Constraints manuscript No.
(will be inserted by the editor)

On the Reification of Global Constraints

Nicolas Beldiceanu · Mats Carlsson · Pierre Flener · Justin Pearson

Received: February 17, 2012 / Revised: October 16, 2012/ Accepted: ?

Abstract We introduce a simple idea for deriving reified global constraints in a systematic way. It is based on the observation that most global constraints can be reformulated as a conjunction of total function constraints together with a constraint that can be easily reified.

Keywords Global constraint · reification · reformulation · functional dependency

1 Introduction

Conventional wisdom has it that many global constraints cannot be easily reified, i.e., augmented with a 0-1 variable reflecting whether the constraint is satisfied (value 1) or not (value 0). Reified constraints are useful for expressing propositional formulas over constraints and for expressing that a certain number of constraints hold (e.g., the cardinality operator [14]). Using standard algorithms from automata theory, we have previously shown [4, Section 3.7.196 of 2010 version] how to reify a global constraint that can be expressed in terms of a counter-free finite automaton [3,12]. However, many global constraints, such as ALLDIFFERENT and CUMULATIVE, cannot be expressed by an automaton whose size is polynomial in the number of variables of the constraint. The importance of the negation of global constraints has recently increased, e.g.,

N. Beldiceanu
TASC team (INRIA/CNRS), Mines de Nantes, FR – 44307 Nantes, France
E-mail: Nicolas.Beldiceanu@mines-nantes.fr

M. Carlsson
SICS, P.O. Box 1263, SE – 164 29 Kista, Sweden, E-mail: Mats.Carlsson@sics.se

P. Flener · J. Pearson
Uppsala University, Box 337, SE – 751 05 Sweden, E-mail: Firstname.Surname@it.uu.se

in the context of a constraint seeker with negative samples [5] and for proving the equivalence of constraint models [1,10].

Many early constraint programming systems, such as CHIP, GNU Prolog, Ilog Solver, and SICStus Prolog, provide reification for arithmetic constraints. However, when global constraints started to get introduced (e.g., ALLDIFFERENT and CUMULATIVE), reification was not available for global constraints. We believe that, in the early 1990s, reification was not considered for global constraints since it was believed that reification could only be obtained by modifying the filtering algorithms attached to each global constraint. Nowadays, Minion [8,9] features *constraint trees*, which constitute a very efficient mechanism for executing Boolean combinations of primitive as well as global constraints.

In this letter, we present a *portable* reification method that is useful on solvers that do not have such features, and so this work is orthogonal to specific implementation approaches.

2 How to Derive Reified Global Constraints

A global constraint $GC(\mathcal{A})$ can be defined by restrictions $R(\mathcal{A})$ on its arguments $\mathcal{A}$, e.g., restrictions on the bounds of its arguments, and by a condition $C(\mathcal{A})$ on its arguments, i.e., we have $GC(\mathcal{A}) \equiv R(\mathcal{A}) \wedge C(\mathcal{A})$. For instance, for a constraint defined by a finite automaton (e.g., GLOBAL_CONTIGUITY [4, page 1058]), a typical restriction is that the variables take values in a given alphabet (e.g., values 0 and 1 for GLOBAL_CONTIGUITY). See [4, pages 9–17] for other examples of such restrictions. Note that the set of restrictions may be empty, that is $R(\mathcal{A})$ may be always satisfied. We define the *reified version* of $GC(\mathcal{A})$ as $R(\mathcal{A}) \wedge (C(\mathcal{A}) \Leftrightarrow b)$, where b is a 0-1 variable reflecting whether $GC(\mathcal{A})$ holds or not. In particular, we require the negation of $GC(\mathcal{A})$ to satisfy the same restrictions $R(\mathcal{A})$.

Let a *core reifiable constraint* be a constraint of the form of a Boolean combination of linear arithmetic equalities and inequalities and 0-1 variables. We assume that such constraints are already reifiable, without resorting to the methods being developed in this letter. This is the case in all constraint programming systems that we are aware of.

