

Cooperating Constraint Solvers

Petra Hofstedt

Department of Computer Science, Berlin University of Technology

Abstract. We propose a general scheme for the cooperation of different constraint solvers. On top of a uniform interface for constraint we stepwise develop reduction systems which describe the behaviour of an overall combined system. The modularity of our definitions of reduction relations at different levels allows the definition of cooperation strategies for the solvers according to the current requirements such that our overall system forms a general framework for cooperating solvers.

1 Constraint Systems and Constraint Solvers

The combination of several constraint solving techniques enables to solve problems that none of the single solvers can handle alone (for examples see [2, 4, 5]). Thus, we developed an open and very flexible combination mechanism.

A *signature* $\Sigma = (S, F, R; ar)$ is defined by a set S of sorts, a set F of function symbols, a set R of predicate symbols, and an arity function $ar : F \cup R \rightarrow S^*$. 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 1. an S -sorted family of nonempty carrier sets \mathcal{D}^s , 2. a family of functions $f^{\mathcal{D}}$, and 3. a family of predicates $r^{\mathcal{D}}$, appropriate to F and R . $X = \bigcup_{s \in S} X^s$ is a many sorted set of variables. A *constraint system* is a tuple $\zeta = \langle \Sigma, \mathcal{D} \rangle$, a *constraint over* Σ is a string of the form $r \ t_1 \dots t_m$ where $r \in R^{s_1 \times \dots \times s_m}$ and $t_i \in \mathcal{T}(F, X)^{s_i}$ (, i.e. t_i is a term over F of sort s_i with variables from X). The set of constraints over Σ is denoted by $Cons(\Sigma)$. Let $\forall\psi$ resp. $\exists\psi$ denote the universal closure and the existential closure, respectively. $\exists_{\tilde{Y}}\psi$ denotes the existential closure of formula ψ except for the variables occurring in the sequence \tilde{Y} of variables. Let \mathcal{D} be a Σ -structure, and let Φ be a constraint conjunction over Σ . Let $var(\Phi)$ denote the set of variables of Φ . A *solution* of Φ in \mathcal{D} is a valuation $\sigma : V \rightarrow \bigcup_{s \in S} \mathcal{D}^s$ of a finite set V of variables, $var(\Phi) \subseteq V$, such that $(\mathcal{D}, \sigma) \models \forall\Phi$ holds. Solving Φ means finding out whether there exists a solution for Φ or not. A *constraint solver* consists of a collection of operations which can be used to solve and to transform constraints of a constraint system. A solver works on a constraint store $C \in CStore$ which, in the following, consists of a disjunction of constraint conjunctions. C has the property that it is satisfiable in the corresponding structure \mathcal{D} , i.e. $\mathcal{D} \models \exists C$ holds.

2 A Uniform Interface for Constraint Solvers

To enable a cooperation of constraint solvers to solve a given problem, the solvers need to exchange information. We want to enable them to communicate in such a way that a very tight cooperation is possible.

Let L be the set of indices of constraint systems, $\mu, \nu \in L$. Let to each constraint system a constraint solver be assigned. The following functions built our uniform interface for constraint solvers:

1. $tell: Cons(\Sigma_\nu) \times CStore_\nu \rightarrow \{true_{changed}, true_{redundant}, false\} \times CStore_\nu$
2. $proj: \mathcal{P}(X) \times CStore_\nu \rightarrow CStore_\nu$
3. $proj^{\nu \rightarrow \mu}: \mathcal{P}(X) \times CStore_\nu \rightarrow CStore_\mu$

1. The (partial) function $tell$ is due to constraint satisfaction, i.e. an operation which is usually offered by constraint solvers. $tell$ adds a constraint $c \in Cons(\Sigma_\nu)$ to a constraint store $C \in CStore_\nu$ if the conjunction of c and C is satisfiable, i.e. if $\mathcal{D} \models \exists(C \wedge c)$ holds. We require $tell$ to be defined as follows:

if $tell(c, C) = (true_{redundant}, C)$,	then $\mathcal{D} \models \forall(C \rightarrow c)$,
if $tell(c, C) = (true_{changed}, C')$, where $\mathcal{D} \models \forall((C \wedge c) \leftrightarrow C')$,	then $\mathcal{D} \models \exists(C \wedge c)$,
if $tell(c, C) = (false, C)$	then $\mathcal{D} \not\models \exists(C \wedge c)$.

