Bull. Nov. Comp. Center, Comp. Science, 23 (2005), 101–111

Constraint-based analysis of composite solvers

E. Petrov, É. Monfroy

Abstract. Cooperative constraint solving is an area of constraint programming that studies the interaction between constraint solvers with the aim of discovering the interaction patterns that amplify the positive qualities of individual solvers. Automatisation and formalisation of such studies is an important issue of cooperative constraint solving.

In this paper, we present a constraint-based analysis of composite solvers that integrates reasoning about the individual solvers and the processed data. The idea is to approximate this reasoning by resolution of set constraints on the finite sets representing the predicates that express all the necessary properties. We illustrate application of our analysis to two important cooperation patterns: loop and deterministic choice.

1. Introduction

Cooperative constraint solving is an area of constraint programming that studies interaction between constraint solvers with the aim of discovering the interaction patterns that amplify the positive qualities of individual solvers. Papers [2, 7, 9, 10, 14] describe some examples of successful cooperative constraint solving systems.

At present, successful patterns of cooperation are detected with the sole help of intuition, feelings, experience, and experiments with software engineering tools supporting cooperative constraint solving such as [8, 11, 12, 13]. One cannot make such experiments systematic without a mathematical framework for analysis of cooperative constraint solving systems ("composite solvers").

An important practical application for such a framework is integration and development of domain-specific software. The expected applications for this analysis are detection of inconsistencies in specifications of individual software packages and construction of the specifications for the packages needed in order to meet the requirements on the overall functionality. The COCONUT [3] project is an example of such a software development project in the domain of numerical optimization and constraint programming.

The authors of [5] illustrate practical importance of analysis of composite solvers with examples from interval constraint programming. They point out two aspects of this analysis: (a) detection of good cooperation strategies subject to expectations from the composite solver and the individual solvers in hand; (b) reasoning about the properties of individual solvers subject to the expectations from the composite solver. In this paper, we focus on the latter aspect.

Certain properties of composite solvers are expressible in terms of the properties of the processed *data*. One can study the properties of this sort using the frameworks of software verification, programming logics, model checking program analysis. In order to reason about the properties of *individual solvers*, one needs a new kind of analysis.

In this paper, we present a constraint-based analysis of composite solvers that integrates reasoning about the individual solvers and the processed data. The idea is to approximate this reasoning by resolution of set constraints on the finite sets representing the predicates that express all the necessary properties. We illustrate application of our analysis to two important cooperation patterns: deterministic choice and loop.

Before going further we give a quick motivational example.

Example 1. Motivation.
Suppose that we, rather "engineers" than experts in global optimization, have in hand a library of numerical algorithms that includes the standard methods for local and global optimization, different tests, etc. and develop a software for minimization of quadratic functions (quadratic programming). Algorithm 1 specifies the composite solver that we have built with the intention to avoid the costly exhaustive search in the case of convex objective functions.

---

if f is convex **then**
  solve $x = \arg\min_{x \geq 0} f$ by the method of steepest descent
else
  solve $x = \arg\min_{x \geq 0} f$ by the exhaustive search
end if

A naive composite solver for quadratic programming.

Our logic is as follows: if the graph of the objective is like a tea-cup ("convex"), then we use the steepest descent to go to its lowest point; otherwise we do the exhaustive search. Will this work? Well, sometimes: we have not thought of the objectives that are convex and unbounded from below in the orthant $x \geq 0$, e.g. $f(x_1, x_2) = (x_1 - x_2)^2 - x_1$. The method of steepest descent may fail to terminate in this case.

How can we detect similar situations automatically? How do we change a composite solver in order that it work correctly? In this paper, we propose a formalism suitable for this kind of analysis.

The paper is structured as follows. In Section 2, we introduce the basic notions of the paper and describe the operation of composite solvers. In

Section 3, we describe a technique for reasoning about composite solvers in terms of constraints on finite sets. In Section 4, we explain how to express certain properties of the composite solvers in terms of set constraints. Section 5 concludes the paper.

2. Solvers, contexts, operation model

In this section, we introduce the basic notions of the paper and describe the operation of composite solvers.

A set of individual solvers that interact and exchange some data through a shared data store is called a *composite solver*. We number the individual solvers by integers from 1 to n. The content of the data store, called *context*, is divided into the set of the individual solvers that "are called" and *application specific data*. In the constraint programming framework, the application specific data are usually specifications in some declarative language. The set of all contexts is denoted by $\mathcal{C}$.

