Excluding Symmetries in Concurrent Constraint Programming

Rolf Backofen and Sebastian Will
Institut für Informatik, LMU München
Oettingenstraße 67, D-80538 München
Email: {backofen,wills}@informatik.uni-muenchen.de

1 Introduction

In many problems (especially those concerned with combinatorial search), non-determinism naturally arises in an concurrent constraint-based formulation in form of disjunctive constraints. To check consistency of disjunctive constraints, one needs to perform search by splitting the constraint store into local computation stores.

In formulizing disjunctive constraints, an important problem is to eliminate symmetries which arise from this kind of formulation. The symmetries give rise to many different solutions found by the search procedure, which are all considered to be "similar". Consider for example constraint problems, where finite domain variables have a geometric interpretation such as in the N-queens problem or the square-tiling problem (where set of squares of given sizes must fit exactly into a fixed square). A newer and more complex problem is the lattice protein structure prediction problem [2]. In this case, different solutions are symmetric if they can be generated using reflections or rotations. In other problems, symmetric solutions can be generated by performing permutations on variables valuations. In other problems, symmetric solutions can be generated by permuting the solution.

In many cases, one is not interested in also getting all the symmetric solutions to every found solution. Then, generally those symmetries will give rise to an (often exponential) amplification of the search space. Hence, symmetry exclusion promises to efficiently prune the search tree.

For this reason, many constraint programmers have constructed mechanisms for their programs to exclude at least some of the symmetries of their problem. Unfortunately, symmetry exclusion was often not straightforward to implement and had to be redesigned for every special problem or enumeration strategy, which leaded to an inflexibility in the program structure once it was introduced.

Thus, the widespread usage of symmetry exclusion was strongly hindered by its complexity. Often symmetry exclusion was not done at all or only for a small set of symmetries. In other cases, where symmetry exclusion was implemented it distracted the programmers attention from more important tasks.

The contribution of our work is to give a generally applicable method for the exclusion of arbitrary symmetries. To our knowledge, this is the first general and declarative method for excluding symmetries in concurrent constraint programming. Note that our notion of symmetries extend previously used definitions. In [3], the symmetries considered are only symmetries which involve the valuation of one variable. This restricted notion of symmetries does not include geometrical symmetries such as rotation and reflection, which map tuples of valuations of variables to symmetric tuples. In [1], only permutations of Boolean variables are considered. Thus, again for example geometric symmetries and symmetries on finite domain variables cannot be considered.

We have implemented our method in the concurrent constraint programming language Oz [6] using the programmable search engine described in [5]. Although we describe our technique in the case where we have and-parallelism but no or-parallelism, the technique can be extended to the case where we have in addition angelic or-parallelism as e.g. described in [4].

In the next section, we will give some preliminaries and define the notion of symmetries. In Section 3, we will explain the concept of symmetries in the case where we have finite domain variables with a geometric interpretation. We will then introduce the technique of symmetry exclusion in Section 4 and prove its correctness. Finally, we will discuss how to apply the symmetry exclusion technique in the case of permutations, using the photo problem as an example.

2 Preliminaries

We fix a first-order signature Σ including the equality $\doteq$ with a set of variables $\mathcal{V}$. Constraints are literals, and constraint formulae are quantifier-free formulae over Σ. We identify $t \doteq t'$ with $t' \doteq t$. $\mathcal{C}$ denotes the set of all constraints. A set of constraints $C \subseteq \mathcal{C}$ is interpreted as the conjunction of the constraints contained in C, and we will freely mix set notation and conjunction.

We fix a standard interpretation $\mathcal{A}$, which describes our constraint theory. A constraint set C *determines* a set of variables $\mathcal{X}$ to an assignement α, iff for all $x \in \mathcal{X}$ exists a t such that $x \doteq t \in C$.

In the following, we assume a fixed constraint set C_{Pr} describing the problem to be solved.

