TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

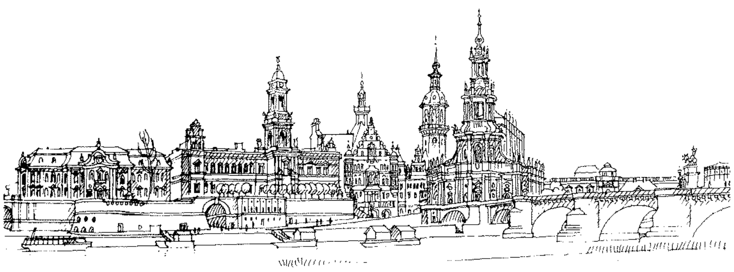
TUD / FI 98 / 08 - August 1998

P. Hofstedt
Grundlagen der Programmierung
Institut für Softwaretechnik I

Technische Berichte
Technical Reports

ISSN 1430-211X

An Architecture for the Combination of Constraint Solvers



Technische Universität Dresden
Fakultät Informatik
D-01062 Dresden
Germany

URL: http://www.inf.tu-dresden.de/

An Architecture for the Combination of Constraint Solvers

Petra Hofstedt

Foundations of Programming,
Department of Computer Science, Dresden University of Technology,
hofstedt@inf.tu-dresden.de

Abstract. In this paper, an architecture for the combination of different constraint solvers with the help of projections is proposed.
By means of the definition of an interface and restricting properties of the operations of constraint solvers we are able to define a mechanism for the open and consistent combination of constraint systems. We enable the use of a functional logic language as constraint solver for constraints over functional expressions in the overall system.
The syntax of a language which allows the specification of mixed constraints of different constraint domains and its operational semantics are introduced.
We compare our approach with BALI, an environment for designing and executing constraint solver combinations.

1 Introduction

The paradigm of constraint programming ([7]) offers efficient mechanisms to handle constraints of various constraint domains. However, one constraint solver is only able to handle constraints of one particular constraint domain. It is desirable and advantageous to combine several constraint systems and the associated constraint solving techniques because this combination makes it possible to solve problems that none of the single solvers can handle alone. Thus, our aim is to integrate different constraint systems and to combine different constraint solvers. We want to find a method to get a consistent and open combination of constraint solvers. The combination is open in the sense that, whenever a new constraint system with associated constraint solver (satisfying certain properties) arises, it can be easily incorporated into the whole system, independently of its constraint domain and the language in which the constraint solver is implemented.

Functional logic languages ([5]) offer features from functional programming as well as from logic programming. Using a functional logic language the user can state a problem in a very comfortable way. Therefore we want to build our overall system in such a way that a functional logic language can be integrated as constraint solver for constraints over functional expressions. Moreover, a functional logic language can form a basis for the construction of an open constraint programming system.

Several systems and schemes have been proposed to combine different solvers of the same constraint system ([11],[6]). In contrast to them, we are able to combine constraint solvers of different constraint systems.

In [2], Baader and Schulz introduce the concept of 'free amalgamation' as a general methodology for interweaving solution structures for symbolic constraints. It is shown how constraint solvers for two components can be lifted to a constraint solver for the free amalgam. While their approach is directed to an amalgamated system based on a single theory we aim at a separated and hierarchical system, where all components are kept separated.

In [8], an environment for designing and executing constraint solver combinations BALI is presented. There are some similarities between the BALI and our approach. However, the differences between the systems are evident, and the ways and fields of application and the obtained results differ. In the end of our paper, we will compare BALI with our approach.

We want to design a consistent and open architecture to combine different constraint solvers, especially we want to enable the use of a functional logic language as constraint solver for constraints over functional expressions.

The paper is organized as follows: We start with basic definitions in Sect.2. Section 2.1 is dedicated to constraint programming. We introduce the concept of constraint solvers in Sect.2.2. In Sect.2.3, we define necessary notions of functional logic programming. Section 3 is dedicated to the description of the architecture for the combination of different constraint solvers. The overall architecture and the control of the combining system are shown in Sect.3.1. The syntax of a language which enables the specification of mixed constraints of different constraint systems is described in Sect.3.2; we define the operational semantics of such a language by means of a reduction relation in Sect.3.3; the proofs of termination and confluence of the defined reduction relation are shortly presented in this section too. The advantages of the combination of constraint solvers are shown by comparison of solution sets of constraints of the combined system and of the single systems in Sect.3.4. Different possible computation strategies are described in Sect.3.5. In Sect.4, we compare our system with the system BALI ([8]), we show the main differences between the systems and indicate advantages and disadvantages. Finally, we conclude in Sect.5 and point out directions of future research.

2 Preliminaries

Definition 1 (disjoint union). *Let A_i, $i \in \{1, \dots, n\}$, be sets.* $\dot{\bigcup}_{i\in\{1,\dots,n\}} A_i$ *is the* disjoint union *of A_i, $i \in \{1, \dots, n\}$, where*

$$\dot{\bigcup}_{i\in\{1,\dots,n\}} A_i = \begin{cases} \bigcup_{i\in\{1,\dots,n\}} A_i, & \text{if } \bigcap_{i\in\{1,\dots,n\}} A_i = \emptyset, \\ \bigcup_{i\in\{1,\dots,n\}} (A_i \times \{i\}), & \text{otherwise.} \end{cases}$$

Let S be a set of sorts. X^s denotes the set of variables $x_1, \dots, x_n$ of sort $s \in S$. $X = \dot{\bigcup}_{s\in S} X^s$ is a many sorted set of variables.

In this work, formulas of the form $\mathcal{H} = D \vee \mathcal{H}'$, with $\mathcal{H}' = D_1 \vee \dots \vee D_m$, $m \geq 0$, appear. The index value $m = 0$ means that $\mathcal{H} = D$ holds.

2.1 Constraint Programming

Definition 2 (signature). *A (many sorted)* signature *$\Sigma = (S, F, R; ar)$ is defined by a set S of sorts, a set F of function symbols, and a set R of predicate symbols. S, F, and R are mutually disjoint. The function $ar : F \cup R \to S^\star$ is called arity function of the signature Σ. For every $f \in F$, $ar(f) \neq \epsilon$ holds.*

Let Σ be a signature. In the following, we will also denote S, F, R, and ar by S_Σ, F_Σ, R_Σ, and ar_Σ, respectively.

We write $f : s_1 \times \dots \times s_n \to s$ to denote that $f \in F$ with $ar(f) = s_1 \dots s_n s$, and we write $r : s_1 \times \dots \times s_m$ to denote that $r \in R$ with $ar(r) = s_1 \dots s_m$.

Definition 3 (set of terms). *For every sort $s \in S$ the* set $\mathcal{T}(F, X)^s$ of terms of sort s *is the least set containing*

1. *every $x \in X^s$ (of sort s) and every nullary function symbol $f \in F$ with $f : \to s$, and*
2. *every $f(t_1, \ldots, t_n)$, where $f : s_1 \times \ldots \times s_n \to s$ is a function symbol in F with range s, and every $t_i, i \in \{1, \ldots, n\}$, is a term in $\mathcal{T}(F, X)^{s_i}$.*

A term is *linear* if it does not contain multiple occurrences of one variable.

Definition 4 (Σ-structure). *A* Σ-structure $\mathcal{D} = (\{\mathcal{D}^s \mid s \in S\}, \{f^{\mathcal{D}} \mid f \in F\}, \{r^{\mathcal{D}} \mid r \in R\})$ *consists of*

- *an S-sorted family of nonempty carrier sets $\mathcal{D}^s$, where $s \in S$,*
- *a family of functions $f^{\mathcal{D}}$, where $f \in F$, and*
- *a family of predicates $r^{\mathcal{D}}$, where $r \in R$.*

Given $f \in F$ with $ar(f) = s_1 \ldots s_n s$, $f^{\mathcal{D}}$ is an n-ary function such that $f^{\mathcal{D}} : \mathcal{D}^{s_1} \times \ldots \times \mathcal{D}^{s_n} \to \mathcal{D}^s$.
Given $r \in R$ with $ar(r) = s_1 \ldots s_m$, $r^{\mathcal{D}}$ is an m-ary predicate such that $r^{\mathcal{D}} \subseteq \mathcal{D}^{s_1} \times \ldots \times \mathcal{D}^{s_m}$.

Definition 5 (constraint system, constraint, constraint domain). *A* constraint system *is a tuple $\zeta = \langle \Sigma, \mathcal{D} \rangle$, where $\Sigma = (S_\Sigma, F_\Sigma, R_\Sigma; ar_\Sigma)$ is a signature and $\mathcal{D}$ is a Σ-structure.*
R_Σ contains at least a predicate symbol $=_{s,\Sigma}$ for every $s \in S_\Sigma$ with associated predicate $=^{\mathcal{D}}_s$, where $=^{\mathcal{D}}_s \subseteq \mathcal{D}^s \times \mathcal{D}^s$ with $a =^{\mathcal{D}}_s a$ for every $a \in \mathcal{D}^s$.
A constraint *is a string of the form $r\ t_1 \ldots t_m$ where $r \in R_\Sigma^{s_1 \times \ldots \times s_m}$ and $t_i \in \mathcal{T}(F_\Sigma, X)^{s_i}$. The set of constraints over Σ is denoted by $Cons(\Sigma)$.*
$(\mathcal{D}, Cons(\Sigma))$ is called a constraint domain.

The terms and constraints in $Cons(\Sigma)$ come from a first order language.

$var(t)$ denotes the set of variables occurring in a term t. Let $\tilde{y}$ denote a sequence of distinct variables. Similarly, $var(c)$, $var(\bigwedge_{i \in \{1,\ldots,m\}} c_i)$, $var(\{c_1, \ldots, c_m\})$, and $var(\tilde{y})$ denote the sets of variables which occur in a constraint c, in a conjunction $\bigwedge_{i \in \{1,\ldots,m\}} c_i$ of constraints $c_1, \ldots, c_m$, in a set $\{c_1, \ldots, c_m\}$ of constraints $c_1, \ldots c_m$, and in a sequence of variables $\tilde{y}$, respectively.
In the following, constraints are typically denoted by c, c_i, c', c'_j. If the variables $x_1, \ldots, x_n$ occur in c, i.e. $\{x_1, \ldots, x_n\} \subseteq var(c)$, then this fact is denoted by $c(x_1, \ldots, x_n)$.

Definition 6 (substitution, composition of substitutions, idempotent substitution). *A* substitution σ *is a function $\sigma : X \to \mathcal{T}(F, X)$ with*

1. *$\sigma(x) \in \mathcal{T}(F, X)^s$ for every $x \in X^s$,*
2. *$\sigma(x) = x$ for almost every $x \in X$, and*
3. *$dom(\sigma) = \{x \mid \sigma(x) \neq x\}$ is finite.*

If $dom(\sigma) = \{x_1, \ldots, x_n\}$ and $\sigma(x_i) = t_i$, we write $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. The identity substitution is denoted by id. The function σ is extended to $\sigma : \mathcal{T}(F, X) \to \mathcal{T}(F, X)$ by $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ if $f : s_1 \times \ldots \times s_n \to s, n \geq 0$.
The composition of the substitutions σ and ϕ *is defined by $(\sigma \circ \phi)(t) = \sigma(\phi(t))$ for every $t \in \mathcal{T}(F, X)$.*
A substitution σ is idempotent *iff $\sigma \circ \sigma = \sigma$ holds.*

Let $ISubst$ be the set of idempotent substitutions. Let $\uparrow : ISubst \times ISubst \to ISubst$ be the parallel composition of idempotent substitutions as defined in [9] by Palamidessi.

2.2 Constraint Solvers

The notions 'interpretation' and 'model' and the satisfaction relation $\models$ between a formula of first order predicate logic and an interpretation are defined as usual, for example see [3].

Definition 7 (valuation). *A* valuation *of a set V of variables wrt an S-sorted family of nonempty carrier sets $\mathcal{D}^s$, $s \in S$, is a mapping $\sigma: V \to \bigcup_{s \in S} \mathcal{D}^s$.*

Definition 8 (solution). *Let $\mathcal{D} = (\{\mathcal{D}^s \mid s \in S\}, \{f^{\mathcal{D}} \mid f \in F\}, \{r^{\mathcal{D}} \mid r \in R\})$ be a Σ-structure, and let $\bigwedge_{j \in \{1,\dots,n\}} c_j$ be a conjunction of constraints $c_1, \dots, c_n$ over Σ. A* solution *of $\bigwedge_{j \in \{1,\dots,n\}} c_j$ in $\mathcal{D}$ is a valuation $\sigma : X \to \bigcup_{s \in S} \mathcal{D}^s$ such that $(\mathcal{D}, \sigma) \models \bigwedge_{j \in \{1,\dots,n\}} c_j$ holds. Solving a conjunction of constraints means finding all solutions of this conjunction.*

Definition 9 (universal closure, existential closure). *Let ψ be a formula of first order predicate logic, and let $\{x_1, \dots, x_n\}$ be the set of free variables of ψ. The* universal closure *$\forall\psi$ and the* existential closure *$\exists\psi$ of formula ψ are defined as follows:*

$$\forall\psi = \forall x_1 \dots \forall x_n : \ \psi,$$
$$\exists\psi = \exists x_1 \dots \exists x_n : \ \psi.$$

$\exists_{\neg\tilde{y}}\psi$ denotes the existential closure of formula ψ except for the variables occurring in $\tilde{y}$.

A conjunction $\bigwedge_{j \in \{1,\dots,n\}} c_j$ of constraints $c_1, \dots, c_n$ is called

satisfiable or *consistent* in $\mathcal{D}$	if $\mathcal{D} \models \exists \bigwedge_{j \in \{1,\dots,n\}} c_j$,
valid in $\mathcal{D}$	if $\mathcal{D} \models \forall \bigwedge_{j \in \{1,\dots,n\}} c_j$,
unsatisfiable in $\mathcal{D}$	if $\mathcal{D} \not\models \exists \bigwedge_{j \in \{1,\dots,n\}} c_j$,
nonvalid in $\mathcal{D}$	if $\mathcal{D} \not\models \forall \bigwedge_{j \in \{1,\dots,n\}} c_j$.

