A Preliminary Study on Formal Verification of Formula Transformations for Mathematical Programming

Masaki Nakamura, and Kazutoshi Sakakibara

Abstract—**To use a solver for optimization problems, we need to formulate a mathematical programming model acceptable by the solver. Such a formulation may need some heuristic techniques with professional knowledge and the user is responsible for the correctness of the formulation. In this paper we investigate a way to verify the correctness of formulation by using formal verification techniques. Through examples of formulation techniques in optimization problems, we give a way to describe formal specifications of given models and a way to verify the correctness of the models.**

Index Terms—**optimization problems, mathematical programming, formula transformations, formal methods, algebraic specifications.**

I. Introduction

Optimization problems are the problems to find the best solution from all feasible solutions which satisfy expressions of equalities or inequalities. There are lots of applications, for example, supply chain planning, transport planning, and so on [1], [2], [3]. In general, an optimization problem is modeled as follows:

$$\text{minimize } f(x) \text{ subject to } C$$

where $f(x)$ is an objective function and C is a set of constraints written in equalities or inequalities, e.g. $g_0(x) \leq a_0, g_1(x) = a_1, g_2(x) \geq a_2, \ldots$. The goal of this optimization problem is to find the value of x satisfying C with minimal value $f(x)$. There are lots of solvers to solve optimization problems. To use a solver, we need to formulate models which can be accepted by the solver. For example, when we use an integer linear programming (ILP) solver, we have to formulate a model with linear relationships. The following is an example of ILP models: $f(x_1, x_2) = -4x_1 - 5x_2$ and $C = \{\ 2x_1 + 2x_2 \leq 7,\ 3x_1 + 5x_2 \leq 14,\ x_1 \geq 0\ x_2 \geq 0\ \}$ where $x_1, x_2 \in \mathcal{Z}$ [4]. When ignoring domain of integers for x_1 and x_2, the problem can be regarded as a linear programming problem and the simplex method, for example, can easily achieve the optimal solution $(x_1, x_2) = (7/4, 7/4)$ with the minimal value $f(x_1, x_2) = -15.75$. To obtain the solution of the original integer linear programming problem, we need to round down x_1 and round up x_2 such that $(x_1, x_2) = (1, 2)$ with the minimal value $f(x_1, x_2) = -14$. In general, it is not so easy to find the solution of an integer linear programming problem from the solution of the corresponding linear programming problem. With the integer constraint, all points satisfying the constraints of C are $(0,0), (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (2,0)$ and $(2,1)$, and thus the above problem can be rewritten into the following formulation: $C' = \{\ x_1 + x_2 \leq 3,\ x_2 \leq 2,\ x_1 \geq 0\ x_2 \geq 0\ \}$.

Formula transformations may help us to find a solution of a given optimization problem to which it is hard or impossible to apply a solver. The goal of formula transformations is to find a formula such that (1) a solver can be applied to the formula and/or it is easier to obtain a solution than the original formula and (2) a solution obtained from the transformed problem is a solution of the original problem. To find such transformations is not easy and needs professional knowledges and insights. We focus on the latter condition (2), that is, the correctness of formula transformations. In the above example, the transformed formula C' have to be sufficient to C, i.e., $C' \Rightarrow C$ should hold. The user is responsible for correctness of the formula transformation.

Our motivation is to give a way to prove the correctness of formula transformations by using formal method techniques. Formal methods are mathematical techniques for the specification and verification of software and/or hardware systems. Formal specification languages play an important role in formal methods. Algebraic specification languages, e.g. OBJ language family: OBJ3[5], CafeOBJ[6], Maude[7], are formal specification languages whose models are algebra. Algebraic specifications are used for describing and verifying wide ranges of systems: from abstract data types to abstract machines. For example, we can describe a specification of natural numbers and operators on them (in Peano style) , and verify a property of such operators (e.g. the distribution law on multiplication and addition). A behavior of a system, like authentication protocols, bank account systems, e-government systems, can be specified and some desired properties of those systems can be verified formally [8], [9], [10], [11]. In this study, we use CafeOBJ algebraic specification language and give a way to specify constraints of optimization problems and to verify that formula transformations are correct or not.