Definition 2.1 *An* assignment α in $\mathcal{A}$ *is a partial function*

$$\alpha : \mathcal{V} \to \mathcal{D}^{\mathcal{A}}.$$

We say that an (possible partial) assignment α satisfies *ϕ (short $\alpha \models \phi$) if there is a total assignment $\alpha' \supseteq \alpha$ with $\mathcal{A}, \alpha' \models \phi$. We write $\phi \models \psi$ if for every α we have $\alpha \models \phi \to \psi$.*

In many problems, one has a set of solution variables, and some auxilary variables whose valuations are uniquely determined by the valuation of the solution variables. This is captured by the next definition.

Definition 2.2 (Solution Variables) *Let $\mathcal{X} \subseteq \mathcal{V}$ be a set of variables. $\mathcal{X}$ is a* solution variables set for *C if*

$$\forall\alpha\,[(\mathrm{dom}(\alpha) = \mathcal{X} \wedge \alpha \models C) \Rightarrow (\exists!\alpha'(\alpha \subseteq \alpha' \wedge \mathrm{dom}(\alpha') = \mathcal{V} \wedge \alpha' \models C)]\,,$$

where $\exists!\alpha\phi(\alpha)$ ('exists a unique α satisfying $\phi(\alpha)$') is short for $\exists\alpha(\phi(\alpha) \wedge \forall\alpha' : \phi(\alpha') \Rightarrow \alpha = \alpha')$. For a solution variables set $\mathcal{X}$, we say that α is a $\mathcal{X}$-solution for *C if* $\mathrm{dom}(\alpha) = \mathcal{X}$ *and $\alpha \models C$. With $\|C\|^{\mathcal{X}}$ we denote the set of $\mathcal{X}$-solutions of C.*

In the following, we fix $\mathcal{X}$. Hence, we use the term 'solution' as short for '$\mathcal{X}$-solution for C', and we write $\|C\|$ as short for $\|C\|^{\mathcal{X}}$.

Definition 2.3 (Symmetry) *A* symmetry *s* for *C_{Pr} is a total injective function*

$$s : \|C_{\mathrm{Pr}}\| \to \|C_{\mathrm{Pr}}\|.$$

A symmetry set *S* for *C_{Pr} is a set of symmetries operating on $\|C_{\mathrm{Pr}}\|$, which is closed under inversion. A* symmetry group *S is a symmetry set for C_{Pr} which is a group.*

Note that for every symmetry s for C_{Pr}, also s^{-1} is a symmetry for C_{Pr}. We denote the identity function on $\|C_{\mathrm{Pr}}\|$ with $\mathrm{id}_{C_{\mathrm{Pr}}}$ (which is a symmetry by definition). Clearly, the set of all symmetries for C_{Pr} is a group. But in general, we do not want to consider all symmetries, since either there are too many of them, or some of them do not have an intuitive characterisation. In this case, we consider only a subset of all symmetries, which may not be a group.

3 Geometric Symmetries

We will give now a concrete example for symmetries, namely geometry symmetries. There are many problems where one encodes the two-dimensional (resp. three-dimensional) position of an object i using finite domain integer variables X_i, Y_i (resp. X_i, Y_i, Z_i). Examples are the N-queens and the tiling problem. A more complex and newer example is the lattice protein structure prediction problem [2].

Using finite domain integers variables for encoding the position of an object i implies that we interpret the position of i to be in the cubic lattice $\mathbb{Z}^d$, where $d = 2$ or $d = 3$. Thus, the allowed symmetries are defined by affine mappings $S : \mathbb{Z}^d \to \mathbb{Z}^d$ with $S(\vec{x}) = A_S\vec{x} + \vec{v}_S$ that map $\mathbb{Z}^d$ onto $\mathbb{Z}^d$. I.e., the matrix A_S is an orthogonal matrix with the property that the set of columns $\{\vec{v}_1, \ldots, \vec{v}_d\}$ of A_S equals

$$\{\pm\vec{e} \mid \vec{e} \text{ is a unit-vector of } \mathbb{Z}^d\}$$

E.g., the matrix

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

denotes the rotation by 90°. For $\mathbb{Z}^3$, we have $6 \times 4 \times 2$ such matrices, and henceforth 47 non-trivial symmetries.