A *constraint solver* is an algorithm which solves and transforms constraints of a constraint system. Usually a constraint solver offers the user the following tests and operations on conjunctions of constraints:

- *Constraint Satisfaction*: $\mathcal{D} \models \exists \bigwedge_{j \in \{1,\dots,m\}} c_j$
  The solver tries to check whether the conjunction $\bigwedge_{j \in \{1,\dots,m\}} c_j$ of constraints $c_1, \dots, c_m$ is satisfiable in the structure $\mathcal{D}$.
- *Constraint Entailment*: $\mathcal{D} \models \forall(\bigwedge_{k \in \{1,\dots,l\}} c_k \longrightarrow \bigvee_{i \in \{1,\dots,m\}}(\bigwedge_{j \in \{1,\dots,n\}} c_{i,j}))$
  It is checked whether the conjunction $\bigwedge_{k \in \{1,\dots,l\}} c_k$ of constraints $c_1, \dots, c_l$ implies the disjunction $\bigvee_{i \in \{1,\dots,m\}}(\bigwedge_{j \in \{1,\dots,n\}} c_{i,j})$ of conjunctions of constraints in $\mathcal{D}$.
- *Constraint Projection*: $\mathcal{D} \models \forall((\bigwedge_{j \in \{1,\dots,n\}} c'_j) \longleftrightarrow \exists_{\neg\tilde{x}}(\bigwedge_{i \in \{1,\dots,m\}} c_i))$
  We assume that $var(\tilde{x}) \subseteq var(\bigwedge_{i \in \{1,\dots,m\}} c_i)$. The projection of a conjunction $(\bigwedge_{i \in \{1,\dots,m\}} c_i)$ of constraints wrt the set $var(\tilde{x})$ is a conjunction $(\bigwedge_{j \in \{1,\dots,n\}} c'_j)$ of constraints in which variables from $var(\tilde{x})$ occur.

- *Determination Detection*: $\mathcal{D} \models \exists z \forall x, \tilde{y} : \ c(x, \tilde{y}) \longrightarrow x = z$
  The solver detects that, given a constraint c, there is only one value that a variable x can take that is consistent with c.

- *Simplification*:
  If $\mathcal{D} \models \forall((\bigwedge_{i \in \{1,\dots,m\}} c_i) \longleftrightarrow (\bigwedge_{j \in \{1,\dots,n\}} c'_j))$ and $\bigwedge_{j \in \{1,\dots,n\}} c'_j$ 'is simpler than' $\bigwedge_{i \in \{1,\dots,m\}} c_i$, then the constraint solver replaces $\bigwedge_{i \in \{1,\dots,m\}} c_i$ by $\bigwedge_{j \in \{1,\dots,n\}} c'_j$. $\bigwedge_{j \in \{1,\dots,n\}} c'_j$ 'is simpler than' $\bigwedge_{i \in \{1,\dots,m\}} c_i$ usually means that $\bigwedge_{j \in \{1,\dots,n\}} c'_j$ is a normal form of $\bigwedge_{i \in \{1,\dots,m\}} c_i$.

In a constraint solver, the constraint store plays a central role. A constraint store $C \in CStore$ consists of a conjunction of constraints; in particular, C has the property that it is satisfiable in the corresponding domain $\mathcal{D}$. In the initial situation, the constraint store is empty. If the satisfiability of a conjunction of constraints is to be checked, and a solution is to be found, then the constraints of the conjunction are taken one after the other, and they are propagated to the constraint store. For every chosen constraint c it is checked whether the conjunction of c and C is satisfiable in the corresponding constraint domain. In this case, C becomes $C \wedge c$, otherwise C remains as it is. The satisfiability check and the addition of the constraint to the constraint store are performed by the constraint solver (*Constraint Satisfaction, Constraint Entailment*). Moreover, by means of the above mentioned operations the constraint solver is able to simplify the set of constraints (*Determination Detection, Simplification*) and to infer constraints over subsets of variables (*Constraint Projection*).

If a constraint solver is integrated into an overall system, an interface for the communication of the constraint solver and the system is needed. The following three functions form such a uniform interface for constraint solvers:

ask:	$Cons(\Sigma) \times CStore \rightarrow \{true, false, suspend\}$
tell:	$Cons(\Sigma) \times CStore \rightarrow \{true_{changed}, true_{redundant}, false\} \times CStore$
proj:	$X \times CStore \rightarrow CStore$

ask is due to the entailment check. That means, it is checked whether a constraint c is implied by the constraint store C. In this case, i.e. if $\mathcal{D} \models \forall(C \longrightarrow c)$ holds, the result is *true*. If $C \wedge c$ is unsatisfiable, i.e. if $\mathcal{D} \not\models \exists(C \wedge c)$ holds, the result is $false$. *suspend* holds if $C \wedge c$ is satisfiable but nonvalid, i.e. if $(\mathcal{D} \models \exists(C \wedge c)) \wedge (\mathcal{D} \not\models \forall(C \wedge c))$ holds. *ask* does not change the constraint store. An *ask*-operation which is answered at a certain time by *true* will be answered always by *true* at a later stage.

tell adds a constraint c to a constraint store C if the conjunction of c and C is satisfiable. If c can be inferred by C, i.e. if $\mathcal{D} \models \forall(C \longrightarrow c)$ holds, then the result is $true_{redundant}$, and C does not change. If c cannot be inferred by C but the conjunction of c and C is satisfiable, i.e. if $(\mathcal{D} \not\models \forall(C \wedge c)) \wedge (\mathcal{D} \models \exists(C \wedge c))$ holds, then we get $true_{changed}$, and the constraint store C becomes $C \wedge c$. If the conjunction of C and c is unsatisfiable, i.e. $\mathcal{D} \not\models \exists(C \wedge c)$ holds, then the result is $false$, and C does not change. So a *tell*-operation restricts the sets of admissible values of the variables of the constraint store, or the constraint store remains constant.

The operations *ask* and *tell* are due to the concurrent constraint model ([12]), and they correspond to the entailment test and the satisfiability test, respectively. In the concurrent constraint model, *tell* has the form

tell: $\quad Cons(\Sigma) \times CStore \rightarrow \{true, false\} \times CStore,$

in fact, it has been extended for this work.

proj corresponds to the operation constraint projection. A constraint store C is projected wrt a variable x. The result is a conjunction C' of constraints induced by the constraint store: $\mathcal{D} \models \forall(C' \longleftrightarrow \exists_{-x} C)$.

The operations *ask*, *tell*, and *proj* of each constraint solver are deterministic and terminating for all input values. 'Deterministic' means that for each specific input value the computed result value is always the same. 'Terminating' describes the property of the operations to deliver the result value after a finite period of time.

Definition 10 ((in)complete constraint solver). *A constraint solver whose implemented constraint satisfaction algorithm is not able to detect all unsatisfiable constraints, that is, it cannot find out for every unsatisfiable constraint $c \in Cons(\Sigma)$ of its associated constraint domain $(\mathcal{D}, Cons(\Sigma))$ and an arbitrary conjunction $\bigwedge_{i\in\{1,\dots,n\}} c_i$ of constraints $c_i \in Cons(\Sigma)$, $i \in \{1,\dots,n\}$, whether $\mathcal{D} \vDash \exists((\bigwedge_{i\in\{1,\dots,n\}} c_i) \wedge c)$ holds or not, is an* incomplete constraint solver*. Otherwise, it is* complete*.*

If, in case of an incomplete constraint solver, the operation *tell* cannot find out for a specific constraint store C and a specific unsatisfiable constraint c whether $\mathcal{D} \vDash \exists(C \wedge c)$ holds or not then c is added to C and the result is $true_{changed}$ or $true_{redundant}$.

2.3 Functional Logic Programs

Definition 11 (multiset). *Given a set B, a* multiset *is a mapping $M : B \to \mathbb{N}$. The empty multiset is denoted by $[_M]_M$.*

We consider a signature Σ, where F is partitioned into a set Δ of *constructors* and a set Γ of *defined functions*. We write $\delta^{(n)} \in \Delta$ and $\gamma^{(n)} \in \Gamma$ for n-ary constructor symbols and function symbols, respectively. There is at least one sort *bool* which contains the 0-ary Boolean constructors *true* and *false*.
The set $\mathcal{T}(\Delta, X)$ is called the set of *constructor terms*.

Definition 12 (functional logic program). *Let $\Sigma = (S, F, R; ar)$ be a signature. A* functional logic program *P over Σ is a finite set of rules of the form $f(t_1, \dots, t_n) \to r$ `where` G., where $f \in \Gamma^{s_1\times\dots\times s_n\to s}$, $t_i \in \mathcal{T}(\Delta, X)^{s_i}$, and $r \in \mathcal{T}(F, X)^s$. $f(t_1, \dots, t_n)$ is linear, $var(r) \subseteq var(f(t_1, \dots, t_n))$ holds, and G is a finite set of constraints over Σ.*

In the usual way, P induces a congruence relation $=_P$.
Let $F_{\mathcal{FL}} \subseteq F$ be the smallest set of function symbols such that for every rule $l \to r$ `where` G. of P for every l, r holds: $l, r \in \mathcal{T}(F_{\mathcal{FL}}, X)$. Then we denote the signature $(S_{\mathcal{FL}}, F_{\mathcal{FL}}, R_{\mathcal{FL}}; ar_{\mathcal{FL}})$ by $\Sigma_{\mathcal{FL}}$, where $S_{\mathcal{FL}} \subseteq S$ is the set of sorts corresponding to $F_{\mathcal{FL}}$, $R_{\mathcal{FL}} = \{=_P\}$, and $ar_{\mathcal{FL}} = ar \mid_{F_{\mathcal{FL}} \cup R_{\mathcal{FL}}}$.

Definition 13 (answer expression). *An* answer expression *is a pair $\sigma \| e$ consisting of a substitution σ and a term e. An answer expression is called* solved *if e is a constructor term. An* extended answer expression *$\sigma \| e, G$ consists of an answer expression $\sigma \| e$ and a finite set G of constraints.*

Definition 14 (disjunctive expression). *A* disjunctive expression *is a (multi) set of extended answer expressions $[_M(\sigma_1 \| e_1, G_1), \dots, (\sigma_n \| e_n, G_n)]_M$. The set of all disjunctive expressions is denoted by Ω.*

In the following, we presuppose a function $cs : \mathcal{T}(F_{\mathcal{FL}}, X) \to \Omega \cup \{suspend\}$ which, according to a given functional logic program, performs a single computation step on a term t. It computes either a disjunctive expression of all the possible successor goals or the special constant *suspend* indicating that no reduction is possible for t.

The notions 'answer expression' and 'disjunctive expression' and the function *cs* are due to [4]. But, additionally to [4], we have to permit constraints in the rule-nodes of the definitional trees of functions as follows, and we have to use 'extended answer expressions' as defined above instead of 'answer expressions' in [4]:
$cst(t, rule(l \to r \texttt{ where } \{c_1, \ldots, c_n\}.)) = (id \| \sigma(r), \{\sigma(c_1), \ldots, \sigma(c_n)\})$ if σ is a substitution with $\sigma(l) = t$.

3 Combination of Constraint Solvers

3.1 The Architecture

The paradigm of constraint programming offers efficient mechanisms to handle constraints of various constraint domains. However, one constraint solver is only able to handle constraints of one particular constraint domain. Often it is desirable and advantageous to integrate several constraint systems and the associated constraint solving techniques (see Example 35, Sect.3.4).

One possibility for the combination of different constraint solvers is the application of meta programming techniques to the constraint programming framework. Meta programming distinguishes between the object level and the meta level. At object level statements are made about objects, and at meta level statements about the object level. We want to use this method to get a consistent and open combination of constraint solvers.

Figure 1 shows the architecture of the overall system. It works as follows: At object level different constraint solvers CS_i, $i \in \hat{L}$, $\hat{L} = \{1, \ldots, \hat{l}\}$, infer about objects of their constraint domains. To every individual constraint solver CS_r, $r \in \hat{L}$, a constraint store C^r is assigned. In this situation, we call C^r *active constraint store*. An active constraint store C^r contains the already propagated constraints of constraint system ζ_r. At meta level a mechanism handles both the constraint solvers and constraints as objects, and it coordinates the work of the different object level solvers. We call the mechanism at the meta level *meta constraint solver*. The meta constraint solver manages the *passive constraint store*. This store contains the constraints which have not been propagated so far.

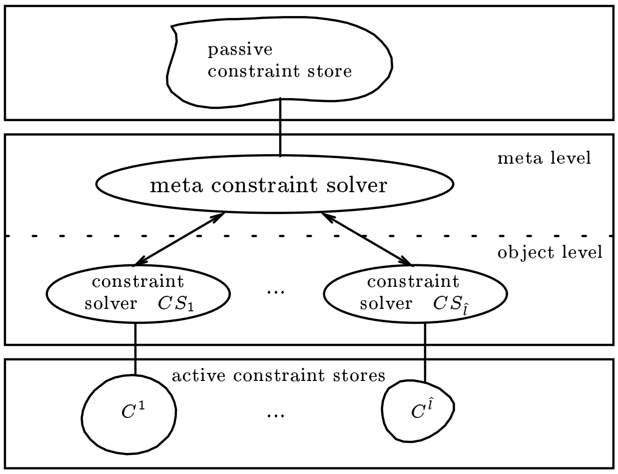


Fig. 1. Architecture of the overall system

The meta constraint solver takes constraints from the passive constraint store and passes them to the constraint solvers of the corresponding constraint domains. Each of the individual constraint solvers is able to handle a subset of the given set of constraints of the passive constraint store independently of the other constraint solvers. The meta constraint solver manages the exchange of (sub-) solutions of the individual constraint solvers. They build projections of their constraint stores wrt the other constraint domains and the variables of the overall system. These projections are added by the meta constraint solver to the passive constraint store, which is afterwards handled as before. So each individual constraint solver is able to get more information than only that of its associated subset of the originally given set of constraints of the passive constraint store. In this way, it is possible to get an overall solution.

The meta constraint solver can be used to handle meta constraints on projected constraints by transforming them in order to stronger restrict solution spaces of variables. These meta constraints can be added after the integration of a new constraint solver into a combined system. Thus, it is possible to enhance black box constraint solvers without changing them directly and after their integration into the combined architecture. In the following, the application of meta constraints to projected constraints by the meta constraint solver is left out.

The part of the architecture which consists, first, of the meta constraint solver and, second, of the individual constraint solvers CS_i, $i \in \hat{L}$, and their associated active constraint stores C^i can be regarded itself as single constraint solver MCS with associated active constraint store $MC = \bigwedge_{i \in \hat{L}} C^i$. In this way, it is possible to build a hierarchical system.

3.2 Syntax

Assume that for every $r \in L, L = \{1, \ldots, l\}$, a constraint system $\zeta_r = \langle \Sigma_r, \mathcal{D}_r \rangle$ with associated constraint solver CS_r is given. Moreover, the special constraint system $\zeta_{\mathcal{FL}}$ with constraint solver $CS_{\mathcal{FL}}$ based on a functional logic language and a user-given functional logic program is given. We want to solve a conjunction $(c_1 \wedge \ldots \wedge c_n)$ of constraints c_j, $j \in \{1, \ldots, n\}$, where each c_j, may contain function symbols and predicate symbols of different constraint systems.