II. Preliminaries

We introduce algebraic specification languages briefly (See [12], [6] for more details).

A CafeOBJ specification consists of modules. The following is an example of CafeOBJ modules:

```
mod! ABC{
  [Elt]
  ops a b c : -> Elt
```

Manuscript received Dec 8th, 2017; revised Jan 16th, 2018.
Masaki Nakamura and Kazutoshi Sakakibara are with the Department of Electrical and Computer Engineering, Toyama Prefectural University, Japan (e-mail: masaki-n@pu-toyama.ac.jp, sakakibara@pu-toyama.ac.jp).

```
  op f : Elt -> Elt
  eq a = b .
  eq f(X:Elt) = X .
}
```

The module name is ABC. A module consists of imports, a signature, and axioms. There no imports in ABC. A signature consists of sorts and operation symbols. Sorts are declared between [and]. Operation symbols are declared after op or ops. An operation symbol has its rank of sorts s_1 s_2 s_{n-1} -> s_n, where $s_1, \ldots s_{n-1}$ are called arity sorts and s_n is called co-arity sort. The operation symbols a, b and c have the rank -> Elt. The empty arity operation symbols are called constants. The operation symbol f has the rank Elt -> Elt. A term is a tree structure whose nodes are operation symbols and leaves are variables. An operation symbol d : -> s with the empty arity forms a term of the sort s by itself. A variable x of the sort s is denoted by x :s in term expressions, and it is also a term of s. From an operation symbol f with the rank s_1 s_2 s_{n-1} -> s_n and terms $t_1, \ldots, t_{n-1}$ of the sorts $s_1, \ldots s_{n-1}$, the expression $f(t_1, \ldots, t_n)$ is a term of the sort s_n. Axioms are equations or transitions. An equation is declared with eq l = r, where l and r are terms of the same sort. When equations have variables, the variables mean the arbitrary terms. For example, f(X:Elt) = X means that f(t) = t for any term t of Elt.

A CafeOBJ module with the signature Σ and axioms E denotes (a set of) (Σ, E)-algebras. A (Σ, E)-algebra M consists of carrier sets and operations on the sets. A sort s is interpreted into a carrier set M_s and an operation symbol $f : s_1 s_2 \ldots, s_{n-1} \to s_n$ is interpreted into an operation $M_f : M_{s_1} \times M_{s_2} \times \cdots \times M_{s_{n-1}} \to M_{s_n}$. A term with variables is also interpreted into an operation which takes elements corresponding to the variables. In a (Σ, E)-algebra M, the left-hand side and the right-hand side of an equation $l = r$ is interpreted into a same function or element $M_l = M_r$. The following algebra M_{01} is one of the denotation of the module ABC: $(M_{01})_{\mathtt{Elt}} = \{0, 1\}$ $(M_{01})_{\mathtt{a}} = 0$, $(M_{01})_{\mathtt{b}} = 0$, $(M_{01})_{\mathtt{c}} = 1$, $(M_{01})_{\mathtt{f}}(x) = x$ for each $x \in \{0, 1\}$.

CafeOBJ supports semi automated equational reasoning by term rewriting, which is implemented as the reduction command in CafeOBJ interpreter. In the theory of term rewriting, a term is reduced by applying equations as left-to-right rewrite rules. The followings are examples of the CafeOBJ executions.

```
OP3> select ABC
ABC> red f(a) = f(b) .
(true):Bool
ABC> red c = f(f(c)) .
(true):Bool
ABC> red f(c) = b .
(c = b):Bool
```

The three equations $f(a) = f(b)$, $c = f(f(c))$ and $f(c) = b$ are tried to be proved. For the first and second ones, CafeOBJ interpreter returns true, which means that those equations are deducible from the equations of axioms in the module ABC, and thus they are true in every denotational model of ABC (e.g. M_{01}). The last equation is not proved and the reduction command returns the equation $c = b$. A returned value except true does not imply the disproof of the input equation. When true is not returned, the input equation may or may not hold. The CafeOBJ reduction command shows a hint for the proof. In this case, $c = b$ is returned and it means that $f(c) = b$ holds if $c = b$ holds.

