \ S_1 \ \square \ \ldots \ \square \ (\Pi_i \ \square \ \theta_i) \ \square \ \ldots \ \square \ S_n)$. Since $S_i \equiv \Pi_i \ \square \ \theta_i$ and $\mu' \in Sol_{\mathcal{C}}(X \ \#==_{d_i,d_j} \ u')$, we also have $\mu' \in Sol_{\mathcal{P}}(P \ \square \ C \ \square \ X \ \#==_{d_i,d_j} \ u', M \ \square \ H \ \square \ S_1 \ \square \ \ldots \ \square \ S_n)$. Since $\overline{U'} = \overline{U}$ if $X \notin \overline{U}$ and $\overline{U'} = \overline{U} \setminus \{X\}$ otherwise, we deduce that $\mu' =_{\setminus \overline{U}} \mu$ and then $\mu \in Sol_{\mathcal{P}}(G)$. This proves $Sol_{\mathcal{P}}(G) \supseteq Sol_{\mathcal{P}}(G')$.

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

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

Case $\mathbf{MS}_4$:

In this case $k = 0$ and we must prove that $Sol_{\mathcal{P}}(G) = \emptyset$. Indeed, this is by definition of $Sol_{\mathcal{P}}(G)$, because $equiv_{d_i,d_j}^{\mathcal{M}}\ u\ u' \to false$ with $u \in \mathcal{U}_{d_i}^{\mathcal{D}_i}$ and $u' \in \mathcal{U}_{d_j}^{\mathcal{D}_j}$ by the applicability conditions of $\mathbf{MS}_4$, and then $Sol_{\mathcal{C}}(u \ \#==_{d_i,d_j} \ u') = \emptyset$. □