The Symmetry Domain: Extended Abstract

Iain D Stalker and Patricia M Hill

School of Computing
University of Leeds
Leeds
LS2 9JT
ENGLAND
{stalker,hill}@comp.leeds.ac.uk

1 Introduction

The declarative nature of Constraint Logic Programming means that a constraint program echoes the problem specification and can be easily understood. However, to ensure that a program is efficient enough to solve the problem in an acceptable amount of time, considerable further program development effort is often required. One potential cause of significant inefficiency is the presence of symmetry. A symmetry partitions a solution space into equivalence classes: each class contains only solutions or no solutions.

A number of techniques have been devised to eliminate or exploit symmetries once they have been identified [1–4]. On the other hand, there appears to be little research on how symmetries occurring in a constraint program may be detected automatically. This paper describes our work on this issue and proposes a technique based on abstract interpretation [5, 6] using a new symmetry domain.

2 Preliminaries

We assume a familiarity with sets, lattices [7] and abstract interpretation [6, 5]. If S is a set, $\wp(S)$ denotes the power set of S, $\wp_{\mathrm{f}}(S)$ denotes the set of all finite subsets of S and $\sharp S$ the cardinality of S. A bijective mapping $\rho : S \to S$ is called *a renaming* of the elements of S. We denote the set of all renamings of the set S by R_S. The renaming which maps each element to itself is called the *identity renaming* and is denoted ι.

Let $\mathcal{L}$ be a first order language over a set of function symbols, F, and predicate symbols, P, and a countable set of variables, *Vars*. We associate with each symbol in F and P an arity $n \in \mathbb{N}$. It is assumed that $= /2 \in P$. Let D be the domain of interpretation and $\mathcal{T}$ a first order theory in $\mathcal{L}$ over D. An atomic formula $p(x_1, \ldots, x_n)$, where $p \in P$ has arity n and $\{x_1, \ldots, x_n\} \in \wp_{\mathrm{f}}(\mathit{Vars})$, is called a *primitive constraint on* $\{x_1, \ldots, x_n\}$ *with predicate* p/n. We let $\mathcal{C}^\flat$ denote the set of all primitive constraints over *Vars*. A *constraint*, $c^\flat(S)$, on $S \in \wp_{\mathrm{f}}(\mathit{Vars})$ is a conjunction of a finite number of primitive constraints, i.e. $c^\flat = \wedge_{i=1}^m p_m, p_i \in \mathcal{C}^\flat, m \in \mathbb{N}$. A *constraint store* $C^\flat \in \wp_{\mathrm{f}}(\mathcal{C}^\flat)$ *on* $S \in \wp_{\mathrm{f}}(\mathit{Vars})$

is a finite set of constraints on subsets of S. We denote the variables of a particular constraint store $C^\flat$ by $vars(C^\flat)$. If p/n is a primitive constraint on a set of variables $\{x_1, \ldots, x_n\}$ and $\rho : \{x_1, \ldots, x_n\} \to \{x_1, \ldots, x_n\}$ is a renaming of the variables, then we use $\rho(p)$ to denote $p(\rho(x_1), \ldots, \rho(x_n))$. This notation extends in a natural manner to constraint stores: $\rho(C^\flat)$ denotes the store $C^\flat$ after the renaming ρ has been applied to each of the variables in the store, $vars(C^\flat)$. With each point of interest on path of through a given program, $\mathcal{P}$, we associate the store of all constraints on that path from a chosen starting point to the point of interest. We interpret each constraint store, $C^\flat$, as the conjunction of the constraints it contains and write $\mathcal{T} \vdash \forall V C^\flat$ if $\mathcal{T} \vdash \forall \bigwedge C^\flat$. We treat a single constraint as the constraint store associated with the program path which begins and ends at that constraint: we refer to such a program path as *degenerate*.

We distinguish two kinds of constraints: those which which imply an order on the variables used, for example, $x < y$; those which do not, for example, $x \neq y$. We consider only the second kind in this paper and refer to them as containing *local symmetries*.