The following is another example of CafeOBJ modules:

```
mod! AMGM{
  pr(INT)
  op p : Int Int -> Bool .
  eq p(X:Int, Y:Int) =
    (X + Y) * (X + Y) > 4 * X * Y .
}
```

The module AMGM specifies the inequality of arithmetic and geometric means. The module imports the built-in module INT in the protect mode. A denotational model M of a module imported in the protect mode is a sub model of a model M' of the importing module, that is, the carrier set of each sort in the imported module is same in both M and M' ($M_s = M'_s$ for each sort). In the built-in module INT, the sort Int of the set of integer numbers, the constants $\ldots - 2, -1, 0, 1, 2, \ldots$ of the integer numbers, and several operations $+, -, *, >, \ldots$ on integers are declared. A model of AMGM has the carrier set $M_{\mathtt{Int}} = \mathcal{Z}$. Importing modules can use members of imported modules. The built-in Boolean module BOOL with the sort Bool is also imported implicitly in CafeOBJ. AMGM specifies the predicate p(x, y) on a pair of the integer numbers which checks the inequality of arithmetic and geometric means ($\frac{x+y}{2} > \sqrt{xy}$). For example, the term p(3, 4) is reduced into true as follows:

```
AMGM> red p(3,4) .
---> (((3 + 4) * (3 + 4))
                    > (4 * (3 * 4))):Bool
---> ((7 * (3 + 4)) > (4 * (3 * 4))):Bool
---> ((7 * 7) > (4 * (3 * 4))):Bool
---> (49 > (4 * (3 * 4))):Bool
---> (49 > (4 * 12)):Bool
---> (49 > 48):Bool
---> (true):Bool
```

Next, we introduce a way to describe a transition system in CafeOBJ. The following is an example of CafeOBJ modules of transition systems.

```
mod! CHECK{
  pr(INT)
  [Pair]
  op pair : Int Int -> Pair
  op n : -> Int
  trans n => n + n .
  trans n => 1 .
  trans n => 3 .
}
```

The module CHECK imports INT and declares the sort Pair for the pair of integer numbers, the operation symbol pair of a constructor of the pairs, and the constant n of Int. A transition rule is declared like trans l => r. Roughly speaking, a CafeOBJ module with transition rules denotes a transition system $(T, \to)$ where the set T of states is the set of terms modulo equations and the state transition

$\to$ on T is obtained from the transition rules. The relation $t_1 \to t_2$ holds if and only if t_2 is obtained by applying the transition rules repeatedly from t_1. In the above example, $n \to 1$ and $n \to 3$ hold from the latter transitions. $n \to 2$ and $n \to 4$ also hold since $n \to n+n \to 1+1 = 2$ and $n \to n+n \to 1+3 = 4$.

The exhaustive search command is supported in CafeOBJ for specifications with transition rules, denoted by `red` t_0 `=(`m,n`)=>*` t_1 `suchThat` p. The command searches m states (terms) which are reachable from the initial state t_0 by the transition rules within the depth n, match the pattern t_1 and satisfy the predicate p.

The following is an example of the exhaustive search command:

```
select CHECK .
red pair(n, n) =(3,10)=>*
    pair(X1:Int,X2:Int)
    suchThat (X1 + X2 = 5) .
```

CafeOBJ interpreter computes all states matching the pattern `pair(X1, X2)` reachable from the initial state `pair(n, n)` within the depth ten, and three states satisfying `X1 + X2 = 5` is returned as follows:

```
** Found [state 72] (pair(2,3)):Pair
** Found [state 74] (pair(4,1)):Pair
** Found [state 95] (pair(1,4)):Pair
-- found required number of solutions 3.
```

III. Proving the correctness of formula transformations by algebraic specifications

In this section, we investigate how to prove the correctness of formula transformations in optimization problems by using (1) exhaustive searching and (2) interactive theorem proving in CafeOBJ.

A. Exhaustive search

Consider the following example of optimization problems: there are ten projects P_i and their costs a_i. At least five projects should be done. For the projects P_1, P_2 and P_3, either exactly two of them or no projects are chosen. Choose projects to minimize the total costs.