The problem is that the vector $\vec{v}_S$ is not yet fixed. But in many constraint problems with geometric variables, there is a simple condition which allows one to fix this vector. This condition is just that every possible solution lies within a fixed square (in two dimension) or cube (in three dimension). E.g., this condition states for two dimensions that for all $\alpha, \alpha' \in \|C_{\mathrm{Pr}}\|$ we have

$$\begin{aligned} \min\{\alpha(X_i)\} &= x_{min} = & \min\{\alpha'(X_i)\} \\ \max\{\alpha(X_i)\} &= x_{max} = & \max\{\alpha'(X_i)\} \\ \min\{\alpha(Y_i)\} &= y_{min} = & \min\{\alpha'(Y_i)\} \\ \max\{\alpha(Y_i)\} &= y_{max} = & \max\{\alpha'(Y_i)\}. \end{aligned}$$

Thus, in every solution the minimal square around the position of all objects is defined by the points (x_{min}, y_{min}) and (x_{max}, y_{max}). We call this the *frame* of the problem C_{Pr}. In the N-queens problem, this is just the board.

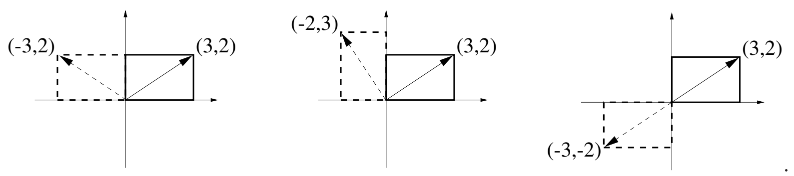
Now knowing the frame of a problem C_{Pr}, we can fix the vector $\vec{v}_S$ for all symmetries. Consider as an example a problem whose frame is defined by $(0,0)$ and $(3,2)$. Furthermore, consider the three symmetries reflection at the y-axis, rotation by 90° and rotation by 180°, which we will name S_1, S_2 and S_3 in the following. The corresponding mappings are defined by

$$S_i(\vec{x}) = A_{S_i}(\vec{x}) + \vec{v}_{S_i},$$

where the matrices A_{S_1}, A_{S_2} and A_{S_3} are defined by

$$A_{S_1} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad A_{S_2} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad A_{S_3} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The corresponding mappings of the frame are



A symmetry S is compatible with the above defined frame if the frame is mapped to itself, i.e., if $\{\vec{v} \mid (0,0) \le \vec{v} \le (3,2)\} = \{S(\vec{v}) \mid (0,0) \le \vec{v} \le (3,2)\}$. For a given matrix A_S, there exists a $\vec{v}_S$ such that $S(\vec{x}) = A_S\vec{x} + \vec{v}_S$ satisfies this condition if and only if A_S satisfies

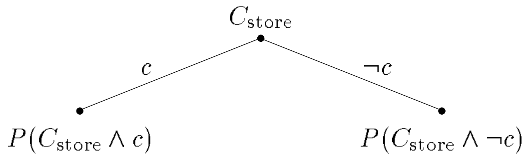
$$A_S(3,2) = (\pm 3, \pm 2) \tag{1}$$

For the matrices A_{S_1}, A_{S_2} and A_{S_3}, we get $(-3,2)$, $(-2,3)$ and $(-3,-2)$, which excludes the symmetry characterized by A_{S_2}. We finally get

$$\vec{v}_{S_1} = (3,0) \quad \text{and} \quad \vec{v}_{S_3} = (3,2).$$

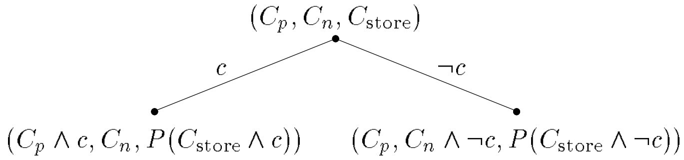
4 Symmetry Excluding Search Trees

Now we have to define the notion of a search tree as we need it for the symmetry exclusion. Usually, a search tree is a labeled tree, where the node are labeled with the constraints in the constraint store, and the edges are labled by constraints or negated constraints. A single node v is of the form