A constraint $R(v_1, \ldots, v_n)$ is a *total function* (TF) if and only if its variables can be partitioned into two non-empty sets, X and Y, such that for any assignment to the variables of X there is a *unique* assignment to the variables of Y satisfying R. For instance, NVALUE$(nv, \langle v_1, \ldots, v_n \rangle)$ [4, page 1466] is a TF since variable nv is uniquely determined by *the* number of distinct values of the set $\{v_1, \ldots, v_n\}$ of variables. However, ALLDIFFERENT$(\langle v_1, \ldots, v_n \rangle)$ is not a TF, because no subset of the variables $\langle v_1, \ldots, v_n \rangle$ uniquely determines the other variables. The set Y may contain more than one variable, witness the SORT$(\langle v_1, \ldots, v_n \rangle, \langle w_1, \ldots, w_n \rangle)$ constraint [4, page 1772], where $\langle v_1, \ldots, v_n \rangle$ uniquely determine $\langle w_1, \ldots, w_n \rangle$. The global constraint catalogue [4] contains a significant number (23%) of TF constraints.

We now provide the key observation that allows us to reify most global constraints in a straightforward way. Given a global constraint $GC(\mathcal{A})$ defined by $R(\mathcal{A}) \wedge C(\mathcal{A})$, it turns out that the condition $C(\mathcal{A})$ can often be reformulated as a conjunction $CF_1(\mathcal{A}_1, \mathcal{V}_1) \wedge \cdots \wedge CF_p(\mathcal{A}_p, \mathcal{V}_p) \wedge CN(\mathcal{A}_{p+1})$ of constraints, where $R(\mathcal{A})$ implies that each constraint $CF_i(\mathcal{A}_i, \mathcal{V}_i)$ (with $1 \leq i \leq p$) is a TF for which the determined variables $\mathcal{V}_i$ do not occur in $\mathcal{A}$, and where $CN(\mathcal{A}_{p+1})$ is a core reifiable constraint (i.e., we may use $CN(\mathcal{A}_{p+1}) \Leftrightarrow b$). The arguments of the constraints $CF_i(\mathcal{A}_i, \mathcal{V}_i)$ (with $1 \leq i \leq p$) and $CN(\mathcal{A}_{p+1})$ must obey the following conditions:

- $\mathcal{V}_i$ (with $1 \leq i \leq p$) is a non-empty set of distinct new variables, i.e., it has an empty intersection with $\mathcal{A} \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_{i-1} \cup \mathcal{V}_{i+1} \cup \cdots \cup \mathcal{V}_p$.
- $\mathcal{A}_i \subseteq \mathcal{A} \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_{i-1}$ (with $1 \leq i \leq p+1$), i.e., $\mathcal{A}_i$ gets fixed when $\mathcal{A}, \mathcal{V}_1, \ldots, \mathcal{V}_{i-1}$ are fixed.
- $\mathcal{V}_i$ has a non-empty intersection with $\mathcal{A}_{i+1} \cup \cdots \cup \mathcal{A}_{p+1}$, i.e., each introduced variable is used at least once.

If all the variables of $\mathcal{A}$ that occur in one of the $\mathcal{A}_i$ (with $1 \leq i \leq p$) are fixed, then all variables in $\mathcal{V}_i$ (with $1 \leq i \leq p$) are also fixed, by the TF constraints. Note that, from the first two conditions, the conjunction $CF_1(\mathcal{A}_1, \mathcal{V}_1) \wedge \cdots \wedge CF_p(\mathcal{A}_p, \mathcal{V}_p)$ determines the variables of $\mathcal{V}_1, \ldots, \mathcal{V}_p$ from the arguments $\mathcal{A}$.

In this context, the reified version of $GC(\mathcal{A})$ is expressed as follows:

$$R(\mathcal{A}) \wedge CF_1(\mathcal{A}_1, \mathcal{V}_1) \wedge \cdots \wedge CF_p(\mathcal{A}_p, \mathcal{V}_p) \wedge (CN(\mathcal{A}_{p+1}) \Leftrightarrow b)$$

3 Sample Reifications of Global Constraints

We now illustrate our approach on some constraints of the Global Constraint Catalogue, showing how to reify them by using a conjunction of TF constraints and a constraint for which reification is directly available.