2. The function $proj$ is due to the operation constraint projection of constraint solvers. Usually, the aim of projecting a constraint store $C \in CStore_\nu$ wrt a sequence \tilde{Y} (with $Y \subseteq X$) of variables which occur in C is to find a disjunction C' of conjunctions of constraints which is equivalent to $\exists_{\tilde{Y}} C$ and where the variables which do occur in C but not in \tilde{Y} are eliminated: $\mathcal{D}_\nu \models \forall(\exists_{\tilde{Y}} C \leftrightarrow C')$. However, since sometimes it is not possible to compute C' or it is not possible to compute it efficiently, we require $proj$ to be defined as follows:

$proj(Y, C) = C'$, where $Y \subseteq X$ and $\mathcal{D}_\nu \models \forall(\exists_{\tilde{Y}} C \rightarrow C')$.

3. Since we want to use projections for information exchange between the different constraint solvers, we need a function which projects a constraint store C^ν wrt another constraint system ζ_μ , $\mu \in L \setminus \{\nu\}$. $proj^{\nu \rightarrow \mu}$ can be defined by means of $proj$ and a conversion function. Thus each single constraint solver can be regarded as black box solver equipped with a projection function which allows the projection of the constraint store wrt a set of variables. These black box solvers are extended by functions for converting projections wrt other constraint systems.

Example 1. Consider a solver CS_{rat} for arithmetic constraints over rational numbers, and a solver CS_{FD} of a finite domain constraint system ζ_{FD} .

The projection functions $proj$ and $proj^{FD \rightarrow rat}$ of CS_{FD} could work as follows:

$proj(x, C^{FD}) = x \in_{FD} \{4, 5, 6\}$, and

$proj^{FD \rightarrow rat}(x, C^{FD}) = ((x \geq 4) \wedge (x \leq 6))$, where

$C^{FD} = ((y = 3) \wedge (x > y) \wedge (x \in_{FD} \{2, 3, 4, 5, 6\}))$.

The function $tell$ of CS_{rat} could yield the following results:

$tell((x = 4), C) = (true_{changed}, C')$, where $C = true$ and $C' = (x = 4)$,

$tell((x > 3), C') = (true_{redundant}, C')$, and $tell((x < 3), C') = (false, C')$. \square

Clearly, the projection functions $proj$ and $proj^{\nu \rightarrow \mu}$, $\nu, \mu \in L$, must be defined in such a way that, first, projecting a constraint store wrt another constraint

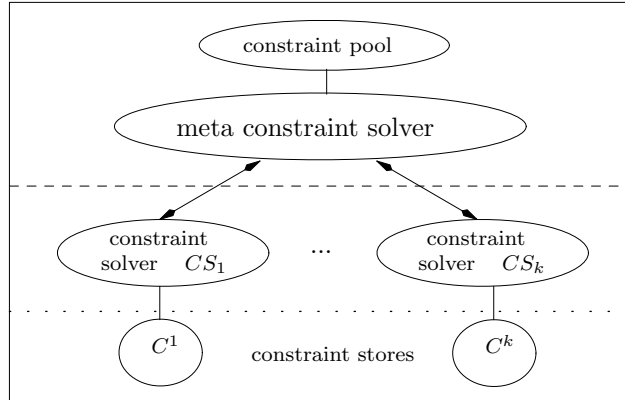


Fig. 1. Architecture of the overall system

system, no solutions of the constraints of the store are lost and, second, that a projection of a constraint store implies previous projections of this (at a previous time less restricted) store. We call these required properties *soundness* and *monotonicity*, a formal description is left out because of space limitations. We require the functions $tell$, $proj$, and $proj^{\nu \rightarrow \mu}$, $\nu, \mu \in L$, to be computable.