The problem is straightforwardly formulated as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{10} a_i x_i \\ \text{subject to} & \sum_{i=1}^{10} x_i \geq 5 \\ & \left\{ \begin{array}{l} x_1 + x_2 + x_3 = 2 \\ \vee \\ x_1 + x_2 + x_3 = 0 \end{array} \right. \end{array} \tag{1}$$

where each $x_i \in \{0, 1\}$ represents a decision variable, that is, P_i is chosen when $x_i = 1$, otherwise P_i is not.

The last disjunctive constraint, $x_1 + x_2 + x_3 = 2 \ \vee \ x_1 + x_2 + x_3 = 0$, is not acceptable for most solvers. For example, the above formulation is out of the scope of the integer linear programming, which allows only a conjunction of equalities and inequalities. The disjunction can be transformed to a conjunction of four inequalities as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{10} a_i x_i \\ \text{subject to} & \sum_{i=1}^{10} x_i \geq 5 \\ & x_1 + x_2 + x_3 \leq 2 \\ & x_1 - x_2 - x_3 \leq 0 \\ & x_2 - x_3 - x_1 \leq 0 \\ & x_3 - x_1 - x_2 \leq 0 \end{array} \tag{2}$$

Those two formulations are described in CafeOBJ specifications as follows:

```
mod! OP{
  pr(INT)
  [Int < EInt]
  [Triple]
  op n : -> EInt .
  op triple : EInt EInt EInt -> Triple
  op req : EInt EInt EInt -> Bool .
  op cond : EInt EInt EInt -> Bool .

  vars X1 X2 X3 : Int

  eq (X1 = X2) = X1 <= X2 and X1 >= X2 .

  eq req(X1, X2, X3) =
    X1 + X2 + X3 = 2 or
    X1 + X2 + X3 = 0 .

  eq cond(X1, X2, X3) =
    X1 +   X2 +   X3 <= 2 and
    X1 + - X2 + - X3 <= 0 and
    X2 + - X3 + - X1 <= 0 and
    X3 + - X1 + - X2 <= 0 .

  trans n => 0 .
  trans n => 1 .
}
```

The module `OP` imports the built-in module `INT`. The sort `EInt` is introduced to avoid unnecessary computation in the exhaustive searching. The sort `EInt` is declared as a super sort of `Int`. A super sort is interpreted into a super set of the carrier set of its sub sorts in a denotational model. The constant `n` of the sort `EInt` is declared. In the last part of `OP`, two transition rules are declared. The constant `n` has two successor states `0` or `1` of `Int`. The operation symbols `req` and `cond` correspond to the formulas $x_1 + x_2 + x_3 = 2 \vee x_1 + x_2 + x_3 = 0$ (`req`) and $x_1 + x_2 + x_3 \leq 2 \wedge x_1 - x_2 - x_3 \leq 0 \wedge x_2 - x_3 - x_1 \leq 0 x_3 - x_1 - x_2 \leq 0$ (`cond`). Note that the variables `X1`, `X2` and `X3` are declared as variables of `Int`. The term `req(1,0,1)` is equivalent to the term `1 + 0 + 1 = 2 or 1 + 0 + 1 = 0`, however, the term `req(n,n,n)` is not equivalent to the term `n + n + n = 2 or n + n + n = 0` since the constant `n` is not of the sort `Int`. Because of this mechanism, we can avoid unnecessary check for `n + n + n = 2 or n + n + n = 0`, which cannot be evaluated into `true` or `false`.

The correctness of the transformation can be proved by exhaustive searching in CafeOBJ. To prove the predicate P

on states, we prove that there is no reachable state s satisfying the negation of $P(s)$. The predicate to be proved is the equivalence of req(n_1, n_2, n_3) and cond(n_1, n_2, n_3) for each $n_1, n_2, n_3 \in \{0, 1\}$. Thus, we can prove the correctness of the formula transformation by the following command:

```
red triple(n,n,n) =(*,*)=>*
    triple(X1,X2,X3)
    suchThat
(not (cond(X1,X2,X3) = req(X1,X2,X3))) .
```

The above command returns false, that is, there is no combination of $n_1, n_2, n_3 \in \{0, 1\}$ satisfying req(n_1, n_2, n_3) $\neq$ cond(n_1, n_2, n_3).