ALLDIFFERENT$(\langle v_1, \ldots, v_n \rangle)$ [4, page 434] is reified as follows:

$$\text{SORT}(\langle v_1, \ldots, v_n \rangle, \langle w_1, \ldots, w_n \rangle) \wedge (w_1 < w_2 \wedge \cdots \wedge w_{n-1} < w_n) \Leftrightarrow b$$

GLOBAL_CARDINALITY$(\langle x_1, \ldots, x_n \rangle, \langle v_1\ o_1, \ldots, v_m\ o_m \rangle)$ [4, page 1034], where v_j and o_j (with $j \in [1, m]$) respectively denote the value for which we count the number of occurrences and the corresponding number of occurrences among the x_i variables (with $i \in [1, n]$), is reified as follows:

$$\text{GLOBAL\_CARDINALITY}(\langle x_1, \ldots, x_n \rangle, \langle v_1\ p_1, \ldots, v_m\ p_m \rangle) \wedge (o_1 = p_1 \wedge \cdots \wedge o_m = p_m) \Leftrightarrow b$$

Being a TF, GLOBAL_CARDINALITY is itself used in the TF part of its reformulation ($p = 1$), but with *other* determined variables; the core reified constraint of the reformulation compares the two sets of determined variables.

ELEMENT$(i, \langle t_1, \ldots, t_n \rangle, v)$ [4, page 958] is reified as follows:

$$\text{ELEMENT}(i, \langle t_1, \ldots, t_n \rangle, w) \wedge (v = w) \Leftrightarrow b$$

Since its restriction, $1 \leq i \leq n$, implies that ELEMENT is a TF(the variable v is uniquely determined by the index i and the table $\langle t_1, \ldots, t_n \rangle$), ELEMENT is itself used in the TF part of its reformulation, but with another determined variable w; it suffices to reify $v = w$.

CUMULATIVE$(\langle s_1\ d_1\ e_1\ r_1, \ldots, s_n\ d_n\ e_n\ r_n \rangle, \mathit{limit})$ [4, page 786], where s_i, d_i, e_i, and r_i (with $i \in [1, n]$) respectively denote the *start*, *duration*, *end*, and *resource consumption* of task i, can be reified by a reformulation that uses TF constraints for determining the maximum resource consumption:

- For each pair of tasks i, j (with $i, j \in [1, n]$) we create a variable r_{ij}, which is the resource consumption of task j if task j overlaps the start of task i, and 0 otherwise:
  - For $j = i$: $(d_i = 0 \wedge r_{ij} = 0) \vee (d_i > 0 \wedge r_{ij} = r_i)$
  - For $j \neq i$: $((s_j \leq s_i \wedge e_j > s_i \wedge s_i < e_i) \wedge r_{ij} = r_j) \vee ((s_j > s_i \vee e_j \leq s_i \vee s_i = e_i) \wedge r_{ij} = 0)$
- For each task i (with $i \in [1, n]$) we create a variable sr_i, which is the sum of the resource consumptions of the tasks that overlap the start of task i (task i overlaps its own start), i.e., $sr_i = r_{i1} + \cdots + r_{in}$.

Finally, $(s_1 + d_1 = e_1 \wedge \cdots \wedge s_n + d_n = e_n \wedge sr_1 \leq \mathit{limit} \wedge \cdots \wedge sr_n \leq \mathit{limit}) \Leftrightarrow b$ is the reified constraint.