To handle conjunctions of constraints by constraint solvers, it is necessary to detect overloaded symbols by analysis and to convert every conjunction of constraints into a conjunction such that every constraint is defined by function symbols and predicate symbols of exactly one constraint system. This is done by a function $Simplify$ which is provided by the meta constraint solver.

Example 15 Given two constraint systems $\zeta_1 = \langle \Sigma_1, \mathcal{D}_1 \rangle$ and $\zeta_{\mathcal{FL}} = \langle \Sigma_{\mathcal{FL}}, \mathcal{D}_{\mathcal{FL}} \rangle$ with $\Sigma_1 = \{=_1, +_1\}$, $\Sigma_{\mathcal{FL}} = \{=_P, fun_{\mathcal{FL}}\}$. Then the following transformation can be performed:

$$\begin{aligned} &Simplify(\ (\ true =_P fun_{\mathcal{FL}}(u +_1 4, v)\ )\ ) \ = \\ &\qquad (\ (true =_P z) \wedge (fun_{\mathcal{FL}}(y, v) =_P z) \wedge (y =_1 u +_1 4)\ ). \end{aligned}$$

Moreover, as in the example, $Simplify$ converts constraints of the form $(e_1 =_P e_2)$ for $e_1, e_2 \in \mathcal{T}(F_{\mathcal{FL}}, X), e_1, e_2 \notin X$, into two constraints: $(e_1 =_P y)$ and $(e_2 =_P y)$, $y \in X$, it converts a constraint $(y =_P e)$, $e \in \mathcal{T}(F_{\mathcal{FL}}, X)$, $e \notin X$, $y \in X$, into $(e =_P y)$.

To every constraint system ζ_r, $r \in L \cup \{\mathcal{FL}\}$, exactly one solver CS_r is assigned. A constraint solver can consist of a combination of constraint solvers.

3.3 Operational Semantics

Now we describe the behaviour of the overall architecture by means of the operational semantics. The coordination of the individual constraint solvers which is specified by the operational semantics is controlled by the meta constraint solver.

An overall configuration $\mathcal{H}$ consists of a disjunction $D_1 \vee \ldots \vee D_m$ of (sub-) configurations $D_1, \ldots, D_m$, $m > 0$. Every configuration D_q has the following form: $D_q = \mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r)$.

A configuration corresponds to the architecture of the overall system (see Fig.1) in such a way that the conjunction $(c_1 \wedge \ldots \wedge c_n)$ of constraints which is to be solved forms the passive constraint store $\mathcal{P}$. $\mathcal{P}$ is the smallest set of constraints such that $\forall j \in \{1, \ldots, n\} : \ c_j \in \mathcal{P}$ holds. $(\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r)$ is the conjunction of all active constraint stores. In particular, for every $r \in L \cup \{\mathcal{FL}\}$, C^r is the active constraint store of the individual constraint system ζ_r. (Recall that L contains the indices of all constraint systems of the combined overall system except that one which is based on a functional logic language.) An active constraint store contains the already propagated constraints. If $c'_1, \ldots, c'_p$ have been successfully propagated in store C^r, then $\mathcal{D} \vDash \forall (C^r \longrightarrow \bigwedge_{j \in \{1, \ldots, p\}} c'_j)$ holds. We note that every subconfiguration D_q of an overall configuration may have a different passive constraint store and different active constraint stores. Thus, an overall configuration $\mathcal{H} = D_1 \vee \ldots \vee D_m$ consists of $m > 0$ (mostly different) occurrences of the architecture.

In some cases, constraints can be represented by substitutions. To generalize the presentation of active constraint stores and to permit a functional logic language as constraint solver we denote an active store by $(\sigma \| c'_1 \wedge \ldots \wedge c'_p)$.

An overall configuration $\mathcal{H}$ is in normal form if $\mathcal{H} = \bigvee_{\kappa \in \{1, \ldots, \lambda\}} (\{true\} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r \kappa))$ or $\mathcal{H} = \{false\} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_0)$, where $\forall r \in L \cup \{\mathcal{FL}\}$: $C^r_0 = (id \| true)$.

Sometimes it is possible and advantageous to split an active constraint store C into different active stores $C1, \ldots, Ch$. In this case, we use a function *split*[1]: $split(C) = \bigvee_{\beta \in \{1, \ldots, h\}} C\beta$.

In the following, X', $X' \subseteq X$, denotes the set of variables occurring in the constraints of all passive constraint stores of all subconfigurations arising during the derivation of an overall configuration.

For the definition of the operational semantics of the overall system every constraint solver CS_ν, $\nu \in L \cup \{\mathcal{FL}\}$, must be defined in such a way that the following holds:
For every $\nu, \mu \in L \cup \{\mathcal{FL}\}$, for every $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, holds:

$$proj^{\nu \to \mu}(x, C^\nu) = \bigvee_{\gamma \in \{1, \ldots, s\}} (\bigwedge_{\delta \in \{1, \ldots, t_\gamma\}} c_{\gamma, \delta}),$$

where for every $\gamma \in \{1, \ldots, s\}$, $\delta \in \{1, \ldots, t_\gamma\}$ holds: $c_{\gamma,\delta} \in \{true, false\} \cup Cons(\Sigma_\mu)$.

That is, the projection of a constraint store C^ν wrt a variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and a constraint system ζ^μ is a disjunction of conjunctions of constraints.

[1] For example, if $C = (id \| (1 < x) \wedge (x \neq 5) \wedge (x < 9))$, then $split(C) = C1 \vee C2$ with $C1 = (id \| (1 < x) \wedge (x < 5))$ and $C2 = (id \| (5 < x) \wedge (x < 9))$.

Property 16 (monotonicity of the operation projection) To prevent a loss of information because of a different order of choice of propagated constraints, projection must be a monotone operation; $\forall \nu, \mu \in L \cup \{\mathcal{FL}\}$:

If c is a constraint which is projected by a constraint store C^ν wrt a variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and a constraint system ζ_μ:
$proj^{\nu \to \mu}(x, C^\nu) = \ldots \vee (\ldots \wedge c \wedge \ldots) \vee \ldots$, and

${C'}^\nu_{new}\beta$ is a new constraint store after k application(s) of the operation *tell* (and if necessary the operation *split*) to the constraint store C^ν (*tell* restricts the solution space of the constraint store, or it remains constant, according to Sect.2.2):
$tell(c_1, C^\nu) = (val, {C'}^\nu)$,
$split({C'}^\nu) = \bigvee_{\beta' \in \{1,\ldots,h'\}} {C'}^\nu \beta'$
$\ldots$
$tell(c_k, {C'}^\nu \beta' \ldots \gamma') = (val, {C'}^\nu_{new})$,
$split({C'}^\nu_{new}) = \bigvee_{\beta \in \{1,\ldots,h\}} {C'}^\nu_{new}\beta$, and

if c' is a constraint which is projected by the new constraint store ${C'}^\nu_{new}\beta$ wrt the variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and the constraint system ζ_μ
$proj^{\nu \to \mu}(x, {C'}^\nu_{new}\beta) = \ldots \vee (\ldots \wedge c' \wedge \ldots) \vee \ldots$, then the following holds:

If c' is successfully added to the appropriate constraint store C^μ:
$tell(c', C^\mu) = (val, {C'}^\mu)$ and
$split({C'}^\mu) = \bigvee_{\beta'' \in \{1,\ldots,h''\}} {C'}^\mu_{new}\beta''$,

then the constraint c is redundant wrt the new constraint store ${C'}^\mu_{new}\beta''$:
$tell(c, {C'}^\mu_{new}\beta'') = (true_{redundant}, {C'}^\mu_{new}\beta'')$.

At this, $val \in \{true_{changed}, true_{redundant}\}$, $c, c' \in Cons(\Sigma_\mu)$, and $\forall i \in \{1, \ldots, k\}$: $c_i \in Cons(\Sigma_\nu)$ hold.

In the following, we will specify the operational semantics of the overall system.

1. At the beginning, the active stores of all individual constraint solvers are empty:
   $\forall r \in L \cup \{\mathcal{FL}\}: \; C^r = C^r_0 = (id \| true)$.
   The passive constraint store $\mathcal{P}$ is the smallest set which contains all constraints $c_1, \ldots, c_n$ of the conjunction $(c_1 \wedge \ldots \wedge c_n)$ of constraints which is to be solved.
   So the initial configuration is $\mathcal{P} \odot \bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_0$.

2. A derivation step of the overall configuration is given in the following:

   $\mathcal{H} \vee \mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \vee \mathcal{H}' \longrightarrow^\nu \mathcal{H} \vee \mathcal{H}'' \vee \mathcal{H}'$,
   $\mathcal{H} = D_1 \vee \ldots \vee D_m$, $m \geq 0$, $\mathcal{H}' = D'_1 \vee \ldots \vee D'_{m'}$, $m' \geq 0$,
   if $\mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r)$ is not in normal form.

   (a) For every constraint system ζ_ν except those with a constraint solver based on a functional logic language, i.e. $\nu \neq \mathcal{FL}$, $\mathcal{H}''$ is specified as follows:

i. $\mathcal{H}'' = \bigvee_{\beta\in\{1,\dots,h\}} \bigvee_{\alpha\in\{1,\dots,m_\beta\}}$
$((\mathcal{P}\backslash\{c\}) \cup (\bigcup_{\theta\in\{1,\dots,n_\alpha\}}\{c_{\beta,\alpha,\theta}\})) \odot ((\bigwedge_{r\in L\cup\{\mathcal{FL}\},r\neq\nu} C^r) \wedge C^\nu_{new}\beta)$

if $c \in Cons(\Sigma_\nu)\backslash\{true, false\}$, $c \in \mathcal{P}$,
$tell(c, C^\nu) = (true_{changed}, C^\nu_{new})$,
$split(C^\nu_{new}) = \bigvee_{\beta\in\{1,\dots,h\}} C^\nu_{new}\beta$.

Projection of $C^\nu_{new}\beta$ wrt every variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and every constraint system $\zeta_r, r \in L, r \neq \nu$:
$\bigwedge_{r\in L\cup\{\mathcal{FL}\},r\neq\nu} \bigwedge_{x\in X'\cap X^s, s\in S_{\Sigma_\nu}} proj^{\nu\to r}(x, C^\nu_{new}\beta) =$
$\bigvee_{\alpha\in\{1,\dots,m_\beta\}}(\bigwedge_{\theta\in\{1,\dots,n_\alpha\}} c_{\beta,\alpha,\theta})$.

$tell(c, C^\nu) = (true_{changed}, C^\nu_{new})$ means that the constraint c is added to the active constraint store C^ν of system ζ_ν to build the new active constraint store C^ν_{new}. The conjunction of c and C^ν is satisfiable but nonvalid.
The new active constraint store C^ν_{new} is projected wrt each variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and wrt each other constraint system ζ_r, $r \in L \cup \{\mathcal{FL}\}, r \neq \nu$. In this way, each of the new alternative (sub-) constraint stores $C^\nu_{new}\beta$ is projected.
For each of the newly formed configurations of $\mathcal{H}''$ the corresponding new constraint store $C^\nu_{new}\beta$ is the active constraint store of ζ_ν.

ii. C^ν does not change adding a redundant constraint c to C^ν. c can be inferred from C^ν: $\mathcal{D} \models \forall(C^\nu \longrightarrow c)$.

$\mathcal{H}'' = ((\mathcal{P}\backslash\{c\}) \cup \{true\}) \odot (\bigwedge_{r\in L\cup\{\mathcal{FL}\}} C^r)$

if $c \in Cons(\Sigma_\nu)\backslash\{true, false\}$, $c \in \mathcal{P}$, and
$tell(c, C^\nu) = (true_{redundant}, C^\nu)$.

iii. The constraint c is in contradiction to the active constraint store C^ν: $\mathcal{D} \not\models \exists(C^\nu \wedge c)$.

$\mathcal{H}'' = ((\mathcal{P}\backslash\{c\}) \cup \{false\}) \odot (\bigwedge_{r\in L\cup\{\mathcal{FL}\}} C^r)$

if $c \in Cons(\Sigma_\nu)\backslash\{true, false\}$, $c \in \mathcal{P}$, and
$tell(c, C^\nu) = (false, C^\nu)$.

(b) In a constraint system $\zeta_{\mathcal{FL}}$ with the special constraint solver $CS_{\mathcal{FL}}$ which is based on a functional logic language and a given functional logic program P, $\mathcal{H}''$ is computed as follows ($\nu = \mathcal{FL}$):
   i. If $(e =_P y)$ is the constraint which is to be propagated and $\sigma[\![\sigma(e)$ is unsolved, that is, $\sigma(e)$ is not a constructor term, then the constraint solver tries to perform a computation step and computes $cs(\sigma(e))$. If this is possible, then the result of $cs(\sigma(e))$ is a disjunctive expression. At the derivation of $\sigma(e)$ arise sets of constraints.
   The new constraint store $C^{\mathcal{FL}}_{new}$ is computed by the composition of the computed substitutions σ_β, $\beta \in \{1,\dots,h\}$, of the disjunctive expression and the substitution σ of $C^{\mathcal{FL}}$. $C^{\mathcal{FL}}_{new}$ is split into different active constraint stores.

Each of the new passive constraint stores of the configurations of $\mathcal{H}''$ is built of three components: first, the old passive constraint store $\mathcal{P}$ without c, second, projections of the new active constraint store of the constraint system $\mathcal{FL}$ wrt the other constraint systems, and third, one of the extended answer expressions of the disjunctive expression.

$$\mathcal{H}'' = \bigvee_{\beta\in\{1,\dots,h\}} \bigvee_{\alpha\in\{1,\dots,m_\beta\}} ((\mathcal{P}\backslash\{c\}) \cup (\bigcup_{\theta\in\{1,\dots,n'_\alpha\}}\{c'_{\beta,\alpha,\theta}\}) \cup \{c_{\beta,1},\dots,c_{\beta,n_\beta}\} \cup \{(e_\beta =_P y)\}) \odot ((\bigwedge_{r\in L} C^r) \wedge C^{\mathcal{FL}}_{new}\beta)$$

if $c \in \mathcal{P}$ has the form $(e =_P y)$, $y \in X'$, $c \in Cons(\Sigma_{\mathcal{FL}})$,
$e, \sigma(e) \in \mathcal{T}(F_{\mathcal{FL}}, X')$, $\sigma(e) \notin X'$,
$\sigma\|\sigma(e)$ is unsolved, $cs(\sigma(e)) \neq suspend$ and $cs(\sigma(e)) \neq \emptyset$.
$C^{\mathcal{FL}} = (\sigma\|true)$.