$P(C_{\text{store}} \wedge c)$ (resp. $P(C_{\text{store}} \wedge \neg c)$) just denotes the closure of $C_{\text{store}} \wedge c$ (resp. $C_{\text{store}} \wedge \neg c$) under constraint propagation. The root node is labeled with $P(C_{\text{Pr}})$.

But we additionally want to store at a node v also the positive and negative constraints that have been used along the path from the root to v to generate the node v. Thus, the nodes are labeled with triples $(C_p, C_n, C_{\text{store}})$, and a node in the search tree has the form



The reason for distinguishing between positive and negative constraints is just that C_n describes the previously found solutions, i.e.

$$\forall v' \prec v \forall \alpha \in \|C_{\text{store}}(v')\| : (\alpha \models \neg C_n).$$

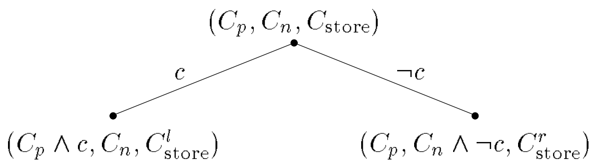
Here, we denote with $\prec_t$ the partial ordering of nodes induced by t.

But for the symmetry exclusion, we need also the flexibility to insert additional constraints during search in order to exclude symmetries. Thus, we have the following definition.

Definition 4.1 (Search Tree) *Let t be a finite, binary, rooted, ordered tree, whose egdes are labeled by literals, and whose nodes are labeled by triples of constraint sets. The tree t is a* search tree *for C_{Pr} if the following conditions are satisfied:*

1. *The root node v_r has the label $(\emptyset, \emptyset, C_{init})$, where C_{init} is the initial constraint store with $C_{init} \models\!| C_{\text{Pr}}$.*

2. *every binary node has the form*



with $C^l_{\mathrm{store}} \models C_{\mathrm{store}} \wedge c$ *and* $C^r_{\mathrm{store}} \models C_{\mathrm{store}} \wedge \neg c$

Note that we do not force C^l_{store} (resp. C^r_{store}) to be equivalent to $C_{\mathrm{store}} \wedge c$ (resp. $C_{\mathrm{store}} \wedge \neg c$). The reason just is we do not only want the constraint store to be closed under constraint propagation (as described above), but also may add some additional constraints during search for excluding symmetries. Since (as described above) C_{store} describes all constraints valid at some node v, we set $\|v\| = \|C_{\mathrm{store}}\|$ for a node v in t with label $(C_p, C_n, C_{\mathrm{store}})$.

Definition 4.2 (Expanded, C_{Pr}-Complete and S-Reduced Trees)
The search tree t is completely expanded *if every unary node* $v = (C_p, C_n, C_{\mathrm{store}})$ *satisfies*

$$(\ \|v\| = \{\alpha\} \text{ and } C_{\mathrm{store}} \text{ determines } \mathcal{X} \text{ to } \alpha \ ) \quad \text{or} \quad \perp \in C_{\mathrm{store}}$$

A search tree t is C_{Pr}-complete *if for every* $\alpha \in \|C_{\mathrm{Pr}}\|$ *there is a leaf v in t with* $\{\alpha\} = \|v\|$. *Let S be a symmetry group for* C_{Pr}. *A search tree is* C_{Pr}-complete w.r.t. S *if for every* $\alpha \in \|C_{\mathrm{Pr}}\|$ *there is a leaf v for such that*

$$\|v\| = \{\alpha\} \ \vee \ \exists s \in S \backslash \{\mathrm{id}_{C_{\mathrm{Pr}}}\} : \|v\| = \{s(\alpha)\}.$$