Let A and B be sets. The *symmetric difference* of A and B is $\mathrm{sd}(A, B) \stackrel{\mathrm{def}}{=} (A \setminus B) \cup (B \setminus A)$.

Lemma 1. *Let A and B be two sets. Then* $\mathrm{sd}(A, B) = (A \cup B) \setminus (A \cap B)$

This result motivates our definition of a "generalised" symmetric difference of sets.

Definition 1. (Generalised Symmetric Difference) *Let $A_1, \ldots, A_n \subseteq V$ be subsets of a set of variables of interest V. Then $\mathrm{sd} : \wp(\wp(V)) \to \wp(V)$ is defined*

$$\mathrm{sd}(\{A_1, \ldots, A_n\}) \stackrel{\mathrm{def}}{=} \left(\bigcup_{j=1}^{n} A_j \right) \setminus \left(\bigcap_{j=1}^{n} A_j \right) \tag{1}$$

3 The Concrete Domain

Let $V \in \wp_{\mathrm{f}}(Vars)$ be our variables of interest. Suppose $\mathcal{P}$ is a constraint logic program and G is a goal with $V \supseteq vars(G)$. We wish to detect symmetries between the elements of V occurring in the execution $\mathcal{P} \cup \{G\}$.

Definition 2. (Acceptable Renamings) *Let $S \in \wp_{\mathrm{f}}(Vars)$. Let $C^\flat$ be a constraint store on S. The* set of acceptable renamings, $A_{C^\flat}$, for the constraint store $C^\flat$ *is*

$$A_{C^\flat} \stackrel{\mathrm{def}}{=} \{\rho \mid \rho \in R_X, X \subseteq S, \mathcal{T} \vdash \forall(C^\flat \leftrightarrow \rho(C^\flat))\} \tag{2}$$

We will refer to the set of all acceptable renamings for a constraint store as the renaming set.

The set of acceptable renamings of a constraint store, $C^\flat$, captures the symmetries among the variables of the store, $vars(C^\flat)$, deriving from the conjunction of the constraints it contains.

Example 1. Suppose we have the constraint store $C^\flat = \{x \neq y, y \neq z, z \neq x\}$. $A_{C^\flat} = \{\{x/y, y/z, z/x\}, \{x/z, y/x, z/y\}, \{x/y, y/x\}, \{x/z, z/x\}, \{y/z, z/y\}, \iota\} = R_{\{x,y,z\}}$. Here, we observe a complete symmetry between the variables x, y and z. This is reflected by the renaming set $R_{\{x,y,z\}}$.

Notation We introduce the notation

$$\langle R_{S_1}, \ldots, R_{S_k} \rangle \stackrel{\text{def}}{=} \{\rho_1 \cup \ldots \cup \rho_k \mid \rho_i \in R_{S_i}, i = 1, \ldots, k\}$$

where $S_1, \ldots, S_k$ are sets of variables.

Lemma 2. *For any constraint store $C^\flat$, $A_{C^\flat} = \langle R_{S_1}, \ldots, R_{S_k} \rangle$, where $S_i \subseteq vars(C^\flat)$, $S_i \neq \varnothing, S_i \cap S_j = \varnothing, i \neq j, i, j = 1, \ldots, k$ and $k \in \mathbf{N}$. In particular,*

1. *$A_{C^\flat} = R_{vars(C_i^\flat)}$ when $k_i = 1$ and $S_1 = vars(C^\flat)$.*
2. *$A_{C^\flat} = \{\iota\}$ when $k = 1$, $S_1 = vars(C^\flat)$ and $R_{S_1} = \{\iota\}$.*

We establish a connection between a constraint store associated with a given program path and the constraints it contains, considered as degenerate paths, through conjunction.

Conjunction often induces symmetry between variables of distinct constraints deriving from the same predicate, or the same combination of predicates. However, this does not apply when the constraints derive from different predicates.