The operation of the composite solver is divided into ticks. At the beginning of a tick, every solver s checks if it is called. If yes, it may change some part of the context. Otherwise, it does nothing. The initial context is provided by the user of the composite solver. Thus our composite solvers are either sequential or synchronous bulk parallel systems.

A solver s determines a transformation $F_s : \mathcal{C} \to \mathcal{C}$ of the contexts at the beginning of the ticks into the contexts at the end of the ticks. The synchronous modifications of the data store must be coherent, that is, $F_s(c) = F_{s'}(c)$ for any context c indicating that solvers s and s' are called.

A context is *feasible* if it is generated from the initial context by some sequence of transformations F_s's where s's are some solvers.

Now we proceed to the description of our constraint-based formalism.

3. From composite solvers to finite sets

In this section, we describe a technique for reasoning about composite solvers in terms of constraints on finite sets. We axiomatize the operation model from Section 2 in the first order logic and view the axioms as set constraints on interpretation of the symbols involved therein. Since the exact solutions to these constraints may be (non-constructible explicitly) infinite sets, we solve our set constraints approximately modulo a finite set of clusters of the contexts $\mathcal{C}$. We assume that the reader is familiar with the generic concepts of a constraint and constraint satisfaction.

The axioms in question are written in terms of a constant symbol c_0 denoting the initial context, a unary predicate symbol p denoting the set of feasible contexts and unary function symbols f_s's denoting the transformations F_s. The axioms are plain and open (n is the number of individual solvers):

$$p(c_0); \tag{1}$$
$$\forall c\, p(c) \wedge f_s(c) = c' \Longrightarrow p(c') \quad s = 1, \ldots, n; \tag{2}$$
$$\forall c' \exists c\, p(c') \wedge c' \neq c_0 \Longrightarrow \big(c' = f_1(c) \vee \ldots \vee c' = f_n(c)\big) \wedge p(c). \tag{3}$$

The axiom (1) states that the initial context is feasible. The n axioms (2) say that the transformations F_s's map feasible contexts onto themselves. The axiom (3) states that every feasible context except the initial one has a feasible pre-image under some transformation F_s.

The axioms (1)–(3) are nothing else but constraints on the interpretations $\dot{c_0}$, $\dot{p}$ and $\dot{f_s}$'s, in the model-theoretical sense, of the symbols c_0, p and f_s's. One can write down these constraints as follows:

$$\texttt{fun}\left(\dot{f_1}\right), \ldots, \texttt{fun}\left(\dot{f_n}\right); \tag{4}$$
$$\dot{p} = \{\dot{c_0}\} \cup \texttt{img}\left(\dot{f_1}, \dot{p}\right) \cup \cdots \cup \texttt{img}\left(\dot{f_n}, \dot{p}\right). \tag{5}$$

The domains of $\dot{c_0}$, $\dot{p}$ and $\dot{f_s}$'s consist of the contexts $\mathcal{C}$, of all the subsets of $\mathcal{C}$ and, respectively, of all binary relations on $\mathcal{C}$. The symbols $=$ and $\cup$ denote equality and union of subsets of $\mathcal{C}$. The symbol `fun` denotes the constraint "is a function from $\mathcal{C}$ to $\mathcal{C}$". The symbol `img` denotes the image of a subset of $\mathcal{C}$ under a binary relation on $\mathcal{C}$.

Since many exact solutions to the constraints (4)–(5) involve infinite sets, we sacrifice precision for tractability and group the individual contexts from $\mathcal{C}$ into finitely many clusters called *context properties*. The set of all clusters is denoted by $\mathcal{C}^\star$. A binary relation on $\mathcal{C}^\star$ whose domain is $\mathcal{C}^\star$ is called an *abstract solver*. We assume that binary relations are collections of ordered pairs.

A context property $c^\star$ *approximates* a context c, iff $c \in c^\star$. A set $P^\star$ of context properties *approximates* a set $P \subseteq \mathcal{C}$, iff every context from P is approximated by some context property from $P^\star$. An abstract solver $R^\star$ *approximates* a binary relation $R \subseteq \mathcal{C}^2$, iff every element from R is componentisely approximated by some element from $R^\star$.