In this case,
$tell(e =_P y, C^{\mathcal{FL}}) := (true_{changed}, C^{\mathcal{FL}}_{new})$ and
$split(C^{\mathcal{FL}}_{new}) = \bigvee_{\beta\in\{1,\dots,h\}} C^{\mathcal{FL}}_{new}\beta$ hold, where
$C^{\mathcal{FL}}_{new}\beta = (\sigma_\beta \circ \sigma\|true)$, $\beta \in \{1,\dots,h\}$.

Derivation of $\sigma(e)$:
$cs(\sigma(e)) = [_M(\sigma_1\|e_1, \{c_{1,1},\dots,c_{1,n_1}\}),\dots,(\sigma_h\|e_h,\{c_{h,1},\dots,c_{h,n_h}\})]_M$.

Projection of $C^{\mathcal{FL}}_{new}\beta$ wrt every variable $x \in X' \cap X^s$, $s \in S_{\Sigma_{\mathcal{FL}}}$, and every constraint system $\zeta_r, r \in L$:

$$\bigwedge_{r\in L} \bigwedge_{x\in X'\cap X^s, s\in S_{\Sigma_{\mathcal{FL}}}} proj^{\mathcal{FL}\to r}(x, C^{\mathcal{FL}}_{new}\beta) = \bigvee_{\alpha\in\{1,\dots,m_\beta\}} (\bigwedge_{\theta\in\{1,\dots,n'_\alpha\}} c'_{\beta,\alpha,\theta}).$$

ii. $\mathcal{H}'' = \bigvee_{\alpha\in\{1,\dots,m\}}((\mathcal{P}\backslash\{c\}) \cup (\bigcup_{\theta\in\{1,\dots,n'_\alpha\}}\{c'_{\alpha,\theta}\})) \odot ((\bigwedge_{r\in L} C^r) \wedge C^{\mathcal{FL}}_{new})$

if $c \in \mathcal{P}$, $c = (e =_P y)$, $y \in X'$, $c \in Cons(\Sigma_{\mathcal{FL}})$, $e, \sigma(e) \in \mathcal{T}(F_{\mathcal{FL}}, X')$, and $(\sigma\|\sigma(e)$ is solved or $\sigma(e) \in X')$.
$C^{\mathcal{FL}} = (\sigma\|true)$.

In this case,
$tell(e =_P y, C^{\mathcal{FL}}) := (true_{changed}, C^{\mathcal{FL}}_{new})$ and
$C^{\mathcal{FL}}_{new} = (\{y/e\} \circ \sigma\|true) = (\sigma'\|true)$ hold.

Projection of $C^{\mathcal{FL}}_{new}$ wrt every variable $x \in X' \cap X^s$, $s \in S_{\Sigma_{\mathcal{FL}}}$, and every constraint system $\zeta_r, r \in L$:
$\bigwedge_{r\in L} \bigwedge_{x\in X'\cap X^s, s\in S_{\Sigma_{\mathcal{FL}}}} proj^{\mathcal{FL}\to r}(x, C^{\mathcal{FL}}_{new}) = \bigvee_{\alpha\in\{1,\dots,m\}} (\bigwedge_{\theta\in\{1,\dots,n'_\alpha\}} c'_{\alpha,\theta}).$

If $(e =_P y)$ is the constraint which is to be propagated and either $\sigma\|\sigma(e)$ is solved, that is, $\sigma(e)$ is a constructor term, or $\sigma(e) \in X'$ holds, then the new constraint

store $C^{\mathcal{FL}}_{new}$ is computed by the composition of the substitutions $\{y/e\}$ and σ of $C^{\mathcal{FL}}$.
The new constraint store is projected wrt the variables $x \in X' \cap X^s$, $s \in S_{\Sigma_{\mathcal{FL}}}$, and all other constraint systems.

iii. If $(e =_P y)$ is the constraint which is to be propagated, $\sigma \| \sigma(e)$ is unsolved, and $cs(\sigma(e)) = \emptyset$, then the propagation of the constraint $(e =_P y)$ fails.

$\mathcal{H}'' = ((\mathcal{P} \backslash \{c\}) \cup \{false\}) \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r)$ and
$tell(c, C^{\mathcal{FL}}) := (false, C^{\mathcal{FL}})$

if $c \in \mathcal{P}$, $c = (e =_P y)$, $y \in X'$, $c \in Cons(\Sigma_{\mathcal{FL}})$, $e, \sigma(e) \in \mathcal{T}(F_{\mathcal{FL}}, X')$, $\sigma \| \sigma(e)$ is unsolved, and $cs(\sigma(e)) = \emptyset$.
$C^{\mathcal{FL}} = (\sigma \| true)$.

iv. If all constraints in $\mathcal{P}$ are constraints c'_i, $i \in \{1, \ldots, p\}$, which have the form $(e_i =_P y_i)$ and $\forall i \in \{1, \ldots, p\}$: $\sigma \| \sigma(e_i)$ is unsolved, $\sigma(e_i) \notin X'$, and $cs(\sigma(e_i)) = suspend$ holds, then the following holds:

$\mathcal{H}'' = \{true\} \odot (\bigwedge_{r \in L} C^r) \wedge C^{\mathcal{FL}}_{new}$

if $\forall c'_i \in \mathcal{P}$, $i \in \{1, \ldots, p\}$: $c'_i = (e_i =_P y_i)$, $c'_i \in Cons(\Sigma_{\mathcal{FL}})$, $e_i, \sigma(e_i) \in \mathcal{T}(F_{\mathcal{FL}}, X')$, $\sigma(e_i) \notin X'$, $y_i \in X'$, $\forall i \in \{1, \ldots, p\}$: $\sigma \| \sigma(e_i)$ is unsolved, and $cs(\sigma(e_i)) = suspend$.
$C^{\mathcal{FL}} = (\sigma \| true)$.

In this case,
$C^{\mathcal{FL}}_{new} := (\sigma \| \bigwedge_{i \in \{1, \ldots, p\}} c'_i)$ holds.

3. Moreover, there are some derivation rules to simplify configurations:

(a) $\mathcal{H} \vee \mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \vee \mathcal{H}' \longrightarrow^0 \mathcal{H} \vee \mathcal{H}'$
if $false \in \mathcal{P}$ and $\mathcal{H} = D_1 \vee \ldots \vee D_m$, $\mathcal{H}' = D'_1 \vee \ldots \vee D'_{m'}$, $m + m' > 0$,
(b) $\mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \longrightarrow^0 \{false\} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r)$
if $false \in \mathcal{P}$ and $\exists c \in \mathcal{P} : c \neq false$,
(c) $\mathcal{H} \vee \mathcal{P} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \vee \mathcal{H}' \longrightarrow^0 \mathcal{H} \vee (\mathcal{P} \backslash \{true\}) \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \vee \mathcal{H}'$
if $true \in \mathcal{P}$, $\exists c \in \mathcal{P} : c \neq true$, and
$\mathcal{H} = D_1 \vee \ldots \vee D_m$, $m \geq 0$, $\mathcal{H}' = D'_1 \vee \ldots \vee D'_{m'}$, $m' \geq 0$.
(d) $\{false\} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r) \longrightarrow^0 \{false\} \odot (\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_0)$
if $\exists \nu \in L \cup \{\mathcal{FL}\}$: $C^\nu \neq C^\nu_0$.

Note 17 In case 2b, only constraints of the form $(e =_P y)$, $y \in X'$, $e \in \mathcal{T}(F_{\mathcal{FL}}, X')$, are regarded. Constraints of the form $(e_1 =_P e_2)$, $e_1, e_2 \in \mathcal{T}(F_{\mathcal{FL}}, X')$, $e_1, e_2 \notin X'$, or $(y =_P e)$, $y \in X'$, $e \in \mathcal{T}(F_{\mathcal{FL}}, X')$, $e \notin X'$, are handled by converting them by the function $Simplify$.

In every derivation step, one constraint $c \in Cons(\Sigma_\nu)$ of the passive constraint store $\mathcal{P}$ of a configuration is added to the active constraint store C^ν of the constraint system ζ_ν.

Projections wrt all other constraint systems and the variables $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, are added to the new passive constraint store afterwards. If the constraint solver is based on a functional logic language and a given functional logic program, then the behaviour is a bit modified. If a constraint $c = (e =_P y)$, $c \in Cons(\Sigma_{\mathcal{FL}})$, is taken from the passive constraint store, and the active constraint store $C^{\mathcal{FL}}$ of $\zeta_{\mathcal{FL}}$ contains the substitution σ, then the constraint solver tries to perform a computation step on $\sigma(e)$. The computed derivations and substitutions are added to the passive constraint store afterwards. This behaviour is repeated until $\sigma(e)$ is a constructor term.

An example for the cooperation of combined constraint solvers to solve a conjunction of constraints is given in the Appendix.

This description of the operational semantics is not implementation oriented. For easier understanding, different possible optimizations are left out, with help of which the number of steps of a derivation sequence could be less in many cases.

Definition 18 (reduction system [1]). *If $\mathcal{E}$ is a set and $\longrightarrow$ is a binary relation over $\mathcal{E}$, then $(\mathcal{E}, \longrightarrow)$ is called a* reduction system.

Definition 19 (reduction system $(\Xi', \Longrightarrow)$).
Let $(\Xi', \Longrightarrow)$ be a reduction system, where Ξ' is the set of configurations and $\Longrightarrow = \bigcup_{\nu \in L \cup \{\mathcal{FL}, 0\}} \longrightarrow^{\nu}$.

After the successful derivation of a given configuration $\bigvee_{v \in \{1,\dots,\omega\}} \mathcal{P}_v \odot \bigwedge_{r \in L \cup \{FL\}} C^r v$ to a normal form $\bigvee_{\kappa \in \{1,\dots,\lambda\}} \{true\} \odot \bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_{new} \kappa$ the λ conjunctions $\bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_{new} \kappa$, $\kappa \in \{1, \dots, \lambda\}$, of the recent active constraint stores contain constraints and substitutions which describe a set of solutions. Moreover, it is possible to assign to each variable $x \in \bigcup_{v \in \{1,\dots,\omega\}} var(\mathcal{P}_v)$ of sort $s \in S$ a subset $\mathcal{D}^s_x$ of $\mathcal{D}^s$ by projections such that for all solutions σ holds: $\sigma(x) \in \mathcal{D}^s_x$. A derivation to normal form $(\{false\} \odot \bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_0)$ is a failing derviation; that is, the conjunction of constraints which is to be solved has no solution.

If we want to compute one or more specific solutions of a configuration after a successful derivation, then this is done as follows; we describe the general method by one possible search strategy, i.e. backtracking[2]: By means of heuristics a variable x of sort $s \in S$ of the finite set of variables $\bigcup_{v \in \{1,\dots,\omega\}} var(\mathcal{P}_v)$ is chosen, for example that variable, whose number of admissible values is minimal in most of the recent active constraint stores but not bound to exactly one value. Now one of the admissible values of the corresponding domain, i.e. an element of $\mathcal{D}^s_x$, is assigned to x. This assignment is projected wrt all constraint systems, and the newly built projections are added to the passive constraint store of each of the subconfigurations of the overall configuration in normal form. Additionally, all constraints of the conjunction $\bigwedge_{i \in \{1,\dots,p\}} c'_i$ of constraints of the active constraint store $C^{\mathcal{FL}} = (\sigma' \| \bigwedge_{i \in \{1,\dots,p\}} c'_i)$ are added to the passive constraint store of the associated subconfiguration and $(\sigma' \| true)$ is assigned to $C^{\mathcal{FL}}$. This is necessary to use the results of the computations of the constraint solvers CS_r, $r \in L$, to compute function applications by residuation. Now a derivation of the overall configuration to normal form follows. In case of a successful derivation, the next variable is chosen and the procedure is repeated as before until to all variables values of their corresponding domains are assigned. In this case, a solution can be obtained by projecting the active constraint stores of the configuration wrt all variables of $\bigcup_{v \in \{1,\dots,\omega\}} var(\mathcal{P}_v)$. Otherwise, the derived normal form of a configuration is $(\{false\} \odot \bigwedge_{r \in L \cup \{\mathcal{FL}\}} C^r_0)$, i.e. the derivation fails; then the assignment for the last chosen variable y must be countermanded and a new value must be assigned to y. If all possible assignments for a variable are followed by a failing derivation, then the assignment of the previously chosen variable z must be

[2] Other strategies (see [10]) are possible too.

changed, afterwards to y once more one of the admissible values is assigned. In this way, we may get a solution.

To get more than one solution, we have to save an obtained solution or to put it out, and afterwards, as in the case of a failed derivation, the choice of the assignment for the last chosen variable y is countermanded and a new value is assigned to y, now the same procedure as in the case of a failed derivation follows.

Termination and Confluency

1. Termination

Definition 20 (termination [1]). *Let $(\mathcal{E}, \longrightarrow)$ be a reduction system. $\longrightarrow$ is terminating if there is no infinite chain $u_1 \longrightarrow u_2 \longrightarrow u_3 \longrightarrow \ldots$*

Let *value* be a function which computes the set of admissible values of a variable wrt a disjunction of conjunctions of constraints:

$$value(x, \bigvee_{\alpha \in \{1,\ldots,m\}} \bigwedge_{\theta \in \{1,\ldots,n_\alpha\}} c_{\alpha,\theta}) = \bigcup_{\alpha \in \{1,\ldots,m\}} \{e \mid e \in \bigcup_{s \in S} \mathcal{D}_r^s \text{ and } (\mathcal{D}_r \models \exists (\bigwedge_{\theta \in \{1,\ldots,n_\alpha\}} c_{\alpha,\theta} \longleftarrow (x = e)))\},$$
where $c_{\alpha,\theta} \in Cons(\Sigma_r)$. ζ_r, $r \in L' \subset L \cup \{\mathcal{FL}\}$, is a constraint system.

To show that $\Longrightarrow$ of the reduction system $(\Xi', \Longrightarrow)$ is terminating, we need the following **assumptions** for the constraint systems ζ_r, $r \in L'$, $L' \subset L \cup \{\mathcal{FL}\}$:

A1 A constraint solver $CS_{\mathcal{FL}}$ based on a functional logic language and a functional logic program is excluded.
A single derivation step according to function cs of the operational semantics of a functional logic language (see [4]) is terminating but also in case of a terminating rewrite relation, the narrowing derivations need not be terminating because the domains need not be finite.

A2 The set $X' \subseteq X$ of variables occuring in the constraints of all passive constraint stores of all subconfigurations arising during the derivation of an overall configuration is finite.