DIFFN$(\langle \langle o_{11}\ s_{11}\ e_{11}, \ldots, o_{1m}\ s_{1m}\ e_{1m} \rangle, \ldots, \langle o_{n1}\ s_{n1}\ e_{n1}, \ldots, o_{nm}\ s_{nm}\ e_{nm} \rangle \rangle)$ [4, page 872], where o_{ik}, s_{ik}, and e_{ik} (with $i \in [1, n]$ and $k \in [1, m]$) respectively denote the *origin*, *size*, and *end* in dimension k of object i, is reified as follows:

$$\left(\bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n \\ i < j}} \bigvee_{1 \leq k \leq m} (s_{ik} = 0 \vee s_{jk} = 0 \vee o_{ik} \geq e_{jk} \vee o_{jk} \geq e_{ik}) \wedge \bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq k \leq m}} (o_{ik} + s_{ik} = e_{ik}) \right) \Leftrightarrow b$$

The constraint holds if each pair of these objects has no overlap. Unlike all the previous examples, we do not need any TF constraints here, i.e., $p = 0$.

SYMMETRIC_ALLDIFFERENT$(\langle s_1, \ldots, s_n \rangle)$ [4, page 1854] holds if $S = \langle s_1, \ldots, s_n \rangle$ is a permutation of $[1, n]$ with $\frac{n}{2}$ permutation cycles of length 2. It is reified as follows. For each i, we get its two closest successors and check that they form a cycle of length 2. The first step is done by stating the following TF constraints: ELEMENT$(s_1, S, t_1) \wedge \cdots \wedge$ ELEMENT(s_n, S, t_n). Finally, $(s_1 \neq t_1 \wedge t_1 = 1 \wedge \cdots \wedge s_n \neq t_n \wedge t_n = n) \Leftrightarrow b$ is the reified constraint.

Automata. Any constraint that can be modelled by an automaton with counters $c_1, \ldots, c_i$ with expected values $v_1, \ldots, v_i$ can be reified using the AUTOMATON meta-constraint [3] as follows:

$$\text{AUTOMATON}(\ldots) \wedge (w_1 = v_1 \wedge \cdots \wedge w_i = v_i \wedge w_{i+1} = 1) \Leftrightarrow b$$

where:

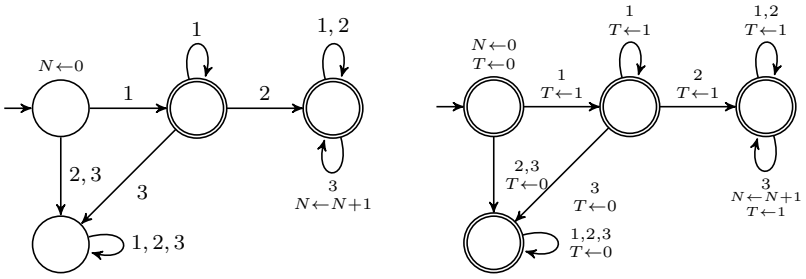

Fig. 1 Left: Automaton for a constraint over $\{1, 2, 3\}$ requiring that the first 2 be preceded by at least one 1, that the first 3 be preceded by at least one 2, and that there be at least one occurrence of 1; the counter N counts the number of occurrences of 3. Right: Its version used for reification, with an auxiliary counter T, reflecting the truth value.

- c_{i+1} is an auxiliary counter, with initial value 1 if the start state is an accept state, and 0 otherwise,
- $w_1, \ldots, w_{i+1}$ are the counter values in the state where the automaton stops,
- any arc leading to an accept state is amended with $c_{i+1} \leftarrow 1$,
- any arc leading to a non-accept state is amended with $c_{i+1} \leftarrow 0$,
- finally, all states are turned into accept states.

Figure 1 shows an example of this transformation. There are 90 such constraints in the catalogue. In [2], we give specialised reification methods for two cases where $i = 0$ (covering 19 and 41 constraints, respectively).

4 Conclusion

Based on the insight that most constraints can naturally be defined by a *determine and test* scheme, where the *determine* part is associated to total function (TF) constraints that determine additional variables, and the *test* part to a core reifiable constraint on these variables, we have shown that most global constraints can be reified. Surprisingly, this simple idea allows us to reify at least 313 of the 381 (i.e., 82%) constraints (details in [2]) of the Global Constraint Catalogue. Most of the constraints not covered are graph constraints involving set variables.