Example 2. Consider, $x \neq y \wedge z \neq y$ is equivalent to $z \neq y \wedge x \neq y$. We can see immediately that the second expression can be derived from the first by the renaming $\{x/z, z/x\}$. Thus, we have an symmetry between x and z, which was not present in either of the individual constraints, but which was induced by the commutativity of conjunction.

Example 3. Consider, $x \neq y \wedge z = y$ is equivalent to $z = y \wedge x \neq y$, but the second expression cannot be derived from the first by a renaming.

Thus, when treating a combination of local symmetries, we want to know whether these derive from the same predicate, or the same combination of predicates, or from different predicates. We introduce a mechanism to facilitate this.

Definition 3. (Catalogue) *Let S be a set. Suppose we have a partition of S $\{P_1 \mid P_2 \mid \cdots \mid P_k\}$, for some $k \in \mathbb{N}$. Construct a family of distinct subsets of S, $\mathcal{I} : \mathbb{N} \to \wp(S)$, $I(i) = P_i$ for $i = 1, \ldots, k$. Extend $I(n)$ to include $n = 0$ by defining $I(0) = \varnothing$.*

We call $\mathcal{I}(S) \stackrel{\text{def}}{=} \{(x, i) \mid x \in I(i), i = 0, 1, \ldots, k\}$ a catalogue *of S. We refer to S as* the feature set *and $\{0, 1, \ldots, k\}$ as the* index *of S.*

Since $\mathcal{C}^\flat$ is set of primitive constraints, we can view $\wp(\mathcal{C}^\flat)$ as a set of constraint stores. We use the notion of a catalogue to associate an index with each constraint store. We first partition $\wp(\mathcal{C}^\flat)$ as follows:

1. Collect together all constraint stores with $\sharp\{vars(C^\flat)\} = i$ into the set Ξ_i, for $i = 1, \ldots, n$, where $n = \sharp V$. The nonempty Ξ_i form a partition of $\wp(\mathcal{C}^\flat)$.
2. For each nonempty Ξ_i, gather into distinct subsets those constraints deriving from the same predicate, or the same combination of predicates, and denote each by $[C^\flat_{ij}]$, for some suitable $i \in [1, \ldots, n]$ and $j = 1, \ldots, k_i$ for some $k_i \in \mathbb{N}$.

$\{[C^\flat_{11}], \ldots, [C^\flat_{nk_n}]\}$ is a partition of $\wp(\mathcal{C}^\flat)$. We construct a catalogue of this partition using Definition 3.

Example 4. Suppose we have a set of n variables, S. Let $\mathcal{C}^\flat = \{p/n, q/n, r/n\}$. Thus, p/n, q/n and r/n are three primitive constraints on S. Then

$$\wp(\mathcal{C}^\flat) = \{\varnothing, \{p(S)\}, \{q(S)\}, \{r(S)\}, \{p(S), q(S)\}, \{p(S), r(S)\}, \\ \{q(S), r(S)\}, \{p(S), q(S), r(S)\}\}$$

represents all possible constraint stores on S which can be composed from the primitives of $\mathcal{C}^\flat$. We construct the partition

$$\wp(\mathcal{C}^\flat) = \{\varnothing \mid \{p(S)\} \mid \{q(S)\} \mid \{r(S)\} \mid \{p(S), q(S)\} \mid \{p(S), r(S)\} \\ \mid \{q(S), r(S)\} \mid \{p(S), q(S), r(S)\}\}.$$

We use Definition 3 to construct the catalogue

$$\mathcal{I}(\mathcal{C}^\flat) = \{(\varnothing, 0), (\{p(S)\}, 1), (\{q(S)\}, 2), (\{r(S)\}, 3), (\{p(S), q(S)\}, 4), \\ (\{p(S), r(S)\}, 5), (\{q(S), r(S)\}, 6), (\{p(S), q(S), r(S)\}, 7)\}.$$