3 Combination of Constraint Solvers

The Architecture. Figure 1 shows the architecture of our overall system for cooperating constraint solvers. To every individual solver CS_ν , a constraint store C^ν is assigned. The *meta constraint solver* coordinates the work of the different individual solvers. The meta solver manages the *constraint pool*. Initially, the constraint pool contains the constraints of the constraint conjunction Φ which we want to solve. The meta constraint solver takes constraints from the constraint pool and passes them to the constraint solvers of the corresponding constraint domains (step 1). Each of the individual constraint solvers is able to handle a subset of the set of constraints of the constraint pool independently of the other solvers, the individual solvers propagate the received constraints to their stores by $tell$ (step 2). The meta constraint solver manages the exchange of information between the individual solvers. It forces them to extract information from their constraint stores by $proj^{\nu \rightarrow \mu}$. This information is added by the meta constraint solver to the constraint pool (step 3). The procedure of steps 1-3 is repeated until the pool contains either the constraint *false* or the constraint *true* only. If the constraint pool contains *false* only, then the initially given conjunction Φ of constraints is unsatisfiable. If the pool contains *true* only, then the system could not find a contradiction. Solutions of Φ can be retrieved from the current constraint stores. Using the described mechanisms, each individual solver deals with more information than only that of its associated constraints of Φ .

Syntax. To allow to solve a conjunction of constraints, where every constraint may contain function symbols and predicate symbols of different constraint systems, it is necessary to detect overloaded symbols by analysis and to convert the constraint conjunction into a conjunction such that every constraint is defined by symbols of exactly one constraint system (by flattening as usually).

Operational Semantics. We describe the operational semantics of our system by means of a reduction relation for *overall configurations*. An overall configuration \mathcal{H} consists of a *formal disjunction* $\dot{\bigvee}_{i \in \{1, \dots, m\}} \mathcal{G}_i$ of configurations \mathcal{G}_i . Formal disjunction $\dot{\vee}$ is commutative. A *configuration* $\mathcal{G} = \mathcal{P} \odot \bigwedge_{\nu \in L} C^\nu$ corresponds to the architecture of the overall system (Fig.1). It consists of the constraint pool \mathcal{P} which is a set of constraints which we want to solve and the conjunction $\bigwedge_{\nu \in L} C^\nu$ of constraint stores.

We lift the application of the functions *tell* resp. *proj* $^{\nu \rightarrow \mu}$ to the level of overall configurations and define the two *basic relations* *prop* resp. *put_proj* which are the basis of the stepwise definition of the operational semantics. We define *strategies for cooperating constraint solvers*, i.e. reduction systems for overall configurations using these basic relations. In general, in one derivation step one or more configurations \mathcal{G}_i , $i \in \{1, \dots, m\}$, are rewritten by a formal disjunction $\mathcal{H}\mathcal{G}_i$ of configurations:

$\begin{aligned} OConf1 &= \mathcal{H}_1 \dot{\vee} \mathcal{G}_1 \dot{\vee} \dots \dot{\vee} \mathcal{H}_i \dot{\vee} \mathcal{G}_i \dot{\vee} \dots \dot{\vee} \mathcal{H}_m \dot{\vee} \mathcal{G}_m \dot{\vee} \mathcal{H}_{m+1} \implies \\ OConf2 &= \mathcal{H}_1 \dot{\vee} \mathcal{H}\mathcal{G}_1 \dot{\vee} \dots \dot{\vee} \mathcal{H}_i \dot{\vee} \mathcal{H}\mathcal{G}_i \dot{\vee} \dots \dot{\vee} \mathcal{H}_m \dot{\vee} \mathcal{H}\mathcal{G}_m \dot{\vee} \mathcal{H}_{m+1} \end{aligned}$
--

Thus, first, we define a derivation relation for configurations and, based on this, we define a derivation relation for overall configurations.