A3 If the set of admissible values of a variable $z \in X'$ wrt an active constraint store $C^i = (id \llbracket \bigwedge_{k \in \{1,\ldots,n\}} c_k)$, $i \in L'$, of a constraint system ζ_i is finite, then the set of admissible values of the projected conjunction of constraints of C^i wrt z and its own constraint system must be finite too:
$\forall z \in X'$, $\forall i \in L'$: If $value(z, \bigwedge_{k \in \{1,\ldots,n\}} c_k)$ is a finite set, $C^i = (id \llbracket \bigwedge_{k \in \{1,\ldots,n\}} c_k)$, then $value(z, proj^{i \to i}(z, C^i))$ must be finite.

A4 $\forall i, j \in L'$, $i \neq j$: If $\exists z \in X'$ such that, $C^i = (id \llbracket \bigwedge_{k \in \{1,\ldots,n\}} c_k)$ and $value(z, \bigwedge_{k \in \{1,\ldots,n\}} c_k)$ is infinite, then for each constraint system ζ_j with infinite domains and for each $y \in X'$ either
- $value(y, proj^{i \to j}(y, C^i))$ is finite or
- $value(y, proj^{i \to j}(y, C^i))$ is infinite and $value(y, proj^{i \to i}(y, C^i))$ is finite.

If the set of admissible values of a variable z for the active constraint store C^i of a constraint system ζ_i is infinite, then the projection of C^i wrt another constraint system ζ_j with infinite domains must be finite or the projections wrt to the constraint system ζ_i itself must be finite.

A5 (a) Let $\forall \nu, \mu \in L'$: $proj^{\nu \rightarrow \mu}(x, C^\nu) = f(proj^{\nu \rightarrow \nu}(x, C^\nu))$ hold.

(b) $\forall i, j \in L', i \neq j$:
If $C^i = (id \llbracket \bigwedge_{k \in \{1,\dots,n\}} c_k)$ and $\mathcal{D}_i \vDash \forall (\bigwedge_{k \in \{1,\dots,n\}} c_k)$, then $\forall x \in X'$:
If $value(x, proj^{i \rightarrow i}(x, C^i))$ is a finite set and $x \in X^s \cap X'$, $s \in S_{\Sigma_i}$,
then $value(x, proj^{i \rightarrow i}(x, C^i)) \neq \emptyset$.
If $value(x, proj^{i \rightarrow i}(x, C^i))$ is infinite,
$value(x, proj^{i \rightarrow j}(x, C^i))$ is finite, and $x \in X^s \cap X'$, $s \in S_{\Sigma_j}$,
then $value(x, proj^{i \rightarrow j}(x, C^i)) \neq \emptyset$.

Theorem 21 (termination of $\Longrightarrow$) Let Assumptions A1, A2, A3, A4, and A5 hold for the reduction system $(\Xi', \Longrightarrow)$. Then $\Longrightarrow$ is terminating.

Sketch of the proof 22 (Theorem 21) The number of constraints in the passive constraint store of each subconfiguration is finite. During a derivation constraints are chosen, one after the other, from the passive constraint store of a subconfiguration of the overall configuration. Each chosen constraint is propagated to the associated active constraint store building a new active constraint store, which is possibly split into a disjunction of active constraint stores. Afterwards, projections of these new active constraint stores wrt all other than the associated constraint systems and a subset of the finite set X' of variables are built. From each projection, i.e. a disjunction of conjunctions of constraints, smallest sets which contain the constraints of one conjunction are built. Since the number of constraints in each conjunction of a projection is finite, the newly built sets of constraints, which are added to the recent passive constraint store of a subconfiguration which is built in this process, are finite. Hence, during each derivation the number of subconfiguration of an overall configuration as well as the number of constraints in the passive constraint store of each subconfiguration is finite.
Thus,

1. by multiple application of Königs Lemma ([1]),
2. by definition of a Noetherian partial ordering $>$ over the set of tuples of constraints and the configuration in which they occur, and
3. by showing that the tuples of constraints (of the passive constraint store) and their configurations of subsequently derived configurations build a descending chain wrt $>$,

it is possible to proof termination.

Note 23 We think that Assumptions A2 - A5 are reasonable and evident, and they are simply to obtain in most cases. Since the number of variables X' is finite according to Assumption A2, the proof can be extended for combined systems including a constraint solver $CS_{\mathcal{FL}}$ for terminating rewrite relations.

2. Confluency

Definition 24 (confluence, local confluence [1]). *Let $(\mathcal{E}, \longrightarrow)$ be a reduction system.*
$\longrightarrow$ *is called* confluent *if* $\overset{\star}{\longleftarrow} \circ \overset{\star}{\longrightarrow} \subseteq \downarrow$ *holds, i.e.: From* $u \overset{\star}{\longrightarrow} x$ *and* $u \overset{\star}{\longrightarrow} y$ *follows that there is a* z *such that* $x \overset{\star}{\longrightarrow} z$ *and* $y \overset{\star}{\longrightarrow} z$, *that is* $x \downarrow y$,
$\longrightarrow$ *is called* locally confluent *if* $\longleftarrow \circ \longrightarrow \subseteq \downarrow$ *holds, i.e.: From* $u \longrightarrow x$ *and* $u \longrightarrow y$ *follows* $x \downarrow y$.

Recall from Sect.2.2 that the operations *ask*, *tell* and *proj* of each constraint solver are deterministic and terminating for all input values.
The derivations of subconfigurations of an overall configuration do not influence each other.

Case 1: If we can assume a deterministic choice of the constraint which is to be propagated from the passive constraint store of a subconfiguration, then the reduction system is confluent because there is always only one matching rule for one subconfiguration. Thus, the following holds: For all subconfigurations $\mathcal{H}$, $\mathcal{H}'$, and $\mathcal{H}''$: From $\mathcal{H} \stackrel{\star}{\Longrightarrow} \mathcal{H}'$ and $\mathcal{H} \stackrel{\star}{\Longrightarrow} \mathcal{H}''$ follows $\mathcal{H}' \downarrow \mathcal{H}''$ because either $\mathcal{H}' \stackrel{\star}{\Longrightarrow} \mathcal{H}''$ holds, or $\mathcal{H}'' \stackrel{\star}{\Longrightarrow} \mathcal{H}'$.

Case 2: If a deterministic choice of the next constraint which is to be propagated of the passive constraint store of a subconfiguration cannot be assumed, then all subconfigurations $\mathcal{H}'$, $\mathcal{H}''$, ..., which can be derived from a subconfiguration $\mathcal{H}$, must be considered. It must be shown that for all subconfigurations $\mathcal{H}$, $\mathcal{H}'$, and $\mathcal{H}''$ the following holds: From $\mathcal{H} \stackrel{\star}{\Longrightarrow} \mathcal{H}'$ and $\mathcal{H} \stackrel{\star}{\Longrightarrow} \mathcal{H}''$ follows $\mathcal{H}' \downarrow \mathcal{H}''$.

In the following, constraint systems ζ_r, $r \in L''$, $L'' \subseteq L' \subseteq L$, for which Assumptions A1, A2, A3, A4, and A5 hold, are considered. It has been shown that $\Longrightarrow$ is terminating for these assumptions. Hence, according to the Newman-Lemma ([1]) it is sufficient to show local confluence for $\Longrightarrow$. We have to show local confluence for each two rules of Items 2a and 3, Sect.3.3, which are applicable to a subconfiguration at the same time.

To split an active constraint store C into different active stores $C1, \dots, Ch$, we use the function *split* (see 3.3). To ensure confluence, the following **assumptions** for *split* must hold:

A6 If it is possible to split an active constraint store C^ν, $\nu \in L''$, into a disjunction of h active constraint stores $C^\nu 1, \dots, C^\nu h$, then the projection of C^ν wrt a constraint system ζ_μ, $\mu \in L''$, and a variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, is the disjunction of the projections of all $C^\nu\beta$, $\beta \in \{1, \dots, h\}$, wrt x:
$\forall \nu, \mu \in L'', \forall x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$:
If $split(C^\nu) = \bigvee_{\beta\in\{1,\dots,h\}} C^\nu\beta$, then $proj^{\nu\to\mu}(x, C^\nu) = \bigvee_{\beta\in\{1,\dots,h\}} proj^{\nu\to\mu}(x, C^\nu\beta)$.

A7 A constraint $c \in Cons(\Sigma_\nu)$ is successfully added to the active constraint store C^ν creating C^ν_{new}, C^ν_{new} is split into a disjunction $\bigvee_{\beta\in\{1,\dots,h\}} C^\nu_{new}\beta$ of active constraint stores. The same disjunction of active constraint stores is built if, first, C^ν is split into a disjunction $\bigvee_{\beta'\in\{1,\dots,h'\}} C^\nu\beta'$, second, constraint c is propagated to each active constraint store $C^\nu\beta'$, $\beta' \in \{1, \dots, h'\}$, and, third, all by successful propagation of c newly created $\hat{C}^\nu_{new}\beta'$, $\beta' \in \{1, \dots, h'\}\backslash\{f, \dots, g\}$, are split into disjunctions of constraint stores and these disjunctions are disjunctively connected.

$\forall \nu \in L''$, $c \in Cons(\Sigma_\nu)$:
$tell(c, C^\nu) = (val, C^\nu_{new})$ and
$split(C^\nu_{new}) = \bigvee_{\beta\in\{1,\dots,h\}} C^\nu_{new}\beta$
$\Longleftrightarrow$
$split(C^\nu) = \bigvee_{\beta'\in\{1,\dots,h'\}} C^\nu\beta'$,
$\forall \beta' \in \{f, \dots, g\}, \{f, \dots, g\} \subset \{1, \dots, h'\}$: $tell(c, C^\nu\beta') = (false, C^\nu\beta')$,
$\forall \beta' \in \{1, \dots, h'\}\backslash\{f, \dots, g\}$: $tell(c, C^\nu\beta') = (val_{\beta'}, \hat{C}^\nu_{new}\beta')$ and
$\bigvee_{\beta'\in\{1,\dots,h'\}\backslash\{f,\dots,g\}} split(\hat{C}^\nu_{new}\beta') = \bigvee_{\beta\in\{1,\dots,h\}} C^\nu_{new}\beta$, where
$val \neq false$, $val_{\beta'} \neq false$.

Note 25 Assumptions A6 and A7 are not necessary if the choice of the constraint of the passive constraint store of a configuration which will be propagated next is deterministic for each constraint system.

Theorem 26 (confluence of $\Longrightarrow$) Let Assumptions A1, A2, A3, A4, A5, A6, and A7 hold for the reduction system $(\Xi', \Longrightarrow)$. Then $\Longrightarrow$ is confluent.

Sketch of the proof 27 (Theorem 26) Since the choice of the rule for the application to one particular constraint of the passive constraint store of a subconfiguration is deterministic, we need to consider only such cases, where $\Longrightarrow$ is applied to different constraints of the passive constraint store of a subconfiguration. We are able to show confluence of $\Longrightarrow$ showing local confluence for the application of each two rules of Sect.3.3 to a subconfiguration and by applying the Newman-Lemma ([1]).

3.4 Comparison of Sets of Solutions

Definition 28 (theory). *Let $\mathcal{PF}$ be the set of formulas of first order predicate logic, $M \subseteq \mathcal{PF}$. M is called consistent iff for all formulas $m \in \mathcal{PF}$ holds: not $((M \models m)$ and $(M \models \neg m))$. M is a* theory *iff M is consistent and for all $m \in \mathcal{PF}$ holds: $(M \models m) \longrightarrow (m \in M)$.*

Definition 29 (Σ-theory). *Let $\mathcal{D}$ be a Σ-structure, let $\mathcal{PF}$ be the set of formulas of first order predicate logic. Then $Th(\mathcal{D}) = \{m \in \mathcal{PF} \mid \mathcal{D} \models m\}$ is the* Σ-theory.

Definition 30 (satisfaction complete theory [7]). *Let $\zeta = \langle \Sigma, \mathcal{D} \rangle$ be a constraint system. $Th(\mathcal{D})$ is* satisfaction complete *wrt $Cons(\Sigma)$ ([7]) if for every constraint $c \in Cons(\Sigma)$ either $Th(\mathcal{D}) \models \exists c$ or $Th(\mathcal{D}) \models \neg\exists c$ holds.*

Definition 31 (valuation σ restricted to a set of variables W). *Let σ be a valuation. A* valuation σ restricted to a set of variables W, *denoted by $\sigma|_W$, is defined as follows:*

$$\sigma|_W(w) = \begin{cases} \sigma(w) & \textit{if } w \in W, \\ w & \textit{otherwise.} \end{cases}$$

Definition 32 ($=_W$, $\leq_W$, $<_W$). *Let $\mathcal{L}$ resp. $\mathcal{L}'$ be the set of all solutions of the conjunction $\bigwedge_{i\in\{1,\ldots,n\}} c_i$ of constraints $c_1, \ldots, c_n$ wrt $Th(\mathcal{D})$ resp. $\bigwedge_{j\in\{1,\ldots,n'\}} c'_j$ of $c'_1, \ldots, c'_{n'}$ wrt $Th(\mathcal{D}')$.*
The set $\mathcal{L}$ of solutions is as good as *the set $\mathcal{L}'$ of solutions wrt the set W of variables, denoted by $\mathcal{L} =_W \mathcal{L}'$, iff $\forall \sigma \in \mathcal{L}$ holds: $\exists \sigma' \in \mathcal{L}'$ s.t. $\sigma|_W = \sigma'|_W$ and $\forall \sigma' \in \mathcal{L}'$ holds: $\exists \sigma \in \mathcal{L}$ s.t. $\sigma|_W = \sigma'|_W$.*
The set $\mathcal{L}$ of solutions is at least as good as *the set $\mathcal{L}'$ of solutions wrt the set W of variables, denoted by $\mathcal{L} \leq_W \mathcal{L}'$, iff $\forall \sigma \in \mathcal{L}$ holds: $\exists \sigma' \in \mathcal{L}'$, $\exists \phi$ s.t. $\sigma|_W = (\phi \circ \sigma')|_W$.*
The set $\mathcal{L}$ of solutions is better than *the set $\mathcal{L}'$ of solutions wrt the set W of variables, denoted by $\mathcal{L} <_W \mathcal{L}'$, iff $\forall \sigma \in \mathcal{L}$ holds: $\exists \sigma' \in \mathcal{L}'$, $\exists \phi$ s.t. $\sigma|_W = (\phi \circ \sigma')|_W$, and $\exists \sigma' \in \mathcal{L}'$: $\nexists \sigma \in \mathcal{L}$ s.t. $\sigma|_W = \sigma'|_W$.*

In the following, we regard constraint systems ζ_j, $j \in \hat{L}$, whose function symbols and relation symbols of the signatures Σ_j are disjunct. Let $\mathcal{CT}_{\hat{L}} = Th(\mathcal{D}_{\hat{L}})$ be the Σ-theory of the combined constraint system $\zeta_{\hat{L}} = \langle \Sigma_{\hat{L}}, \mathcal{D}_{\hat{L}} \rangle$, where $\Sigma_{\hat{L}} = (S, F, R; ar)$, $S = \bigcup_{j\in\hat{L}} S_j$, $F = \dot{\bigcup}_{j\in\hat{L}} F_j$, $R = \dot{\bigcup}_{j\in\hat{L}} R_j$, and $ar : F \cup R \rightarrow S^\star$, and $\mathcal{D}_{\hat{L}} = (\{\mathcal{D}^s \mid s \in S\}, \{f^{\mathcal{D}} \mid f \in F\}, \{r^{\mathcal{D}} \mid r \in R\})$ with $\{\mathcal{D}^s \mid s \in S\}$ is the set of all carrier sets of the structures $\mathcal{D}_j$ of the combined constraint systems ζ_j, $j \in \hat{L}$.