Example 5. Suppose we have a set of variables, $\{x, y, z\}$. Let $\mathcal{C}^\flat = \{p(x', y') \mid \{x', y'\} \subset \{x, y, z\}\}$. Then

$$\wp(\mathcal{C}^\flat) = \{\varnothing, \{p(x, y)\}, \{p(y, x)\}, \{p(x, z)\}, \{p(z, x)\}, \{p(y, z)\}, \{p(z, y)\}\}$$

represents all possible constraint stores on $\{x, y, z\}$ which can be composed from $\mathcal{C}^\flat$. In this case our partition is

$$\wp(\mathcal{C}^\flat) = \{\varnothing \mid \{p(x, y)\}, \{p(y, x)\}, \{p(x, z)\}, \{p(z, x)\}, \{p(y, z)\}, \{p(z, y)\}\}.$$

We use Definition 3 to construct the catalogue

$$\wp(\mathcal{C}^\flat) = \{(\varnothing, 0), (\{p(x, y)\}, 1), (\{p(y, x)\}, 1), (\{p(x, z)\}, 1), (\{p(z, x)\}, 1), \\ (\{p(y, z)\}, 1), (\{p(z, y)\}, 1)\}.$$

We expand our notation to embrace these indices. Each set of acceptable renamings of a given constraint store inherits an index from the catalogue $\mathcal{I}(\mathcal{C}^\flat)$: for example, if $C^\flat$ has index j in the $\wp(\mathcal{C}^\flat)$, then we write $(A_{C^\flat}, j)$.

Example 6. In Example 4, the set of acceptable renamings for $\{p(S), q(S)\}$ has extended notation $(A_{\{p(S), q(S)\}}, 4)$

Furthermore, recalling Lemma 2, $A_{C^\flat} = \{R_{S_1}, \ldots, R_{S_k}\}$, for some appropriate $k \in \mathbb{N}$. We associate the index of $A_{C^\flat}$ with the R_{S_i}. Thus, $(A_{C^\flat}, j) = \{(R_{S_1}, j), \ldots, (R_{S_k}, j)\}$.

Example 7. Suppose $C^\flat = \{x \neq y, vw = 6\}$ has index 3. Then $(A_{C^\flat}, 3) = \langle (R_{\{x,y\}}, 3), (R_{\{v,w\}}, 3) \rangle$.

We choose not to associate an index with the concrete element corresponding to our final abstract element, for two reasons:

1. The use of indices is a convenience to assist in tracking the constraints from which our abstract elements derive (see section 4). Once we have our treated our abstract elements, these indices are incidental.
2. The renaming set which derives from the concretisation of our final abstract element will typically contain information in addition to the renamings we seek. We do not know precisely to which constraint (or constraints) the renaming set applies, only that our desired renaming set is a subset of this final set.

Accordingly, we introduce "$\cdot$" as a marker to indicate a possible choice of index when the particular index is incidental.

Our concrete domain is

$$\mathcal{D}_{\mathcal{C}^\flat} \stackrel{\text{def}}{=} \{R \mid R \in \wp((R_S, n)), S \subseteq V, n \in \mathbb{N} \cup \{\cdot\}\} \tag{3}$$

which we partially order using subset inclusion. $\mathcal{D}_{\mathcal{C}^\flat}$ is a complete lattice having set intersection as the meet operator, set union as the join operator, least element $\varnothing$ and greatest element $\top = \wp((R_S, n))$: $\mathcal{D}_{\mathcal{C}^\flat} = (\mathcal{D}_{\mathcal{C}^\flat}, \subseteq, \cup, \cap, \varnothing, \wp((R_S, n)))$.