Example 2. Hull Consistency in sharpness analysis.
Consider the Hull Consistency (HC) algorithm [1] from interval constraint programming. Given a set i of interval constraints, this algorithm computes a box b that bounds the set $\texttt{sol}(i)$ of the solutions to i. Let the context specify the constraints i and the box b.

The well-known fact about the HC algorithm is that it bounds the set $\texttt{sol}(i)$ sharply, i.e., b cannot be improved without losing a solution to i, if i

has an acyclic constraint graph (see [6]). We can express this fact in terms of the context properties "i has an acyclic constraint graph" (abbreviated $\mathtt{tree}$) and "b is the convex hull of $\mathtt{sol}(i)$" (abbreviated $\mathtt{ok}$) by the abstract solver $HC^\star = \{(\mathtt{tree},\mathtt{ok})\ \ (\mathcal{C},\mathcal{C})\ \ (\mathtt{ok},\mathtt{ok})\}$. For example, $(\mathtt{tree},\mathtt{ok}) \in HC^\star$ means that the HC algorithm bounds $\mathtt{sol}(i)$ sharply, if i has an acyclic constraint graph.

Proposition 1. ***Correctness*** *If $\dot{c}_0 \in \mathcal{C}^\star$, $\dot{p} \subseteq \mathcal{C}^\star$ and abstract solvers $\dot{f}_s$'s approximate some solution to the constraints (4)–(5), then they satisfy these same constraints with respect to the following definition of $=$, $\cup$, $\mathtt{fun}$, $\mathtt{img}$:*

$$\begin{aligned}
\mathtt{fun}(R^\star) &\iff R^\star \textit{ is an abstract solver};\\
\mathtt{img}(R^\star, P^\star) &= \textit{the image of } P^\star \textit{ under } R^\star;\\
P^\star \cup Q^\star &= \textit{the standard union of subsets of } \mathcal{C}^\star;\\
P^\star = Q^\star &\iff \textit{the standard equality of subsets of } \mathcal{C}^\star.
\end{aligned}$$

In practice the constraints (4)–(5) are joined to (some of) the constraints

$$\dot{f}_1 = F_1^\star, \ldots, \dot{f}_n = F_n^\star \tag{6}$$

specifying the abstract solvers. We build these $F_s^\star$'s using two databases that contain patterns of the individual solvers and the relation of logical equivalence on the set of properties of the processed data.

A *pattern* of an individual solver is a collection of *rules* of the form "pre-condition $\rightarrow$ post-condition" that have, as common formal parameters, the processed data and the called solvers. The pre- and post-condition are conjunctions of atomic formulas containing the formal parameters of the pattern. The formal parameters corresponding to the data not modified by the solver can be marked as "read-only".

The fact that a solver is called is expressed by the unary predicate symbol $\mathtt{do}$; the interpretation of other predicate symbols is arbitrary. The pre- and post-conditions in the pattern of a solver s are of the form $\mathtt{do}(s) \wedge C$ and the symbol $\mathtt{do}$ does not occur in the conjunction C. Thus parallelism is not actually allowed.

Let $\pi_1, \ldots, \pi_n$ be the patterns of the individual solvers with instantiated formal parameters. The context properties are conjunctions $\mathtt{do}(s) \wedge C$, — where $s = 1, \ldots, n$ and C is a conjunction of the atomic formulas from π_1, $\ldots$, π_n,— that are not equivalent to the false conjunction. We assume that two equivalent conjunctions are the same object.

The image of a context property $c^\star$ under the abstract solver $F_s^\star$ corresponding to a pattern π_s is built as follows. Let $c^\star = \mathtt{do}(s') \wedge C_{\mathtt{ro}} \wedge C_{\mathtt{rw}}$ such that every conjunct in $C_{\mathtt{rw}}$ contains a value taken by some non read-only formal parameter of π_s and $C_{\mathtt{ro}}$ contains all the other conjuncts from $c^\star$ except $\mathtt{do}(s')$.