Property 33 (soundness of the operation projection wrt a conjunction of constraints)
$\forall \nu, \mu \in \tilde{L}$, $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$:
If $\bigwedge_{i \in \{1,\dots,n\}} c_i$ is the conjunction of constraints which is to be solved,
$\mathcal{P} \odot \bigwedge_{r \in \tilde{L}} C^r \stackrel{\star}{\Longrightarrow} \bigvee_{\beta \in \{1,\dots,h\}} \mathcal{P}'\beta \odot \bigwedge_{r \in \tilde{L}} C^r_{new}\beta$ holds, where
$\mathcal{P}$ is the smallest set containing all c_i, $i \in \{1,\dots,n\}$, and
$proj^{\nu \to \mu}(x, C^\nu_{new}\beta) = \bigvee_{\gamma \in \{1,\dots,u_\beta\}} (\bigwedge_{\delta \in \{1,\dots,t_\gamma\}} c_{\gamma,\delta})$,
then $\mathcal{CT}_{\tilde{L}} \vDash \forall((\bigwedge_{i \in \{1,\dots,n\}} c_i) \longrightarrow \bigvee_{\beta \in \{1,\dots,h\}} (\bigvee_{\gamma \in \{1,\dots,u_\beta\}} (\bigwedge_{\delta \in \{1,\dots,t_\gamma\}} c_{\gamma,\delta})))$.

The disjunction of all projections of the active constraint stores $C^\nu_{new}\beta$, $\nu \in \tilde{L}$, $\beta \in \{1,\dots,h\}$, wrt a variable $x \in X' \cap X^s$, $s \in S_{\Sigma_\nu}$, and a constraint system ζ_μ, $\mu \in \tilde{L}$, is an implication of the conjunction $\bigwedge_{i \in \{1,\dots,n\}} c_i$ of constraints which is to be solved.

In the following, we assume that for all constraint systems ζ_r, $r \in \hat{L}$, Property 33 holds.

Claim 34 Let ζ_j, $j \in \hat{L}$, be constraint systems with associated constraint solvers CS_j. Let $\bigwedge_{i \in \{1,\dots,n\}} c_i$ be the conjunction of constraints which is to be solved, where $\forall i \in \{1,\dots,n\}$ $\exists j \in \hat{L}$: $c_i \in Cons(\Sigma_j)$ holds. For $j \in \hat{L}$ $\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j}$ is the conjunction of all those constraints of $\bigwedge_{i \in \{1,\dots,n\}} c_i$ which are elements of $Cons(\Sigma_j)$.

$\mathcal{L}_{\hat{L}}$ is the set of solutions of $\bigwedge_{i \in \{1,\dots,n\}} c_i$ in $\mathcal{CT}_{\hat{L}}$. $\mathcal{L}^{BT}_{\hat{L}}$ is the set of solutions of $\bigwedge_{i \in \{1,\dots,n\}} c_i$ which is computed in a backtracking sequence using $\Longrightarrow$ in an architecture which is built by combining all constraint systems ζ_j, $j \in \hat{L}$. $\mathcal{L}_j$, $j \in \hat{L}$, is the set of solutions of $\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j}$ in $\mathcal{D}_j$. $\mathcal{L}^{BT}_j$, $j \in \hat{L}$, is the set of solutions of $\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j}$ which is computed in a backtracking sequence using $\Longrightarrow$ in an architecture which is built of the single constraint system ζ_j. The following holds:

1. *$\forall j \in \hat{L}$: $\mathcal{L}_{\hat{L}} \leq_{var(\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j})} \mathcal{L}_j$.*
   $\forall \sigma \in \mathcal{L}_{\hat{L}}$: $\mathcal{CT}_{\hat{L}} \vDash \sigma(\bigwedge_{i \in \{1,\dots,n\}} c_i)$ and $\forall \sigma_j \in \mathcal{L}_j, j \in \hat{L}$: $Th(\mathcal{D}_j) \vDash \sigma_j(\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j})$.

   If $\mathcal{L}_{\hat{L}}$ is better than $\mathcal{L}_j$, $j \in \hat{L}$, wrt $var(\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j})$ then $\mathcal{L}_j$ contains a solution under which $\bigwedge_{i \in \{1,\dots,n\}} c_i$ is not satisfiable in $\mathcal{CT}_{\hat{L}}$:
   $\forall j \in \hat{L}$: If $\mathcal{L}_{\hat{L}} <_{var(\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j})} \mathcal{L}_j$, then $\exists \sigma_j \in \mathcal{L}_j$: $\mathcal{CT}_{\hat{L}} \vDash \nexists \sigma_j(\bigwedge_{i \in \{1,\dots,n\}} c_i)$.

2. *Let $\mathcal{P} \odot \bigwedge_{r \in \hat{L}} C^r \stackrel{\star}{\Longrightarrow} \bigvee_{\beta \in \{1,\dots,h\}} \{true\} \odot \bigwedge_{r \in \hat{L}} C^r_{new}\beta$ and*
   $\mathcal{P}' \odot C^j \stackrel{\star}{\Longrightarrow} \bigvee_{\beta' \in \{1,\dots,h'\}} \{true\} \odot C'^j_{new}\beta'$ hold, where
   $\mathcal{P}$ is the smallest set which contains all c_i, $i \in \{1,\dots,n\}$, and $\mathcal{P}'$ is the smallest set which contains all $c'_{i,j}$, $i \in \{1,\dots,n_j\}$, of one constraint system ζ_j, $j \in \hat{L}$.

   If $\mathcal{CT}_{\hat{L}}$ is satisfaction complete then
   $\mathcal{CT}_{\hat{L}} \vDash \forall(\bigwedge_{i \in \{1,\dots,n\}} c_i \longleftrightarrow \bigvee_{\beta \in \{1,\dots,h\}} \bigwedge_{r \in \hat{L}} C^r_{new}\beta)$ holds.
   If $Th(\mathcal{D}_j)$, $j \in \hat{L}$, is satisfaction complete then
   $Th(\mathcal{D}_j) \vDash \forall(\bigwedge_{i \in \{1,\dots,n_j\}} c'_{i,j} \longleftrightarrow \bigvee_{\beta' \in \{1,\dots,h'\}} C'^j_{new}\beta')$ holds.

   $\mathcal{L}_{\hat{L}} =_{\{var(\bigwedge_{i \in \{1,\dots,n\}} c_i)\}} \mathcal{L}_{\bigcap_{r \in \hat{L}} \zeta_r}$.

   If every $Th(\mathcal{D}_j)$, $j \in \hat{L}$, and $\mathcal{CT}_{\hat{L}}$ are satisfaction complete, all constraint solvers CS_j, $j \in \hat{L}$, are complete, and the computations of $\mathcal{L}^{BT}_{\hat{L}}$ and $\mathcal{L}^{BT}_j$ by backtracking sequences ap-

plying $\Longrightarrow$ terminate, then $\mathcal{L}^{BT}_{\hat{L}} =_{\{var(\bigwedge_{i\in\{1,\dots,n\}c_i})\}} \mathcal{L}_{\hat{L}}$ and $\mathcal{L}^{BT}_{j} =_{var(\bigwedge_{i\in\{1,\dots,n_j\}} c'_{i,j})} \mathcal{L}_j$ hold.
If at least one of the solvers CS_j, $j \in \hat{L}$, is incomplete, and the computations of $\mathcal{L}^{BT}_{\hat{L}}$ and $\mathcal{L}^{BT}_{j}$, $j \in \hat{L}$, by backtracking sequences applying $\Longrightarrow$ terminate, then $\mathcal{L}_{\hat{L}} \le_{\{var(\bigwedge_{i\in\{1,\dots,n\}c_i})\}} \mathcal{L}^{BT}_{\hat{L}}$ and $\mathcal{L}_{\bigcap_{j\in\hat{L}}\zeta_j} \le_{\{var(\bigwedge_{i\in\{1,\dots,n\}c_i})\}} \mathcal{L}^{BT}_{\bigcap_{j\in\hat{L}}\zeta_j}$ holds.

At this, $\mathcal{L}_{\bigcap_{r\in\{1,\dots,\hat{l}\}}\zeta_r} = \{ \underbrace{\uparrow(\dots\uparrow(\uparrow(}_{(\hat{l}-1)}\sigma_1,\sigma_2),\sigma_3),\dots,\sigma_{\hat{l}})\mid \sigma_r \in \mathcal{L}_r\}$, *and*

$\mathcal{L}^{BT}_{\bigcap_{r\in\{1,\dots,\hat{l}\}}\zeta_r}$ is defined accordingly by replacing $\mathcal{L}_r$ by $\mathcal{L}^{BT}_r$.

If $\exists j \in \hat{L}$: $\mathcal{L}_{\hat{L}}$ is better than $\mathcal{L}_j$ wrt $var(\bigwedge_{i\in\{1,\dots,n_j\}} c'_{i,j})$ then, using our approach, the number of steps of

- choosing a variable $x \in var(\bigwedge_{i\in\{1,\dots,n\}} c_i)$,
- assigning an admissible value of the corresponding domain to the chosen variable,
- changing this assignment for x if necessary, and

the overall number of derivations of a configuration to normal form by $\Longrightarrow$ in a backtracking sequence is less than in most cases of computing $\mathcal{L}_j$, $j \in \hat{L}$, of the single systems and finding solutions by backtracking because $\mathcal{L}_j$ can contain solutions under which $\bigwedge_{i\in\{1,\dots,n\}} c_i$ is not satisfiable in $\mathcal{CT}_{\hat{L}}$.
Moreover, in case of incomplete constraint solvers, the combination of constraint solvers by projections sometimes makes it possible to explicitly state solutions which we can not explicitly name without this combination.

Example 35 Let $\mathcal{FD}$ and $\mathcal{FL}$ be constraint systems which are defined as follows:

$\mathcal{FD} = \langle \Sigma_{\mathcal{FD}},\ \mathcal{D}_{\mathcal{FD}}\rangle$ with
$\Sigma_{\mathcal{FD}}$ with $R_{\mathcal{FD}} = R^{nat\times p-nat}_{\mathcal{FD}} \cup R^{nat\times nat}_{\mathcal{FD}}$ and $F_{\mathcal{FD}} = F^{nat\times nat\times nat}_{\mathcal{FD}}$, where
$S_{\mathcal{FD}} = \{nat, p-nat\}$,
$R^{nat\times p-nat}_{\mathcal{FD}} = \{\in_{\mathcal{FD}}\}$,
$R^{nat\times nat}_{\mathcal{FD}} = \{\ =_{\mathcal{FD}}\}$,
$F^{nat\times nat\times nat}_{\mathcal{FD}} = \{\ +_{\mathcal{FD}}\}$,
$\mathcal{D}_{\mathcal{FD}} = (\mathbb{N}_{\mathcal{FD}} \cup \mathcal{P}(\mathbb{N}_{\mathcal{FD}}), \{+\}, \{\in, =\})$ with
$\mathbb{N}_{\mathcal{FD}} = \{1,2,3,4\}$, $\mathbb{N}_{\mathcal{FD}} \subset \mathbb{N}$,
where $\mathbb{N}$ is the set of natural numbers, and
$\in, =, +$ are defined in the usual way.

$\mathcal{FL} = \langle \Sigma_{\mathcal{FL}},\ \mathcal{D}_{\mathcal{FL}}\rangle$ with
$\Sigma_{\mathcal{FL}}$ with
$R_{\mathcal{FL}} = R^{\tau\times\tau}_{\mathcal{FL}}$ and
$F_{\mathcal{FL}} = \dot{\bigcup}_{\tau\in\{nat,bool\}^*\times\{nat,bool\}} F^{\tau}$ contains the function symbols of user defined functions, $true$, $false$ and $n \in \mathbb{N}$,
$R^{\tau\times\tau}_{\mathcal{FL}} = \ \{=_P\}$, $\tau \in \{nat, bool\}$,
$\mathcal{D}_{\mathcal{FL}} = (\mathcal{T}(F_{\mathcal{FL}}, X), \{f_{FL} \mid f \in F_{\mathcal{FL}}\}, \{=_P\})$,
f_{FL} is defined according to the program P, which is given by the user:
$f(3) \to 3$ `where` *true*.
$f(4) \to 1$ `where` *true*.

$X = X^{nat} = \{x, y, w, z\}$.