4 The Abstract Domain

We define the symmetry domain for a given set of variables of interest, V, to be $SY \stackrel{\text{def}}{=} \wp(SX)$, where $SX = \wp(V) \setminus \varnothing \times \mathbf{N} \cup \{\cdot\}$, and order by subset inclusion. The elements of SY sets of pairs, where each pair is of the form $(S, \nu), S \subseteq V, \nu \in \mathbb{N} \cup \{\cdot\}$. as in our concrete domains, we use "$\cdot$" to indicate a possible choice of index and use this when the index is incidental. SY is a complete lattice having set intersection as the meet operator, set union as the join operator, least element $\bot = \varnothing$ and greatest element $\top = \wp(SX)$: $SY = (SY, \subseteq, \cup, \cap, \varnothing, \wp(SX))$. Define an equivalence relation, $\sim$, $sy \in SY$, as follows:

$$(S, \mu) \sim (T, \nu) \Leftrightarrow \mu = \nu, (S, \mu), (T, \nu) \in sy.$$

We denote the equivalence class of $(S, \nu) \in sy$ by $[(S, \nu)]$. The equivalence classes of $\sim$ form a partition of sy.

Definition 4. (Symmetrisation) *Let $sy \in SY$ and let $[(S,\nu)] \subseteq sy$ be an equivalence class in sy. Suppose $[(S,\nu)] = \{(T_1,\nu),\ldots,(T_m,\nu)\}$. We define the* symmetrisation *of $[(S,\nu)]$, to be*

$$\sigma([(S,\nu)]) \stackrel{\text{def}}{=} \{(U,\nu),(V,\nu)\}, \forall[(S,\nu)] \in sy$$

where $U = \cap\{T_1,\ldots,T_m\}$ and $V = \mathrm{sd}(\{T_1,\ldots,T_m\})$, and $U \cap V = \varnothing$

We define the symmetrisation *of $sy = \{[(S_1,\nu_1)],\ldots,[(S_n,\nu_n)]\}$ to be*

$$\Sigma(sy) \stackrel{\text{def}}{=} \{\sigma([(S_1,\nu_1)]),\ldots,\sigma([(S_n,\nu_n)])\}$$

Definition 5. (Desymmetrisation) *Let $sy \in SY$. Let $\Sigma(sy) = \{(U_1,\nu_1), (V_1,\nu_1),\ldots,(U_k,\nu_k),(V_k,\nu_k)\}$ be the symmetrisation of sy. $\Sigma(sy)$ is partitioned into k classes. We denote the jth class cl_j and thus, $\Sigma(sy) = \{cl_1,\ldots,cl_k\}$. We introduce the* desymmetrisation algorithm*:*

```
begin
  do i = 1 to k − 1
    do j = 2 to k
      for each (S, ν_i) ∈ cl_i and (T, ν_j) ∈ cl_j
        if S ∩ T ≠ ∅
          replace (S, ν_i) by (S ∩ T, ν_i), (S \ (S ∩ T), ν_i)
          replace (T, ν_j) by (S ∩ T, ν_j), (T \ (S ∩ T), ν_j)
        end if
      end for
    end do
  replace ν_i by ·
  end do
  replace ν_k by ·
stop.
```

The application of the desymmetrisation algorithm to the symmetrised form of $sy \in SY$ is called the desymmetrisation *of sy. The resulting set is called the* desymmetrised form *of sy and we denote it $\Delta(sy)$.*

Desymmetrisation and symmetrisation are used primarily in the combination of elements of SY. However, we make use of these operators to define our concretisation function, $\gamma : SY \to \mathcal{D}_{\mathcal{C}^\flat}$.

Definition 6. (Concretisation) *Let $sy \in SY$ have desymmetrised form $\Delta(sy) = \{(T_1,\nu_1),\ldots,(T_m,\nu_m)\}$ where $\nu_i \in \mathbb{N} \cup \{\cdot\}$ for $i = 1,\ldots,m$. The concretisation of SY, $\gamma : SY \to \mathcal{D}_{\mathcal{C}^\flat}$, is defined to be:*

$$\gamma(sy) \stackrel{\text{def}}{=} \{(R_{T_1},\nu_1),\ldots,(R_{T_m},\nu_m)\}$$

where (R_{S_i},ν_i) is the renaming set of $S_i \subseteq V$.