A search tree is S-reduced *if for every leaf v with* $\|v\| = \{\alpha\}$ *we have*

$$\forall s \in S \ \forall \mathit{leafs} \ v' \ \mathit{with} \ v' \prec_t v : (\|v'\| = \{\alpha'\} \Rightarrow s(\alpha') \neq \alpha), \tag{2}$$

Note that (2) is equivalent to $\forall s \in S \ \forall v' \neq v : (\|v'\| = \{\alpha'\} \Rightarrow s(\alpha') \neq \alpha)$, since S is closed under inversion.

In the following, we will consider only search trees where the edge labels are atomic constraints having only one free variable. Before we can show how to produce a S-reduced search tree that is C_{Pr}-complete w.r.t S, we first have to define how symmetries operate on constraints.

In the following, we will assume that for every constraint $c(x)$ with one variable and every s, there is a constraint $c'(y)$ such that

$$s^*(\|c(x) \wedge C_{\mathrm{Pr}}\|) = \|c'(y) \wedge C_{\mathrm{Pr}}\|,$$

where we define the function s^* on sets of C_{Pr}-solutions uniquely by

$$s^*(A) = \{s(\alpha) \mid \alpha \in A\}.$$

In the following, we fix a function s_{con} on constraints with one free variable such that

$$s^*(\|c \wedge C_{\mathrm{Pr}}\|) = \|s_{\mathrm{con}}(c) \wedge C_{\mathrm{Pr}}\|.$$

Usually, one can find such a function if one introduces additional variables. E.g., this is of course true if we define symmetries on variables denoting coordinates, where the symmetries are rotations and mirroring.

Proposition 4.3 *Let s_{con} be defined as before. Then*

$$\begin{array}{rcl} s^*(||c \wedge c' \wedge C_{\text{Pr}}||) & = & ||s_{\text{con}}(c) \wedge s_{\text{con}}(c') \wedge C_{\text{Pr}}|| \\ s^*(||(c \vee c') \wedge C_{\text{Pr}}||) & = & ||(s_{\text{con}}(c) \vee s_{\text{con}}(c')) \wedge C_{\text{Pr}}|| \\ s^*(||\neg c \wedge C_{\text{Pr}}||) & = & ||s_{\text{con}}(\neg c) \wedge C_{\text{Pr}}|| \end{array}$$

Hence, we identify s_{con} with its homomorphic extension to constraint sets and arbitrary formulae. Now we can state a general mechanism for generating a search tree that is S-reduced.

Definition 4.4 **(S-excluding)** *Let S be a symmetry set. A search tree for C_{Pr} is S-*excluding *if every binary node v has the form*

$$\begin{tikzcd} & (C_p, C_n, C_{\text{store}}) \arrow[dl, "c"'] \arrow[dr, "\neg c"] & \\ (C_p \wedge c, C_n, C^l_{\text{store}}) & & (C_p, C_n \wedge \neg c, C^r_{\text{store}}) \end{tikzcd}$$

with $C^l_{\text{store}} \models\!| C_{\text{store}} \wedge c$ *and*
$C^r_{\text{store}} \models\!| (C_{\text{store}} \wedge \neg c) \cup E^v(\neg c)$, *where*
$E^v(\neg c) = \{s_{\text{con}}(C_p) \to \neg s_{\text{con}}(c) \mid s \in S\}$

Before we prove S-reducedness and C_{Pr}-completeness w.r.t S of S-excluding search trees, we state a proposition to precise the effect of adding the implications in definition 4.4 to C^r_{store}, first. For convienence, we write C^v_p, C^v_n and C^v_{store} to access C_p, C_n and C_{store} in the label $(C_p, C_n, C_{\text{store}})$ of v. For a binary node v we refer to its left child as v_l and to its right child as v_r.