Some of our insights might be folklore. For instance, Tip 5.3 of [13, page 78] outlines the idea for TF constraints and gives an example, but the notion of TF and our more general pattern of Section 2 are not identified. Similarly, Example 14 of [10, page 58] also provides an example of negation for a TF constraint without identifying the pattern. The reformulations of constraints such as ALLDIFFERENT or GLOBAL_CARDINALITY in [6] can be unfolded to make explicit the TF and core reified constraints. As observed in [7, Section 4], given a global constraint c and its propagator, it is straightforward to construct a propagator for its half-reified version $b \Rightarrow c$, but not so for the only-if version $c \Rightarrow b$. In the context of software verification, the equivalence of constraint

models must sometimes be proven and one needs to negate global constraints [1,10].

The Zinc language [11] introduces local existentially quantified variables, which can be used for the same purpose as our functionally determined variables. In a negated or reified context, Zinc requires that such variables be functions of non-local variables and parameters, and restricts the set of functions that are allowed, whereas our approach admits *any* TF constraint. For example, Zinc allows array expressions as syntactic sugar for ELEMENT, but does not allow SORT as a function, even though both are TF. Thus there seems to be a case for lifting this restriction in Zinc.

Our decomposition scheme is mainly useful in case the solver does not directly support some global constraint in negated or reified contexts, as well as for obtaining reformulations of global constraints. While such reformulations may not be very efficient from a memory point of view for a reformulation whose size is quadratic in the number of variables of the constraint, many reformulations are quite compact.

Acknowledgements Many thanks to Christian Schulte and the anonymous referees for useful comments. The first author was supported by the French ANR project Net-WMS-2. The last two authors were supported by grant 2011-6133 of the Swedish Research Council.

References

1. Alvarez Divo, C.E.: Automated reasoning on feature models via constraint programming. Master's thesis, Uppsala University, Sweden (2011). Tech. Rep. IT 11 041
2. Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On the reification of global constraints. Tech. Rep. T2012:02, Swedish Institute of Computer Science (2012). Available at `http://soda.swedish-ict.se/view/sicsreport/`
3. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: CP 2004, *LNCS*, vol. 3258, pp. 107–122. Springer (2004)
4. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog, 2nd edition (revision a). Tech. Rep. T2012:03, Swedish Institute of Computer Science (2012)
5. Beldiceanu, N., Simonis, H.: A constraint seeker: finding and ranking global constraints from examples. In: J.H.M. Lee (ed.) CP 2011, *LNCS*, vol. 6876. Springer (2011)
6. Bessière, C., Katsirelos, G., Narodytska, N., Quimper, C.G., Walsh, T.: Decompositions of all different, global cardinality and related constraints. In: IJCAI, pp. 419–424 (2009)
7. Feydy, T., Somogyi, Z., Stuckey, P.J.: Half reification and flattening. In: J.H.M. Lee (ed.) CP 2011, *LNCS*, vol. 6876, pp. 286–301. Springer (2011)
8. Gent, I.P., Jefferson, C., Miguel, I.: Minion: a fast scalable constraint solver. In: ECAI 2006, pp. 98–102. IOS Press (2006)
9. Jefferson, C., Moore, N.C.A., Nightingale, P., Petrie, K.E.: Implementing logical connectives in constraint programming. Artificial Intelligence **174**, 1407–1429 (2010)
10. Lazaar, N.: Méthodologie et outil de test, de localisation de fautes et de correction automatique des programmes à contraintes. Ph.D. thesis, Rennes 1 Univ., France (2011)
11. Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P.J., de la Banda, M.G., Wallace, M.: The design of the Zinc modelling language. Constraints **13**(3), 229–267 (2008)
12. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: M.G. Wallace (ed.) CP 2004, *LNCS 2004*, vol. 3258, pp. 482–495. Springer (2004)
13. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and Programming with Gecode (version 3.7.1) (2011). Available from `http://www.gecode.org/`
14. Van Hentenryck, P., Deville, Y.: The *cardinality* operator: a new logical connective in constraint logic programming. In: ICLP 1991. MIT Press (1991)