$\gamma : SY \to \mathcal{D}_{\mathcal{C}^\flat}$ (uniquely) defines the abstraction function [5, 6], $\alpha : \mathcal{D}_{\mathcal{C}^\flat} \to SY$,

$$\alpha((A_{C^\flat},\nu)) = \bigcap\{sy \mid (R_{C^\flat},\nu) \subseteq \gamma(sy)\}.$$

In the concrete domain, we established a connection between a constraint store associated with a given program path and the constraints it contains, considered as degenerate paths, through conjunction. To reflect this in the abstract domain, we introduce an abstract conjunction operator.

Definition 7. (Abstract Conjunction) *We define the* abstract conjunction *of $sy_1, \ldots, sy_m \in SY$, $\wedge^\sharp : SY \to SY$, to be*

$$\wedge^\sharp(\{sy_1, \ldots, sy_m\}) \stackrel{\text{def}}{=} sy \cup \Delta(\mathrm{sd}(sy_1, \ldots, sy_m)).$$

where $sy = sy_1 \cap \ldots \cap sy_m$.

Example 8. Consider `all_different(A,B,C)` and `all_different(A,C,D)`. Abstracting these yields the sets $\{(\{a,b,c\},\nu)\}$ and $\{(\{a,c,d\},\nu)\}$, respectively, for some suitable $\nu \in \mathbb{N}$. Since these derive from the same predicate, they have the same index and we have:

$$\wedge^\sharp(\{\{(\{a,b,c\},\nu)\},\{(\{a,c,d\},\nu)\}\}) = \{(\{a,c\},\nu),(\{b,d\},\nu)\}.$$

Example 9. Consider `W+X+Y=Z+U` and `not_equal(X,Y)`. Abstracting these yields the sets $\{(\{w,x,y\},\mu),(\{z,u\},\mu)\}$ and $\{(\{x,y\},\nu)\}$, respectively, for some suitable $\mu, \nu \in \mathbb{N}$. Since these derive from different predicates, $\mu \neq \nu$ and we have:

$$\wedge^\sharp(\{\{(\{w,x,y\},\mu),(\{z,u\},\mu)\},\{(\{x,y\},\nu)\}\}) = \{(\{x,y\},\cdot),(\{z,u\},\mu),(\{w\},\nu)\}.$$

We use the foregoing concepts to detect the potential symmetries attendant upon a particular program path. We illustrate how in the next section. The following theorem confirms that our approximations are safe, i.e., we do not lose information by working in SY.

Theorem 1. *Let $C_i^\flat \subseteq \mathcal{C}^\flat$, $i = 1, \ldots, m$, be constraint stores, each of which contains a single constraint $c_i^\flat \in \mathcal{C}^\flat$. Associate with each store the index of $\{c_i^\flat\}$ in $\wp(\mathcal{C}^\flat)$, ν_i. Then*

$$(A_{C_1^\flat \wedge \cdots \wedge C_m^\flat}, \mu) \subseteq \gamma\left(\wedge^\sharp\left(\alpha((A_{C_1^\flat},\nu_1)), \ldots, \alpha((A_{C_m^\flat},\nu_m))\right)\right)$$

where μ denotes the index of the conjunction of the constraint stores.

5 Applications

We present a simple application of the abstract interpretation. As we use only a single predicate, we dispense with indexing.