The exhaustive searching can also be used to find wrong formula transformations. Consider the following formulation:

```
  eq cond(X1, X2, X3) =
    X1 +   X2 +   X3 <= 2 and
    X1 + - X2 + - X3 <= 0 and
    X2 + - X3 + - X1 <= 0 and
    X3 + - X1 + - X2 <= 1 .
```

where the right-hand side of the last inequality is one wrongly. Then, the above command returns the following state as a counter-example of req(n_1, n_2, n_3) = cond(n_1, n_2, n_3):

```
** Found [state 20] (triple(0,0,1)):Triple
```

Certainly, the results are different for $(0, 0, 1)$, that is, $cond(0, 0, 1) = true \neq false = req(0, 0, 1)$.

B. Interactive theorem proving

The exhaustive searching is useful since the result is returned automatically and not only a proof but also a disproof can be done. However, because of the exhaustiveness, the computation cost (time and space) increases with increasing the size of the reachable states of a given transition system. The search space is not so wide for the example in the previous section since the domain of variables is $\{0, 1\}$. If the domain is the set $\mathcal{Z}$ of all integers, this approach cannot be taken straightforwardly.

In CafeOBJ, interactive theorem proving can be done by using equational reasoning with the CafeOBJ reduction command. Roughly speaking, the user makes a proof plan of a given property and checks whether components (equations) of the proof are correct or not by using CafeOBJ interpreter. Consider the following integer linear programming problem (shown in Section I):

$$\begin{array}{rl} \text{minimize} & -4x_1 - 5x_2 \\ \text{subject to} & 2x_1 + 2x_2 \leq 7 \\ & 3x_1 + 5x_2 \leq 14 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{array} \tag{3}$$

where $x_1, x_2 \in \mathcal{Z}$. As we already discussed in Section I, the above formulation can be transformed as follows:

$$\begin{array}{rl} \text{minimize} & -4x_1 - 5x_2 \\ \text{subject to} & x_1 + x_2 \leq 3 \\ & x_2 \leq 2 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{array} \tag{4}$$

To prove correctness of the formula transformation, we need prove that the constraints of (3) are equivalent to the constraints of (4). In the following, we show a proof of one implication: the set of the constraints of (4) implies each of the constraints of (3).

We describe a CafeOBJ module of integer variables x_1 and x_2 satisfying the constraints of (4).

```
mod* OP4{
  pr(INT)
  ops x1 x2 : -> Int
  eq x1 + x2 <= 3 = true .
  eq x2 <= 2 = true .
  eq x1 >= 0 = true .
  eq x2 >= 0 = true .
}
```

The module OP4 imports the built-in module INT and declares two constants x1 and x2 of Int. There are four equations which correspond to the constraints of (4). The module denotes all algebras in which the constants x1 and x2 are interpreted into integers satisfying the equations in OP4. For example, M is a denotational model of OP4 if $M_{\mathtt{x1}} = 2$ and $M_{\mathtt{x2}} = 1$. For a given inequality t, if the reduction command reduces t into true, the inequality t holds in OP4. Thus, if all inequalities in the constraints of (3) are reduced into true, the proof of the target implication is done.

Unfortunately, it does not succeed for $2x_1 + 2x_2 \leq 7$ as follows:

```
OP4> red 2 * x1 + 2 * x2 <= 7 .
(((2 * x2) + (2 * x1)) <= 7):Bool
```

The first line is the input and the second line is the output. The input term is returned as it is. To use the equations in OP4, the target term should be rewritten to a term which matches the left-hand side of some equation. We know the distribution law holds for integers:$x \times y + x \times z = x \times (y + z)$. When we add the equation eq X * Y + X * Z = X * (Y + Z) as a lemma to OP4, the output of the reduction is changed as follows:

```
OP4> red 2 * x1 + 2 * x2 <= 7 .
((2 * (x1 + x2)) <= 7):Bool
```

The output is still not true. Because of the integer domain, the output equation can be transformed into $x_1 + x_2 \leq 3$ by dividing the both-hand sides by two. In general, $n \times x \leq y$ is true whenever $x \leq \lfloor y/n \rfloor$, where $\lfloor y/n \rfloor$ is the quotient. This lemma is written in CafeOBJ equations as follows: cq NZ * X <= Y = true if X <= Y quo NZ, where NZ is a variable of non-zero integers. When this equation is also added to OP4, the proof succeeds as follows:

```
OP4> red 2 * x1 + 2 * x2 <= 7 .
(true):Bool
```

The inequality $3x_1 + 5x_2 \leq 14$ in the constraints of (3) can also be proved by adding other two lemmas to `OP4`. The remaining inequalities $x_1 \geq 0$ and $x_2 \geq 0$ are proved without any lemma. Then, the constraints of (3) can be deduced from those of (4) for all integers.