Proposition 4.5 *Let S be a symmetry set and t a S-excluding search tree. For every symmetry $s \in S$ and for every node v of t we have*

$$C^v_{\text{store}} \models s_{\text{con}}(C^v_p) \to s_{\text{con}}(C^v_n).$$

Proof. We prove this by tree induction. Hence fix a $s \in S$. For the root of t the proposition is valid, since $C_n = \emptyset$ and $\emptyset$ is identified with true. Let v be a binary node of t of the form

$$\begin{tikzcd} & (C_p, C_n, C_{\text{store}}) \arrow[dl, "c"'] \arrow[dr, "\neg c"] & \\ (C_p \wedge c, C_n, C^l_{\text{store}}) & & (C_p, C_n \wedge \neg c, C^r_{\text{store}}) \end{tikzcd} .$$

For v_l the claim follows directly by the induction hypothesis, since $C_{\text{store}} \models s_{\text{con}}(C_p) \to s_{\text{con}}(C_n)$ implies $C_{\text{store}} \wedge c \models s_{\text{con}}(C_p \wedge c) \to s_{\text{con}}(C_n)$.

For v_r we have to show

$$C_{\text{store}} \wedge \neg c \wedge E^v(\neg c) \models s_{\text{con}}(C_p) \to s_{\text{con}}(C_n \wedge \neg c),$$

Recall that $E^v(\neg c)$ denotes the set $\{s_{\text{con}}(C_p) \rightarrow \neg s_{\text{con}}(c) \mid s \in S\}$ according to the definition of S-excluding search tree.

From induction hypothesis follows, that

$$C_{\text{store}} \wedge \neg c \wedge E^v(\neg c) \models s_{\text{con}}(C_p) \rightarrow s_{\text{con}}(C_n),$$

therefore it remains to be shown

$$C_{\text{store}} \wedge \neg c \wedge E^v(\neg c) \models s_{\text{con}}(C_p) \rightarrow s_{\text{con}}(\neg c)$$

and this is obviously true, since $(s_{\text{con}}(C_p) \rightarrow s_{\text{con}}(\neg c)) \in E^v(\neg c)$. $\square$

For notational convenience we introduce the notation

$$C_{path}(v) = C_p \wedge C_n \wedge C_{\text{Pr}}.$$

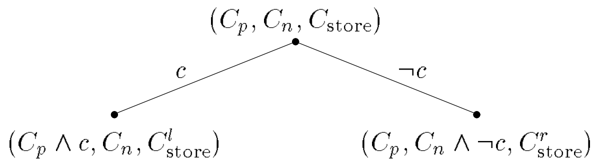
A good intuition is to think of $C_{path}(v)$ describing the constraint store of v in a simple not symmetry excluding search tree.

Reducedness and completeness will be essentially corollaries to the following proposition.

Lemma 4.6 *Let S be a symmetry set. Every S-excluding search tree t for C_{Pr} satisfies for every node v*

$$\begin{array}{rcl} \|v\| &=& \|C_{path}(v)\| - A_S^v, \\ \textit{where}\ \ A_S^v &=& \{s(\alpha) \mid s \in S \wedge \exists v' \prec_t v : \alpha \in \|C_{path}(v')\|\}. \end{array} \qquad (3)$$

Proof. We proof this by tree induction. For the root this is valid, since $A_S^v = \emptyset$ and $\|v\| = \|C_{\text{Pr}}\| = \|C_{path}(v)\|$. Assume that we have proven the claim for a binary node v:



For the left child v_l the claim follows immediately from the induction hypotheses, since $A_S^{v_l} = A_S^v$.

For v_r, we have to show $\|v^r\| = \|C_{path}(v_r)\| - A_S^{v_r}$. Note that $A_S^{v_r} = \{s(\alpha) \mid s \in S \wedge \exists v' \prec_t v_r : (\alpha \in \|C_{path}(v')\|)\}$ subdivides into its two subsets A_S^v and $\{s(\alpha) \mid s \in S \wedge \exists v'(v' \not\prec_t v \wedge v' \prec_t v_r \wedge \alpha \in \|C_{path}(v')\|)\}$. Further according to our definitions $C_{path}(v_r)$ is equivalent to $C_{path}(v) \wedge \neg c$.