If $s \neq s'$ then $\mathtt{img}\,(F_s^\star, \{c^\star\}) = \{c^\star\}$. Otherwise

$$\mathtt{img}\,(F_s^\star, \{c^\star\}) = \bigwedge \{\{C_{\mathtt{ro}}\}\} \cup \{\mathtt{rhs}(c_1^\star, \pi_s) | c_1^\star \text{ is implied by } c^\star\}\,,$$

where each set of conjunctions is interpreted as the disjunction of its elements, e.g. $\{a, b\} \wedge \{x, y\} = \{a \wedge x, a \wedge y, b \wedge x, b \wedge y\}$, and $\mathtt{rhs}(c_1^\star, \pi_s)$ denotes the set of post-conditions following the pre-condition $c_1^\star$ in the pattern π_s. Finally, $c_1^\star$ "is implied by" $c^\star$ iff $c_1^\star \wedge c^\star$ is equivalent to $c^\star$.

The next section illustrates our approach by several examples.

4. Examples

The examples in this section illustrate application of our approach to two important cooperation patterns: deterministic choice and loop. Our ultimate goal (out of the scope of this paper) is to couple the analysis with the language for specification of composite solvers in the framework of the COCONUT [3] project. In order that the reader can feel our approach better, we provide in Appendix A a complete specification of the set constraints from Section 4.2.

4.1. The naive solver from Section 1

The patterns of the individual solvers from the example in Section 1 are as follows:

$$\mathtt{cnvx?}(\mathtt{ro}(F); S_1, S_2) = \{\mathtt{do}(1) \wedge \mathtt{cnvx}(F) \to \mathtt{do}(S_1), \mathtt{do}(1) \to \mathtt{do}(S_2)\};$$
$$\mathtt{dscnt}(\mathtt{ro}(F), X; S) = \{\mathtt{do}(2) \wedge \mathtt{stCnvx}(F) \to \mathtt{do}(S) \wedge \mathtt{min}(F, X), \mathtt{do}(2) \to \mathtt{do}(S)\};$$
$$\mathtt{glblSrch}(\mathtt{ro}(F), X; S) = \{\mathtt{do}(3) \to \mathtt{do}(S) \wedge \mathtt{min}(F, X)\};$$
$$\mathtt{done}() = \{\mathtt{do}(4) \to \mathtt{do}(4)\}.$$

The symbols `cnvx`, `stCnvx`, `min` denote the properties "is convex", "is strictly convex", "is the global minimizer in the positive orthant". The only non-trivial equivalence is $\mathtt{cnvx}(F) \wedge \mathtt{stCnvx}(F) \equiv \mathtt{stCnvx}(F)$. The read-only parameters are marked by `ro`.

The instantiated patterns are $\mathtt{cnvx?}(f; 2, 3)$, $\mathtt{dscnt}(f, x; 4)$, $\mathtt{glblSrch}(f, x; 4)$, $\mathtt{done}()$. The set of context properties is (we use the notation for disjunctions from Section 3): $\{\mathtt{do}(1),\ \mathtt{do}(2),\ \mathtt{do}(3),\ \mathtt{do}(4)\} \wedge \{\mathtt{cnvx}(f),\ \mathtt{stCnvx}(f),\ \mathtt{min}(f, x),\ \mathtt{true}\} \wedge \{\mathtt{min}(f, x),\ \mathtt{true}\}$. In practice these 24 context properties are numbered and the set constraints involve only their numbers.

From the specifications for $F_1^\star$, $F_2^\star$, $F_3^\star$, $F_4^\star$ generated by the procedure from Section 3, we provide the first one:

$$\mathtt{img}(F_1^\star, \{\mathtt{do}(1) \wedge \mathtt{cnvx}(f)\}) = \{\mathtt{do}(2) \wedge \mathtt{cnvx}(f)\},$$
$$\mathtt{img}(F_1^\star, \{\mathtt{do}(1) \wedge \mathtt{stCnvx}(f)\}) = \{\mathtt{do}(2) \wedge \mathtt{stCnvx}(f)\},$$
$$\mathtt{img}(F_1^\star, \{\mathtt{do}(1) \wedge \mathtt{min}(f, x) \wedge \mathtt{cnvx}(f)\}) = \{\mathtt{do}(2) \wedge \mathtt{min}(f, x) \wedge \mathtt{cnvx}(f)\},$$

$$\texttt{img}(F_1^\star, \{\texttt{do}(1) \wedge \texttt{min}(f,x) \wedge \texttt{stCnvx}(f)\}) = \{\texttt{do}(2) \wedge \texttt{min}(f,x) \wedge \texttt{stCnvx}(f)\},$$
$$\texttt{img}(F_1^\star, \{\texttt{do}(1) \wedge \texttt{min}(f,x)\}) = \{\texttt{do}(2) \wedge \texttt{min}(f,x) \wedge \texttt{cnvx}(f), \texttt{do}(3) \wedge \texttt{min}(f,x)\},$$
$$\texttt{img}(F_1^\star, \{\texttt{do}(1)\}) = \{\texttt{do}(2) \wedge \texttt{cnvx}(f), \texttt{do}(3)\},$$

and $\texttt{img}(F_1^\star, \{c^\star\}) = \{c^\star\}$ for the other context properties $c^\star$.