As above, in interactive theorem proving, a proof is made by interaction with CafeOBJ interpreter (the reduction command). We first try to prove a target property by the reduction command, consider suitable lemmas with observation of the result of the reduction, add the lemmas to the module and reduce the target property again. This cycle is repeated until `true` is returned.

IV. Conclusion

We showed how to do formal verification of formula transformation for optimization problems by algebraic specifications. Two approaches have been shown: the exhaustive searching and the interactive theorem proving. The exhaustive searching is fully automated and can be used for both proofs and disproofs. The interactive theorem proving can deal with variables with the domain of infinite elements, like $\mathcal{Z}$. Not only introduction of lemmas but also case splitting and structural inductions can be used in interactive theorem proving, called the proof score method [13]. For example, strictly speaking, before introducing the distribution law $x \times y + x \times z = x \times (y + z)$ for the specification of integers, the equation should be proved. We can describe the specification of integers in the extension of the Peano numbers, and verify the distribution law by the proof score method in CafeOBJ.

From the experiences in this paper, it is confirmed that formal verification techniques with algebraic specifications may be useful to prove the correctness of formula transformations formally. The examples we have shown are very small and simple. One of our future directions is to apply the proposed approaches to more large and complex practical problems, and other formulation techniques, like the Big-M method [14] and so on.

References

[1] A. Conejo, *Decomposition Techniques in Mathematical Programming: Engineering and Science Applications*. Springer, 2006.

[2] Y. Pochet and L. A. Wolsey, *Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[3] J. Mula, D. Peidro, M. Díaz-Madroñero, and E. Vicens, “Mathematical programming models for supply chain production and transport planning,” *European Journal of Operational Research*, vol. 204, no. 3, pp. 377–390, 2010.

[4] T. Fujie, “Introduction to integer linear programming formulations (in Japanese),” *Operations research as a management science research*, vol. 57, no. 4, pp. 190–197, apr 2012.

[5] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud, *Software Engineering with OBJ: Algebraic Specification in Action*. Kluwers Academic Publishers, 2000, ch. Introducing OBJ*.

[6] R. Diaconescu and K. Futatsugi, *CafeOBJ Report*. Singapore: World Scientific, 1998.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, ser. Lecture Notes in Computer Science, vol. 4350. Springer, 2007.

[8] I. Ouranos, K. Ogata, and P. S. Stefaneas, “Formal analysis of tesla protocol in the timed OTS/CafeOBJ method,” in *ISoLA (2)*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 7610. Springer, 2012, pp. 126–142.

[9] W. Kong, K. Ogata, and K. Futatsugi, “Towards reliable e-government systems with the OTS/CafeOBJ method,” *IEICE Transactions*, vol. 93-D, no. 5, pp. 974–984, 2010.

[10] K. Ogata and K. Futatsugi, “Flaw and modification of the iKP electronic payment protocols,” *Inf. Process. Lett.*, vol. 86, no. 2, pp. 57–62, 2003.

[11] ——, “Formal analysis of Suzuki & Kasami distributed mutual exclusion algorithm,” in *FMOODS*, ser. IFIP Conference Proceedings, B. Jacobs and A. Rensink, Eds., vol. 209. Kluwer, 2002, pp. 181–195.

[12] *CafeOBJ*, http://www.ldl.jaist.ac.jp/cafeobj/.

[13] K. Futatsugi, D. Găină, and K. Ogata, “Principles of proof scores in CafeOBJ,” *Theor. Comput. Sci.*, vol. 464, pp. 90–112, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2012.07.041

[14] P. Barth, *Logic-based 0-1 constraint programming*, ser. Operations Research/Computer Science Interfaces. Boston, USA: Kluwer, November 1995.