Consider the graph illustrated in Figure 1. A simple colouring problem applied to this graph is expressed in the following program:

```
:- use_module(library(clpfd)).

colouring(A,B,C,D,E,F,N) :-
```

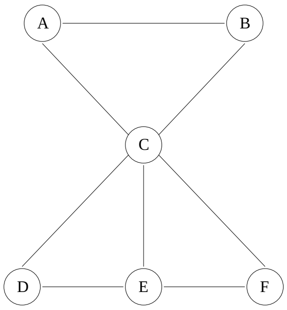

Fig. 1. A Simple Colouring Graph

```
    domain([A,B,C,D,E,F],1,N),
    labeling([],[A,B,C,D,E,F]),
    cdiff(A,B,C,D,E,F).

  cdiff(A,B,C,D,E,F) :-
    all_different([A,B,C]),                 !!  (c1)
    all_different([C,D,E]),                 !!  (c2)
    all_different([C,D,F]).                 !!  (c3)
```

We abstract the sets of acceptable renamings for the individual constraints:

(c1) $\{\{a, b, c\}\}$.
(c2) $\{\{c, d, e\}\}$.
(c3) $\{\{c, d, f\}\}$.

These derive from a single program path: we locate these in the same constraint store. Taking the abstract conjunction of this yields:

$$\wedge^{\sharp}(\{\{a, b, c\}\}, \{\{c, d, e\}\}, \{\{c, d, f\}\}) = \{\{a, b\}, \{e, f\}, \{c\}, \{d\}\}$$

and the concretisation of this is

$$\gamma(\{\{a, b\}, \{e, f\}, \{c\}, \{d\}\}) = R_{\{a,b\}} \cup R_{\{e,f\}} \cup R_{\{c\}} \cup R_{\{d\}}$$

This indicates that there are potential symmetries between a and b and between e and f. These sets provide the means to break the symmetries: imposing an order on each of $\{a, b\}$ and $\{e, f\}$ prevents our revisiting the same equivalence class for a given solution. The concretisation also suggests that both c and d are self-similar, that is, there may be a set of fully interchangeable values [8] for each of these.

6 Conclusion

We have developed an abstract interpretation to assist in the detection of symmetries within a given program. At present that this method focuses on identifying those symmetries which derive from a conjunction of locally symmetric constraints as stated in Section 2. We discover possible global symmetries which result from a "carry through" of local symmetries; or which are conferred through the commutativity of conjunction, as illustrated in Example 2. We have not considered those constraints which impose an order, as distinguished in Section 2. This has two consequences: we can only say that there is a possible symmetry which must checked manually at present; we do not detect symmetries which might arise from a conjunction of constraints which are locally nonsymmetric, for example, the conjunction of $x < y$ and $x < z$ implies a possible symmetry between y and z. Locally nonsymmetric constraints will be addressed in future work.

We observe that the domain SY without indices has been used elsewhere, for example, the set sharing domain [9]. Thus, we can use the above abstract interpretation in conjunction with other program analyses with a minimum extra cost.

References

1. I P Gent and B M Smith. Symmetry breaking during search in constraint programming. In W Horn, editor, *ECAI 2000*, pages 599–603, 2000.
2. I P Gent and B M Smith. Symmetry breaking during search in constraint programming. Research report 99.02, School of Computing, University of Leeds, January 1993.
3. R Backofen and S Will. Excluding symmetries in constraint-based search. In *Principles and Practice of Constraint Programming - Cp'99*, volume LNCS 1713, pages 73–87. Springer, 1999.
4. J-F Puget. On the satisfiability of symmetrical constrained satisfaction problems. In H J Komorowski and Z W Ras, editors, *Methodologies for Intelligent Systems, 7th International Symposium, ISMIS '93, Trondheim, Norway,June 15-18, 1993, Proceedings. Lecture Notes in Computer Science*, volume 689, pages 350–361. Springer, 1993.
5. P Cousot and R Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.
6. P Cousot and R Cousot. Abstract interpretation and applications to logic programs. *Journal of Logic Programming*, 13(2&3):103–179, 1992.
7. G Birkhoff. *Lattice Theory (3rd Edition)*. AMS Colloquim Publication, Providence, RI, 1967.
8. E Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *AAAI'91*, volume LNCS 1713, pages 227–233, 1991.
9. D Jacobs and A Langen. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming*, 13(2&3):291–314, 1992.