The constraint solver $CS_{\mathcal{FD}}$ is based on an arc consistency algorithm. The constraint solver $CS_{\mathcal{FL},nar}$ of the constraint system $\mathcal{FL}$ is based on a functional logic language and the given functional logic program P. $CS_{\mathcal{FL},nar}$ uses narrowing.
The following **conjunction of constraints** is to be solved:

$$
\begin{array}{ll}
c_1 & (\ (f(x) =_P z) \wedge \\
c_2 & \quad (z +_{\mathcal{FD}} w =_{\mathcal{FD}} 4) \wedge \\
c_3 & \quad (f(y) =_P w) \wedge \\
c_4 & \quad (x \in_{\mathcal{FD}} \{1,2,3,4\}) \wedge \\
c_5 & \quad (y \in_{\mathcal{FD}} \{1,2,3,4\}) \wedge \\
c_6 & \quad (z \in_{\mathcal{FD}} \{1,2,3,4\}) \wedge \\
c_7 & \quad (w \in_{\mathcal{FD}} \{1,2,3,4\})\ ).
\end{array}
$$

The sets $\mathcal{L}_{\mathcal{FD}}$ and $\mathcal{L}^{BT}_{\mathcal{FD}}$ of solutions of $\bigwedge_{i \in \{2,4,5,6,7\}} c_i$ are the following:

$$
\begin{array}{ll}
\mathcal{L}_{\mathcal{FD}} & = \\
\mathcal{L}^{BT}_{\mathcal{FD}} & = \{\ \{x/1, y/1, w/1, z/3\}, \{x/1, y/2, w/1, z/3\}, \ldots, \{x/4, y/4, w/1, z/3\}, \\
& \quad\ \ \{x/1, y/1, w/2, z/2\}, \{x/1, y/2, w/2, z/2\}, \ldots, \{x/4, y/4, w/2, z/2\}, \\
& \quad\ \ \{x/1, y/1, w/3, z/1\}, \{x/1, y/2, w/3, z/1\}, \ldots, \{x/4, y/4, w/3, z/1\}\}.
\end{array}
$$

The sets $\mathcal{L}_{\mathcal{FL},nar}$ and $\mathcal{L}^{BT}_{\mathcal{FL},nar}$ of solutions of $\bigwedge_{i \in \{1,3\}} c_i$ are the following:

$$
\begin{array}{ll}
\mathcal{L}_{\mathcal{FL},nar} & = \\
\mathcal{L}^{BT}_{\mathcal{FL},nar} & = \{\ \{x/3, y/3, w/3, z/3\}, \{x/3, y/4, w/1, z/3\}, \\
& \quad\ \ \{x/4, y/3, w/3, z/1\}, \{x/4, y/4, w/1, z/1\}\}.
\end{array}
$$

Sets $\mathcal{L}_{\{\mathcal{FD},\mathcal{FL}\}}$ and $\mathcal{L}^{BT}_{\{\mathcal{FD},\mathcal{FL}\}}$of solutions of $\bigwedge_{i \in \{1,\ldots,7\}} c_i$:

$$
\begin{array}{ll}
\mathcal{L}_{\{\mathcal{FD},\mathcal{FL}\}} & = \\
\mathcal{L}^{BT}_{\{\mathcal{FD},\mathcal{FL}\}} & = \{\ \{x/3, y/4, w/1, z/3\}, \{x/4, y/3, w/3, z/1\}\}.
\end{array}
$$

The following holds: $\mathcal{L}_{\{\mathcal{FD},\mathcal{FL}\}} <_{\{x,y,w,z\}} \mathcal{L}_{\mathcal{FD}}$ and $\mathcal{L}_{\{\mathcal{FD},\mathcal{FL}\}} <_{\{x,y,w,z\}} \mathcal{L}_{\mathcal{FL},nar}$.

$\mathcal{L}_{\{\mathcal{FD},\mathcal{FL}\}} =_{\{x,y,w,z\}} \mathcal{L}_{\mathcal{FD} \cap \mathcal{FL},nar}$, where
$\mathcal{L}_{\mathcal{FD} \cap \mathcal{FL},nar} = \{\uparrow (\sigma_{\mathcal{FD}}, \sigma_{\mathcal{FL},nar}) | \sigma_{\mathcal{FD}} \in \mathcal{L}_{\mathcal{FD}}, \sigma_{\mathcal{FL},nar} \in \mathcal{L}_{\mathcal{FL},nar}\}$.

However, lets consider the same example, where, instead of $CS_{\mathcal{FL},nar}$, we use a constraint solver $CS_{\mathcal{FL},res}$ which uses residuation. In this case, we cannot ensure in general that the computation of $\mathcal{L}^{BT}_{\mathcal{FL},res}$ terminates because the sets of admissible values of the variables are infinite. However, if we combine the systems, then we are able to ensure termination of the computation of $\mathcal{L}^{BT}_{\{\mathcal{FD},\mathcal{FL}\}}$ because the sets of admissible values of the variables are finite in $\mathcal{D}_{\mathcal{FD}}$.

3.5 Different Computation Strategies

As well as the sequential strategy which is defined by the operational semantics of Sect.3.3 it is possible to define different computation strategies. In this section, we want to describe informally two very evident computation strategies.
First, lets consider a *parallel* strategy. In one derivation step, the passive constraint store $\mathcal{P}$ of a subconfiguration $\mathcal{P} \odot \bigwedge_{r \in \hat{L}} C^r$ is split into disjoint subsets of constraints, where each set

contains all and only those constraints of one particular constraint domain. These sets are propagated to the active stores of the corresponding constraint systems. If all constraints of such a subset have been successfully added to the associated active constraint store and there was at least one constraint which was not redundant, then the constraint solver projects the newly built active constraint store wrt all variables and all other constraint systems. If all propagated constraints of a particular constraint domain were redundant then the associated constraint solver projects *true*. If the *tell*-operation of one constraint to a particular active constraint store fails, then the according constraint solver finishes its work and projects $false$. If all constraint solvers finished their work, their projections are used to transform the original configuration into a new disjunction of configurations. After the generation of the new overall configuration simplification steps may be performed. Now each subconfiguration is handled as $\mathcal{P} \odot \bigwedge_{r \in \hat{L}} C^r$ before until the overall configuration is in normal form.

The advantages of the parallel strategy are, first, that all constraint solvers can propagate all constraints of their associated domain of the passive constraint store of a configuration independently of the other constraint solvers and parallel to them until a synchronization by building new configurations occurs, and, second, that projections are not built after the propagation of every constraint of the passive constraint store of a configuration but after the propagation of all constraints of a particular constraint system. Thus, the projection of redundant constraints is prevented.

The other evident strategy describes the *concurrent* propagation of constraints to different active constraint stores. First, choosing a constraint $c \in Cons(\Sigma_\nu)$, $\nu \in \hat{L}$, of the passive constraint store $\mathcal{P}$ of a subconfiguration $\mathcal{P} \odot \bigwedge_{r \in \hat{L}} C^r$ and propagating it to the appropriate active constraint store C^ν by *tell* building C^ν_{new} (*get*), and second, adding the projections of C^ν_{new} wrt all variables $x \in X' \cap X^s, s \in S_{\Sigma_\nu}$, and all other constraint systems to $\mathcal{P} \backslash \{c\}$ and building, if necessary, several subconfigurations (*put*) must be regarded now as single steps to realize this strategy. That is, $\Longrightarrow$ must be split into two derivation relations $\Longrightarrow_{get}$ and $\Longrightarrow_{put}$. Now each constraint solver can take constraints from the recent passive constraint store of a subconfiguration and built from the recent overall configuration a new recent overall configuration. Since the projection of an active constraint store is a disjunction of conjunctions of constraints, the generation of disjunctions of subconfigurations from one subconfiguration must be taken into consideration in case of a *put*-step.

As in the parallel case, all constraint solvers can propagate their constraints independently of the other constraint solvers and at the same time. As in the parallel case too, projections need not be built after the propagation of each constraint of the passive constraint store of a configuration. But now synchronization steps as necessary in the parallel strategies do not appear. However, the control of *get*-steps and *put*-steps of different constraint solvers, especially the coordination of the generation of disjunctive subconfigurations, can become difficult and expensive.

Different computation strategies can be applied in different subarchitectures of an overall architecture in a hierarchical system (see Sect.3.1). Moreover, different strategies can be used in different subsequently following derivations of a configuration to normal form between two steps of value assignments to different variables in a backtracking computation (see Sect.3.3).

The formal definition of these strategies and the specification of language constructs for their application in subarchitectures as well as in different subsequently following derivations of a configuration to normal form in a backtracking computation is one task of future research.

4 Comparison of our Approach and BALI ([8])

In [8], a *B*inding *A*rchitecture for So*L*ver *I*ntegration, BALI, is presented. BALI is a domain independent environment for executing solver collaborations. At this, 'solver collaboration' means 'solver combination' and 'solver coordination', where 'solver combination' focuses on building a solver for the union of theories and 'solver coordination' concerns communication problems between solvers devoted to a single domain but with possibly different admissible constraints. In contrast, we use 'combination' in the sense of combining constraint solvers of different constraint domains as well as different constraint solvers of the same constraint domain. In our architecture, constraint solvers communicate by exchanging projections of their recently propagated constraints and by a set of variables which is common to all individual constraint systems.

Both systems are glass box mechanisms which enable linking black box tools which are the individual (heterogeneous) constraint solvers. While in BALI the single solvers need not be in relation one to each other, in our architecture it is advantageous but not necessary that the constraint solvers build projections wrt the other constraint systems. Moreover, in our system in contrast to BALI, by means of using meta constraints for the transformation of constraints (see Sect.3.1) it is possible to enhance the functionalities of the black box constraint solvers without changing them directly and after integrating them into the overall combined system. Since we allow the integration of incomplete constraint solvers, which is not allowed in BALI, the user has a wider range for the choice of a specific solver for a particular constraint domain. The combination of constraint solvers by projections as presented in Sect.3.3 enables not only the introduction of incomplete constraint solvers. It makes it possible to get solutions which the individual solvers are not able to specify in some cases.

Besides some control primitives (e.g. guard, iterator), BALI provides three collaboration primitives for the composition of individual solvers.

'Sequentiality' enables to execute a solver on a disjunction of conjunctions of constraints which is the result of another solver.

The second primitive 'parallelism' allows the execution of several solvers in parallel. One conjunction of the to be solved disjunction of conjunctions of constraints is split into disjoint subproblems wrt the constraint domains of the single solvers. Then the single solvers are applied to their associated subproblems, and after termination of the work of all solvers a disjunction of conjunctions of the solutions of all solvers is the result. Since it is possible that a constraint is a constraint of the domain of different individual collaborating constraint solvers in BALI, splitting a conjunction of constraints into disjoint subproblems wrt the constraint domains of the individual solvers may cause some trouble.

The third primitive 'concurrency' works as follows: One conjunction of the to be solved disjunction of conjunctions of constraints is split into subproblems wrt the constraint domains of the single solvers. In contrast to 'parallelism', these subproblems need not be disjoint. The result of the application of this primitive is the output of a single constraint solver. This primitive provides a nondeterministic choice to select one of the outputs of the different solvers depending on the execution time of the solvers for the problem which is to be solved.

While BALI's collaboration primitives aim at the coordination of the work of the individual constraint solvers, our strategies (see Sect.3.3, Sect.3.5) aim at the choice of particular constraints from the to be solved disjunction of conjunctions of constraints. Thus, we can say that our composition is more fine grain than that of BALI. Our different strategies can be applied in different subarchitectures of an overall architecture in a hierarchical system. Moreover, they can be used in different subsequently following derivations of a configuration to normal form between two steps of value assignments to different variables in a backtracking computation.

Using our sequential strategy (Sect.3.3) in a special way, we can simulate a strategy which is very similar to the application of the primitive 'sequentiality' in BALI. Our passive strategy is very similar to the primitive 'parallelism' of BALI. In our approach, concurrency is used to make use of the independence of the work of different constraint solvers and to prevent synchronization steps which are necessary in the parallel strategy.

In BALI, the user has to specify the coordination of the work of the individual solvers explicitly. Then, in the fixed, explicitly specified order, the constraint solvers are applied to the problem. Thus, in BALI, the user must know at the beginning of the computation which order of application of constraint solvers is necessary and advantageous to solve a specific problem. In contrast, in our system, a configuration is transformed according to a specified strategy until a nomal form is reached. Using our system the user gets the same result, i.e. normal form of a configuration (provided that certain properties for the constraint solvers hold), independently of the choice of the strategy and without specifying the concrete order of application of the specific individual solvers.

The host language of BALI is a constraint logic programming language which must be integrated by a special mechanism. The host language of our system is a functional logic language with use of which we are able to specify rules like in a constraint logic programming language, and additionally, typical rules of functional logic programs. Our host language is integrated elegantly into the overall system as a special constraint solver for functional expressions. In the same way, another host language could be integrated.

As well as BALI, in [8], a scheme for extending constraint solvers with new function symbols, SoleX, is presented. Together with a new function symbol a database of rewrite rules which express the relations of the new function symbol and the function symbols and relation symbols of the constraint solver which is to be extended must be provided by the user. SoleX itself consists of a set of transformation rules. They are used to transform constraints according to the rewrite rules such that constraints of the original domain extended by the new function symbols can be handled. That is, SoleX enables the extension of a constraint solver by new functions of the same constraint domain. Transformation rules handle the cooperation of the original constraint solver and the new functions. Since new functions are integrated into the overall system by a very general mechanism, the new functions are less expressive, less efficient, and worse integrated than built-in functions but, at the other hand, a solver extension on a very general level is possible.

A similar mechanism is used in our system. In our system, first, meta constraints, or second, a combination of the original constraint solver and a new constraint solver for the new function symbols can be used to extend a constraint solver by new functions. In contrast to the rewrite rules of SoleX, in our system, the handling of constraints which contain new function symbols need not be expressed in the constraint domain of the original solver or a predefined syntax of rewrite rules. But, a restriction to the domain of the original constraint solver becomes active later when projections are built. However, since we are able to combine constraint solvers of different domains, written in different languages and using different strategies, our system is more general than SoleX.

Monfroy ([8]) intends to study the similarities of BALI and SoleX to merge the two concepts for realizing a framework including solver collaboration and solver extension. We think that our system realizes this merge, which makes it possible to combine black box solvers and to enhance the functionalities of this black box solvers without changing them directly and after integrating them into the overall combined system.

5 Conclusion and Future Work

We described a mechanism to combine different constraint solvers by means of projections. The integration of a functional logic language as constraint solver for constraints over functional expressions into the overall system is enabled. Constraints of our system can contain function symbols and predicate symbols of different constraint systems, and they are solved by cooperating solvers. Besides the formal specification of a sequential computation strategy for the combined constraint solvers by means of a reduction system $(\Xi', \Longrightarrow)$, we described other strategies in an informal way. Thus, one of our future tasks will be the analysis and formal description of these strategies and to find out which strategy is advantageous in which cases of application.

Another aspect of future research is the propagation of the initial knowledge of the constraint solvers because of the definition of the constraint domains.

We compared sets of solutions of the combined overall system with that of single constraint systems to show the advantages of our approach. To proof Claim 34, to proof confluence and termination of the informally described strategies and their combinations, and to show that the computed sets of solutions using these strategies and combinations of strategies remain the same wrt sequential computation is one task of future research.

We compared our system with another system for the combination of constraint solvers, showed the main differences between the systems, and indicated advantages and disadvantages.