Solving the constraints (4)–(6) and $\dot{c_0} = \texttt{do}(1)$, we obtain the following approximation for the set of feasible contexts: $\dot{p} = \{\texttt{do}(1), \texttt{do}(2) \wedge \texttt{cnvx}(f), \texttt{do}(3), \texttt{do}(4), \texttt{do}(4) \wedge \texttt{min}(f,x)\}$. Since this $\dot{p}$ contains $\texttt{do}(4)$, we are not sure that our composite solver always finds the minimizer of $f(x)$ subject to $x \geq 0$.

The question "When does our composite solver find the minimizer?" is translated into constraints (4)–(6), "the convexity test is called first" and"the situation after termination is not uncertain" ($\texttt{do}(1)$ "is implied by" $\dot{c_0}$, $\texttt{do}(4) \notin \dot{p}$). Solving these constraints for the initial context $\dot{c_0}$, we obtain the following solutions: $\dot{c_0} = \texttt{do}(1) \wedge \texttt{stCnvx}(f)$, $\dot{c_0} = \texttt{do}(1) \wedge \texttt{min}(f,x) \wedge \texttt{stCnvx}(f)$. This means that the objective has to be strictly convex in order that our solver can find its global minimizer.

4.2. The Simplex method and Hull Consistency

Consider a composite solver that makes cooperate the Simplex method from linear programming and the HC algorithm [1] (a similar composite solver is described, e.g., in [2]). The context specifies some linear, interval and bound constraints, denoted by ℓ, i and, respectively, b. The Simplex method updates b by bounding the solution set $\texttt{sol}(\ell \cup b)$ and calls the HC algorithm, which in its turn updates b by bounding the solution set $\texttt{sol}(i \cup b)$ and calls the Simplex method, and so on until stabilization of b. An important quality of this strategy is that it bounds the solution set $\texttt{sol}(l \cup i \cup b)$ more sharply than the HC algorithm.

The patterns of the individual solvers are as follows:

$$\texttt{cplex}(\texttt{ro}(L), B; S) = \{\texttt{do}(1) \to \texttt{do}(S) \wedge \texttt{ok}(L)\}\,;$$
$$\texttt{hc}(\texttt{ro}(I), B; S) = \{\texttt{do}(2) \to \texttt{do}(S), \texttt{do}(2) \wedge \texttt{tree}(I) \to \texttt{do}(S) \wedge \texttt{ok}(I)\}\,;$$
$$\texttt{same?}(\texttt{ro}(B); S_1, S_2) = \{\texttt{do}(3) \to \texttt{do}(S_1), \texttt{do}(3) \to \texttt{do}(S_2)\}\,;$$
$$\texttt{done}() = \{\texttt{do}(4) \to \texttt{do}(4)\}\,.$$

The symbols $\texttt{ok}$ and $\texttt{tree}$ denote the properties "has the solution set that we can bound sharply", "has an acyclic constraint graph". All the equivalences are trivial.

The instantiated patterns are $\texttt{cplex}(\ell, b; 2)$, $\texttt{hc}(i, b; 3)$, $\texttt{same?}(b; 1, 4)$, $\texttt{done}()$. There are 32 context properties built as follows: $\{\texttt{do}(1), \texttt{do}(2), \texttt{do}(3), \texttt{do}(4)\} \wedge \{\texttt{ok}(\ell), \texttt{ok}(i), \texttt{tree}(i), \texttt{true}\} \wedge \{\texttt{ok}(i), \texttt{tree}(i), \texttt{true}\} \wedge \{\texttt{tree}(i), \texttt{true}\}$.

The specifications for the abstract solvers $F_1^\star$, $F_2^\star$, $F_3^\star$, $F_4^\star$ generated by the procedure from Section 3 are provided in Appendix A.