Step 1. Definition of a derivation relation for configurations (production level). The simplest possibility to define a derivation step $\mathcal{G}_i \rightarrow \mathcal{H}\mathcal{G}_i$ is to chose (nondeterministically) exactly one constraint $c \in \text{Cons}(\Sigma_\nu)$, $\nu \in L$, from the constraint pool of \mathcal{G}_i and to propagate it to its associated constraint store C^ν building a new configuration \mathcal{G}'_i (using the basic relation *prop*). This is followed by projections of the newly built constraint store C'^{ν} wrt other constraint systems building the new overall configuration $\mathcal{H}\mathcal{G}_i$ (using *put_proj*).

There are many other possibilities to define a strategy for the production level. We are able to let the solvers work in parallel as well as to fix the order of the constraint systems or constraints itself which are propagated next which enables to regard choice heuristics, for example to delay particular constraints, as for naive solving nonlinear constraints.

Step 2. Defining a derivation relation for overall configurations (application level). A derivation step $OConf1 \implies OConf2$ for overall configurations is defined on the basis of the derivation relation for configurations (at production level). There are as well many possibilities: for example, we may define a derivation step such that the derivation of exactly one configuration or the derivation of several configurations in parallel or concurrently is allowed.

Using this *two-step frame* different reduction systems which realize different derivation strategies for the derivation of an *initial overall configuration* to normal form can be described. An initial overall configuration is a configuration $\mathcal{G}_0 = \mathcal{P}_\Phi \odot \bigwedge_{\nu \in L} C^\nu_0$, where the constraint pool \mathcal{P}_Φ contains the constraints of

the conjunction Φ which we want to solve and all constraint stores C_0^ν , $\nu \in L$, are empty, i.e. they contain the constraint *true* only.

4 Conclusion and Related Work

We shortly presented a general scheme for cooperating constraint solvers. A uniform interface for the solvers allows to formally specify the information exchange between them. Because of this information exchange the overall combined system is able to solve constraint conjunctions which the single solvers are not able to handle. The modularity of our definitions of the basic relations and of the derivation relations at production level as well as at application level allows the definition of derivation strategies for cooperating solvers according to the current requirements, like properties of the particular constraint solvers as well as properties of the underlying software and hardware. Since our approach allows the integration of constraint solvers of very different constraint systems it is possible to integrate different host languages into the system by treating them as constraint solvers. In [1] we have shown the integration of a functional logic language. This new point of view on the host language of such a system and the possibility to define tight cooperation strategies according to the current requirements allow to specify a wide range of systems of cooperating solvers such that our overall system forms a general framework for cooperating solvers.

Cooperating solvers have been investigated from different points of view [2–6]. Usually this are very specialized approaches and their cooperation strategies are fixed. Our approach allows the definition of similar strategies and even a finer grained definition of strategies according to the current requirements. As far as for the mentioned approaches the form of constraints exchanged between the solvers is given, we are able to express this as well by means of our interface functions. For example, the main idea behind the combination approach in [6] is a mechanism which controls variable equality sharing which is an instance of information exchange by projections as done in our approach.

References

1. P. Hofstedt. A functional logic language as hostlanguage for a system of combined constraint solvers. In R. Echahed, editor, *8th International Workshop on Functional and Logic Programming*, pages 119–132. Grenoble, France, 1999.
2. H. Hong. Confluency of cooperative constraint solvers. Technical Report 94-08, Research Institute for Symbolic Computation, Linz, Austria, 1994.
3. E. Monfroy. *Solver Collaboration for Constraint Logic Programming*. PhD thesis, Centre de Recherche en Informatique de Nancy. INRIA-Lorraine, 1996.
4. M. Rueher. An architecture for cooperating constraint solvers on reals. In A. Podelski, editor, *Constraint Programming. Châtillon Spring School 1994. Selected Papers*, volume 910 of *LNCS*, pages 231–250. Springer-Verlag, 1995.
5. M. Rueher and C. Solnon. Concurrent cooperating solvers over reals. *Reliable Computing*, 3:3:325–333, 1997.
6. C. Tinelli and M.T. Harandi. Constraint logic programming over unions of constraint theories. *The Journal of Functional and Logic Programming*, Article 6, 1998.