Acknowledgement The author would like to thank Heiko Vogler for careful reading of parts of this article, discussions, and helpful suggestions.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press. 1998.
2. Baader, F., Schulz, K.U.: On the Combination of Symbolic Constraints, Solution Domains, and Constraint Solvers. In: Rossi, F., Montanari, U. (Eds.): Principles and Practice of Constraint Programming - CP'95. pp.380-397. LNCS 976. Berlin 1995.
3. Ebbinghaus, H.-D., Flum, J., Thomas, W.: Mathematical Logic. Berlin 1996.
4. Hanus, M.: A Unified Computation Model for Functional and Logic Programming. In: Proc. of the 24th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp.80-93. Paris 1997.
5. Hanus, M.: The Integration of Functions into Logic Programming: From Theory to Practice. In: Journal of Logic Programming, Vol.19&20, pp.583-628. 1994.
6. Hong, H.: Confluency of Cooperative Constraint Solvers. Technical Report. Research Institute for Symbolic Computation. Linz 1994.
7. Jaffar, J., Maher, M.J.: Constraint Logic Programming: A Survey. In: Journal of Logic Programming, Vol.19&20, pp.503-581. 1994.
8. Monfroy, E.: Solver Collaboration for Constraint Logic Programming. PhD Thesis. Centre de Recherche en Informatique de Nancy INRIA-Lorraine 1996.
9. Palamidessi, C.: Algebraic Properties of Idempotent Substitutions. In: Paterson, M.S. (Ed.): Automata, Languages and Programming. pp.386-399. LNCS 443. Berlin 1990.
10. Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. In: Computational Intelligence, Vol.9, No.3, pp.268-299. 1993.
11. Rueher, M.: An Architecture for Cooperating Constraint Solvers on Reals. In: Podelski, A. (Ed.): Constraint Programming: Basics and Trends. Châtillon Spring School 1994. Selected Papers. pp.231-250. LNCS 910. Berlin 1995.
12. Saraswat, V.A.: Concurrent Constraint Programming. Cambridge, Mass., 1993.

Appendix

Example 36 Let $\mathcal{FD}$ and $\mathcal{RAT}$ be constraint systems which are defined as follows:

$\mathcal{FD} = \langle \Sigma_{\mathcal{FD}},\ \mathcal{D}_{\mathcal{FD}} \rangle$ with
$\Sigma_{\mathcal{FD}}$ with $R_{\mathcal{FD}} = R_{\mathcal{FD}}^{rat \times p-rat} \cup R_{\mathcal{FD}}^{rat \times rat}$, where
$S_{\mathcal{FD}} = \{rat, p-rat\}$,
$R_{\mathcal{FD}}^{rat \times p-rat} = \{\in_{\mathcal{FD}}\}$,
$R_{\mathcal{FD}}^{rat \times rat} \ \ = \{=_{\mathcal{FD}}, >_{\mathcal{FD}}\ \}$,
$\mathcal{D}_{\mathcal{FD}} = (\mathbb{Q}_{\mathcal{FD}} \cup \mathcal{P}(\mathbb{Q}_{\mathcal{FD}}), \emptyset, \{\in, =, >\ \})$ with
$\mathbb{Q}_{\mathcal{FD}} = \{1, 2, 3, 4, 5, 6\}$, $\mathbb{Q}_{\mathcal{FD}} \subset \mathbb{Q}$,
where $\mathbb{Q}$ is the set of rational numbers, and
$\in$, $=$, and $>$ are defined in the usual way.

$\mathcal{RAT} = \langle \Sigma_{\mathcal{RAT}},\ \mathcal{D}_{\mathcal{RAT}} \rangle$ with
$\Sigma_{\mathcal{RAT}}$ with
$S_{\mathcal{RAT}} = \{rat\}$,
$F_{\mathcal{RAT}} = F_{\mathcal{RAT}}^{rat \times rat \to rat}$,
$R_{\mathcal{RAT}} = R_{\mathcal{RAT}}^{rat \times rat}$, where
$F_{\mathcal{RAT}}^{rat \times rat \to rat} = \{+_{\mathcal{RAT}}\}$,
$R_{\mathcal{RAT}}^{rat \times rat} \ \ \ \ \ = \{=_{\mathcal{RAT}}, <_{\mathcal{RAT}}, >_{\mathcal{RAT}}, \leq_{\mathcal{RAT}}, \geq_{\mathcal{RAT}}\}$,
$\mathcal{D}_{\mathcal{RAT}} = (\mathbb{Q}, \{+\}, \{=, <, >, \leq, \geq\})$, where
$+, =, <, >, \leq, \geq$ are defined in the usual way.

$X = X^{rat} = \{u, v\}$.

The constraint solver $CS_{\mathcal{FD}}$ is based on an arc consistency algorithm, $CS_{\mathcal{RAT}}$ uses the simplex algorithm for solving linear (in-)equations and a delay technique for nonlinear constraints.
We assume that the functions *ask* and *tell* of the constraint solvers are defined according to the underlying algorithms.

The **projection functions** of $CS_{\mathcal{FD}}$ and $CS_{\mathcal{RAT}}$ are defined by conversion functions $cast_{\nu,\mu}$ and projection functions of the constraint systems wrt themself:
$proj^{\nu \to \mu}(x, C^{\nu}) = cast_{\nu,\mu}(proj^{\nu \to \nu}(x, C^{\nu}))$ for every $\mu, \nu \in \{\mathcal{FD}, \mathcal{RAT}\}$ and $x \in X^{rat}$.
Because of space limitations the formal definitions of these functions are omitted and only informal descriptions are given.

- projection function of $\mathcal{FD}$ wrt $\mathcal{FD}$:
  $proj^{\mathcal{FD} \to \mathcal{FD}}(x, C^{\mathcal{FD}})$ computes the projection of the constraint store $C^{\mathcal{FD}}$ wrt a variable $x \in X^{rat}$. The result is an $\in_{\mathcal{FD}}$-relation between the variable x and the set of admissible values of the domain of x.

- projection function of $\mathcal{FD}$ wrt $\mathcal{RAT}$:
  The projected $\in_{\mathcal{FD}}$-constraints are converted into constraints $r_\alpha \leq_{\mathcal{RAT}} x \leq_{\mathcal{RAT}} r_\beta$. These constraints describe the maximal intervals of admissible values. The intervals are bounded by admissible values.

- projection function of $\mathcal{RAT}$ wrt $\mathcal{RAT}$:
  $proj^{\mathcal{RAT} \to \mathcal{RAT}}(x, C^{\mathcal{RAT}})$ computes the projection of the constraint store $C^{\mathcal{RAT}}$ wrt a variable $x \in X^{rat}$. The result of the projection are $\{<_{\mathcal{RAT}}, >_{\mathcal{RAT}}, \leq_{\mathcal{RAT}}, \geq_{\mathcal{RAT}}\}$-relations between the variable x and the intervals of admissible values of the domain of x.

- projection function of $\mathcal{RAT}$ wrt $\mathcal{FD}$:
  Constraints which express $\{\leq_{\mathcal{RAT}}, \geq_{\mathcal{RAT}}, >_{\mathcal{RAT}}, <_{\mathcal{RAT}}\}$-relations between a variable and a number are converted into $\in_{\mathcal{FD}}$-relation, i.e. constraints of the constraint domain of $\mathcal{FD}$.

Moreover, the following **conjunction of constraints** which is to be solved is given; the names $c_1, \ldots, c_6$ are assigned to the constraints.

$$
\begin{array}{ll}
c_1: (& (u \in_{\mathcal{FD}} \{1,2,3\})\wedge \\
c_2: & (v \in_{\mathcal{FD}} \{1,2,3\})\wedge \\
c_3: & (u >_{\mathcal{FD}} v)\wedge \\
c_4: & (u >_{\mathcal{RAT}} v)\wedge \\
c_5: & (v \geq_{\mathcal{RAT}} 2)\wedge \\
c_6: & (u +_{\mathcal{RAT}} v \leq_{\mathcal{RAT}} 6)\ ).
\end{array}
$$

Overloaded function symbols and predicate symbols must be detected (analysis). The conjunction of constraints which is to be solved must be converted by the function *Simplify*. We assume that this has already been done in our example.

Derivation:

The given conjunction of constraints must be converted into disjunctive normal form. Then a set is built of each conjunction of constraints of this disjunction. So the overall configuration and a **first** derivation step are the following, we choose c_1 first to be propagated (Item 2(a)i of Sect.3.3 occurs):

$$
\begin{array}{l}
\{c_1, c_2, c_3, c_4, c_5, c_6\} \odot (\bigwedge_{j \in \{\mathcal{FD},\mathcal{RAT}\}} C^j) \\
\longrightarrow_l \\
\{(1 \leq_{\mathcal{RAT}} u \leq_{\mathcal{RAT}} 3), (1 \leq_{\mathcal{RAT}} v \leq_{\mathcal{RAT}} 6), c_2, c_3, c_4, c_5, c_6\} \odot ({C'}^{\mathcal{FD}} \wedge C^{\mathcal{RAT}}),
\end{array}
$$

where

- $l = \mathcal{FD}$ because $c_1 = (u \in_{\mathcal{FD}} \{1,2,3\})$, and hence, $c_1 \in Cons(\mathcal{FD})$.
- $C^{\mathcal{FD}} = C^{\mathcal{RAT}} = (id \| true)$.
- $tell(c_1, C^{\mathcal{FD}}) = (true_{changed}, {C'}^{\mathcal{FD}})$.
- $\begin{aligned}[t] proj^{\mathcal{FD}\to\mathcal{RAT}}(u, {C'}^{\mathcal{FD}}) &= cast_{\mathcal{FD},\mathcal{RAT}}(proj^{\mathcal{FD}\to\mathcal{FD}}(u, {C'}^{\mathcal{FD}})) \\ &= cast_{\mathcal{FD},\mathcal{RAT}}(u \in_{\mathcal{FD}} \{1,2,3\}) \\ &= (1 \leq_{\mathcal{RAT}} u \leq_{\mathcal{RAT}} 3). \end{aligned}$
- $proj^{\mathcal{FD}\to\mathcal{RAT}}(v, {C'}^{\mathcal{FD}}) = (1 \leq_{\mathcal{RAT}} v \leq_{\mathcal{RAT}} 6)$.

Second derivation step (Item 2(a)i, Sect.3.3); we choose $c_5 \in Cons(\mathcal{RAT})$:

$$
\begin{array}{l}
\{(1 \leq_{\mathcal{RAT}} u \leq_{\mathcal{RAT}} 3), (1 \leq_{\mathcal{RAT}} v \leq_{\mathcal{RAT}} 6), c_2, c_3, c_4, c_5, c_6\} \odot ({C'}^{\mathcal{FD}} \wedge C^{\mathcal{RAT}}) \\
\longrightarrow_{\mathcal{RAT}} \\
\{(1 \leq_{\mathcal{RAT}} u \leq_{\mathcal{RAT}} 3), (1 \leq_{\mathcal{RAT}} v \leq_{\mathcal{RAT}} 6), c_2, c_3, c_4, \\
\ (u \in_{\mathcal{FD}} \{1,2,3,4,5,6\}), (v \in_{\mathcal{FD}} \{2,3,4,5,6\}), c_6\} \odot (\bigwedge_{j \in \{\mathcal{FD},\mathcal{RAT}\}} {C'}^j),
\end{array}
$$

where

- $tell(c_5, C^{\mathcal{RAT}}) = (true_{changed}, {C'}^{\mathcal{RAT}})$.

- $proj^{\mathcal{RAT}\to\mathcal{FD}}(u, {C'}^{\mathcal{RAT}}) = (u \in_{\mathcal{FD}} \{1,2,3,4,5,6\})$.
- $proj^{\mathcal{RAT}\to\mathcal{FD}}(v, {C'}^{\mathcal{RAT}}) = (v \in_{\mathcal{FD}} \{2,3,4,5,6\})$.

Third derivation step (Item 2(a)ii); we choose constraint $(u \in_{\mathcal{FD}} \{1,2,3,4,5,6\}) \in Cons(\mathcal{FD})$:

$\{(1 \le_{\mathcal{RAT}} u \le_{\mathcal{RAT}} 3), (1 \le_{\mathcal{RAT}} v \le_{\mathcal{RAT}} 6), c_2, c_3, c_4,$
$(u \in_{\mathcal{FD}} \{1,2,3,4,5,6\}), (v \in_{\mathcal{FD}} \{2,3,4,5,6\}), c_6\} \odot (\bigwedge_{j\in\{\mathcal{FD},\mathcal{RAT}\}} {C'}^{j})$
$\longrightarrow^{\mathcal{FD}}$
$\{(1 \le_{\mathcal{RAT}} u \le_{\mathcal{RAT}} 3), (1 \le_{\mathcal{RAT}} v \le_{\mathcal{RAT}} 6), c_2, c_3, c_4,$
$true, (v \in_{\mathcal{FD}} \{2,3,4,5,6\}), c_6\} \odot (\bigwedge_{j\in\{\mathcal{FD},\mathcal{RAT}\}} {C'}^{j})$,
where

- $tell((u \in_{\mathcal{FD}} \{1,2,3,4,5,6\}), {C'}^{\mathcal{FD}}) = (true_{redundant}, {C'}^{\mathcal{FD}})$.

Further derivation step (Item 3c, Sect.3.3):

$\{(1 \le_{\mathcal{RAT}} u \le_{\mathcal{RAT}} 3), (1 \le_{\mathcal{RAT}} v \le_{\mathcal{RAT}} 6), c_2, c_3, c_4,$
$true, (v \in_{\mathcal{FD}} \{2,3,4,5,6\}), c_6\} \odot (\bigwedge_{j\in\{\mathcal{FD},\mathcal{RAT}\}} {C'}^{j})$
$\longrightarrow^{0}$
$\{(1 \le_{\mathcal{RAT}} u \le_{\mathcal{RAT}} 3), (1 \le_{\mathcal{RAT}} v \le_{\mathcal{RAT}} 6), c_2, c_3, c_4,$
$(v \in_{\mathcal{FD}} \{2,3,4,5,6\}), c_6\} \odot (\bigwedge_{j\in\{\mathcal{FD},\mathcal{RAT}\}} {C'}^{j})$.
In this step, the configuration is simplified.

Further derivation steps follow. The overall result and normal form of the derivation sequence is the following:
$\{true\} \odot (C^{\mathcal{FD}}_{new} \wedge C^{\mathcal{RAT}}_{new})$.

At this, the active constraint stores contain substitutions and constraints which describe the solution of the given conjunction of constraints. We project the active store $C^{\mathcal{FD}}_{new}$ wrt $\mathcal{FD}$ and the variables u and v to see the variable assignments:
$\bigwedge_{x\in\{u,v\}} proj^{\mathcal{FD}\to\mathcal{FD}}(x, C^{\mathcal{FD}}_{new}) = (u \in_{\mathcal{FD}} \{3\}) \wedge (v \in_{\mathcal{FD}} \{2\})$.

In this example the projection of the constraint store wrt the variables is sufficient to get the solution but, in the general case, the application of search heuristics like backtracking is necessary.