Solving the constraints (4)–(6) and $\dot{c_0} = \texttt{do}(1)$, we obtain the following approximation for the set of feasible contexts: $\dot{p} = \{\texttt{do}(1), \texttt{do}(1) \wedge \texttt{ok}(\ell), \texttt{do}(2) \wedge \texttt{ok}(\ell), \texttt{do}(3) \wedge \texttt{ok}(\ell), \texttt{do}(4) \wedge \texttt{ok}(\ell)\}$. Since this $\dot{p}$ contains only $\texttt{do}(4) \wedge \texttt{ok}(\ell)$, the solution set of the linear constraints is *always* bounded sharply after termination of the composite solver.

We can find out when the composite solver bounds the solution set $\texttt{sol}(\ell \cup i \cup b)$ sharply, solving the constraints (4)–(6), "the Simplex method is called first" and "the solution set is bounded sharply after termination" ($\texttt{do}(1)$ "is implied by" $\dot{c_0}$, the context after termination $\dot{c}_\infty \in \dot{p}$, $\texttt{do}(4) \wedge \texttt{ok}(i) \wedge \texttt{ok}(\ell)$ "is implied by" $\dot{c}_\infty$). These constraints have 6 solutions. The first two assign $\texttt{do}(4) \wedge \texttt{ok}(\ell) \wedge \texttt{ok}(i)$ to $\dot{c}_\infty$ and either $\texttt{do}(1) \wedge \texttt{ok}(i)$, or $\texttt{do}(1) \wedge \texttt{ok}(i) \wedge \texttt{ok}(\ell)$ to $\dot{c_0}$. The other four assign $\texttt{do}(4) \wedge \texttt{ok}(\ell) \wedge \texttt{tree}(i) \wedge \texttt{ok}(i)$ to $\dot{c}_\infty$ and one of the 4 context properties that "imply" $\texttt{do}(1) \wedge \texttt{tree}(i)$ to $\dot{c_0}$. This means that the composite solver bounds the solution set $\texttt{sol}(\ell \cup i \cup b)$ sharply, if the interval constraints i have an acyclic constraint graph.

5. Conclusion

We have presented a formalism for automatic analysis of composite solvers. This formalism provides a structure for expressing properties of the data store (context properties), a structure for specifying the behaviour of solvers (abstract solvers), a method for approximation of composite solvers by set constraints that can be efficiently solved by conventional set constraint solvers like [4, 15]. The ultimate goal (out of the scope of this paper) is to couple our analysis with the language for specification of composite solvers in the framework of the COCONUT project [3].

References

[1] Benhamou F., Goualard F., Granvilliers L., Puget J.-F. Revising hull and box consistency // Proc. 16th Int. Conf. Logic Programming. — The MIT Press, 1999. — P. 230–244.

[2] Beringer H., De Backer B. Combinatorial problem solving in constraint logic programming with cooperating solvers // Logic Programming: formal methods and practical applications. — Elsevier Science Publishers, 1994.

[3] Bliek C. et al. Algorithms for solving nonlinear constrained and optimization problems: The state of the art // A progress report of the COCONUT project, 2001. —
`http://www.mat.univie.ac.at/~neum/glopt/coconut/StArt.html`

[4] Gervet C. Interval propagation to reason about sets: definition and implementation of a practical language // Constraints. — Kluwer Academic Publishers, 1997. — Vol. 3. — P. 191–244.

[5] Granvilliers L., Monfroy E., Benhamou F. Symbolic-interval cooperation in constraint programming // Proc. ISSAC 2001. — ACM, 2001.

[6] Hansen E. R. Sharpness in interval computations // Reliable Computing. — 1997. — Vol. 3, N 1. — P. 17–19.

[7] Hickey P. CLIP: a CLP(Intervals) dialect for metalevel constraint solving // Proc. Int. Workshop on Practical Aspects of Declarative Languages. — Lect. Notes in Comput. Sci. — 2000. — Vol. 1753. — P. 200–214.

[8] International Computers Limited & Imperial College of London. — ECL^{i}PS^e User Manual (release 5.3), 1996–2000.

[9] Jaffar J., Michaylov S., Stuckey P., Yap R. The CLP($\Re$) language and system // ACM Trans. on Programming Languages and Systems. — 1992. — Vol. 14, N 3. — P. 339–395.

[10] Marti P., Rueher M. Distributed cooperating constraint solving systems // Intern. J. of Artificial Intelligence Tools. — 1995. — Vol. 4, N 1&2. — P. 95–113.

[11] Monfroy E. The constraint solver collaboration language of BALI // Frontiers of Combining Systems 2 / Ed. by D. M. Gabbay and M. de Rijke. — Research Studies Press / Wiley, 2000.

[12] Mozart consortium. — The Mozart programming system, 1999.

[13] PROLOGIA. — Prolog IV: constraints inside (reference manual), 1996.

[14] Van Hentenryck P., Michel L., Deville Y. Numerica: a Modelling Language for Global Optimization. — Cambridge: The MIT Press, 1997.

[15] Yakhno T., Petrov E. Extensional set library for ECL^{i}PS^e // Lect. Notes in Comput. Sci. — 1999. — Vol. 1755. — P. 434–444.

A. Specification of the example from Section 4.2

The constraints from the sharpness example are provided in Figure 1 in the LogiCalc language [15]. We recall its syntax/semantics. The LogiCalc language allows the user to specify constraints on integer numbers, tuples and finite sets. Tuples of sets, sets of tuples, sets of sets, etc. are allowed. The constraints are specified in terms of set inclusion `subset`, membership `in`, equality =, and inequality <=. The left and right hand sides of the constraints are expressions built from variables, arithmetic and set operations, and specifications of finite set.

```
% Notation for the data properties: %
treeI  = 0; okL  = 1; okI      = 2; treeIokL    = 3;
okIokL = 4; true = 5; treeIokI = 6; treeIokLokI = 7;
Cdata = { treeI, okL, okI, treeIokL,
          okIokL, true, treeIokI, treeIokLokI };
% cplex %
F1star = {
        ((1, treeI),       (2, treeIokL)),
        ((1, okL),         (2, okL)),
        ((1, okI),         (2, okIokL)),
        ((1, treeIokL),    (2, treeIokL)),
        ((1, okIokL),      (2, okIokL)),
        ((1, true),        (2, okL)),
        ((1, treeIokI),    (2, treeIokLokI))
        ((1, treeIokLokI), (2, treeIokLokI)) } \/
{ ((i1, z1), (i1, z1)) | i1 in { 2, 3, 4 }, z1 in Cdata };
% hc %
F2star = {
        ((2, treeI),       (3, treeIokI)),
        ((2, okL),         (3, okL)),
        ((2, okI),         (3, okI)),
        ((2, treeIokL),    (3, treeIokLokI)),
        ((2, okIokL),      (3, okIokL)),
        ((2, true),        (3, true)),
        ((2, treeIokI),    (3, treeIokI))
        ((2, treeIokLokI), (3, treeIokLokI)) } \/
{ ((i2, z2), (i2, z2)) | i2 in { 1, 3, 4 }, z2 in Cdata };
% same? %
F3star = { ((3, z3), (i3, z3)) | z3 in Cdata, i3 in { 1, 4 } };
% done %
F4star = { ((4, z4), (4, z4)) | z4 in Cdata };
% constraints (4)--(5): %
img1 = { c11 | (c1, c11) in F1star; c1 in p };
img2 = { c22 | (c2, c22) in F2star; c2 in p };
img3 = { c33 | (c3, c33) in F3star; c3 in p };
img4 = { c44 | (c4, c44) in F4star; c4 in p };
p = { c0 } \/ img1 \/ img2 \/ img3 \/ img4;
```

Figure 1. Specification of the sharpness example in the LogiCalc language; {} denotes the empty set, \/ denotes set union, `F1star` specifies the abstract CPLEX, `F2star` specifies the abstract Hull consistency, `c0` denotes the initial context that we search for, `img1`, `img2` denote the images of the set of feasible contexts under the abstract CPLEX and Hull consistency

Finite sets are specified by either enumeration of the elements, or by their common property. For example,

```
x subset { 2, 3, 5, 7 };
y = { i * j | i in x; j in x; i + 1 <= j };
y = { 6, 10, 15 }
```

specify the set $y = \{i \cdot j | i \in x, j \in x, i < j\} = \{6, 10, 15\}$ and the set $x = \{2, 3, 5\}$ of their prime factors. Note implicit existential quantification of the variables i and j that do not occur outside the specification of y. Note that the equation $y = \{i \cdot j| \ldots\}$ is, in fact, a constraint on x and y.