To show that

$$\|v^r\| \subseteq \|C_{path}(v) \wedge \neg c\| - A_S^{v_r}$$

fix a symmetry $s \in S$, then we have to show for every $v' \prec_t v$ that

$$\forall \alpha \in \|C_{path}(v')\| : s(\alpha) \notin \|v^r\|$$

The first case is $v' \prec_t v$. Let $\alpha \in ||C_{path}(v')||$. Then $s(\alpha) \notin ||v||$ by induction hypotheses. Since $||v^r|| \subseteq ||v||$, this immediately implies $s(\alpha) \notin ||v^r||$.

The second case is $(v' \not\prec_t v)$ and $v' \prec_t v_r$. Let $\alpha \in ||v^l||$ and assume $s(\alpha) \in ||v^r||$. From $\alpha \in ||v^l||$ we have $\alpha \models C_p \wedge c$ and $s(\alpha) \models s_{\text{con}}(C_p \wedge c)$. That's a contradiction, because from definition of S-excluding search tree $s(\alpha) \models s_{\text{con}}(C_p) \rightarrow \neg s_{\text{con}}(c)$, since $(s_{\text{con}}(C_p) \rightarrow \neg s_{\text{con}}(c)) \in E^v(\neg c)$.

It remains to be shown that

$$||v^r|| \supseteq ||C_{path}(v_r)|| - A_S^{v_r}$$

Let $\alpha \notin ||v^r||$. We have to show

$$\alpha \notin ||C_{path}(v_r)|| - A_S^{v_r} \tag{4}$$

We have the following cases:

1. $\alpha \notin ||C_{path}(v_r)||$. Then (4) follows immediately.
2. $\alpha \in ||C_{path}(v_r)||$ and $\alpha \notin ||v||$. We have to show $\alpha \in A_S^{v_r}$.

   Now $\alpha \in ||C_{path}(v_r)||$ implies $\alpha \in ||C_{path}(v)||$ and henceforth $\alpha \in A_S^v \subseteq A_S^{v_r}$ by induction hypotheses.
3. $\alpha \in ||C_{path}(v_r)||$ and $\alpha \in ||v||$. We will show that

$$\begin{aligned} \alpha \in A_S^{v_r} - A_S^v &= \{s(\alpha) \mid s \in S \wedge \exists v' : (v' \prec_t v_r \wedge v' \not\prec_t v \wedge \alpha \in ||C_{path}(v')||\} \\ &= \{s(\alpha) \mid s \in S \wedge \alpha \in ||C_{path}(v_l)||\} \end{aligned}$$

   Since $\alpha \models C_{\text{store}} \wedge \neg c$, but $\alpha \not\models C_{\text{store}}^r$, there is at least one $s \in S$ with

$$\alpha \not\models s_{\text{con}}(C_p) \rightarrow \neg s_{\text{con}}(c)$$

   by the definition of an S-excluding tree. Fix one of these symmetries s. Then $\alpha \models s_{\text{con}}(C_p) \wedge s_{\text{con}}(c)$. Since $C_{\text{store}} \models s_{\text{con}}(C_p) \rightarrow s_{\text{con}}(C_n)$ by Proposition 4.5, we get

$$\alpha \models s_{\text{con}}(C_p) \wedge s_{\text{con}}(c) \wedge s_{\text{con}}(C_n).$$

   Hence, $s^{-1}(\alpha)$, which exists by definition of symmetries, satisfies $s^{-1}(\alpha) \models C_{path}(v) \wedge c$, i.e.

$$s^{-1}(\alpha) \in ||C_{path}(v_l)||.$$

□

Theorem 4.7 *Let S be a symmetry set. Every S-excluding search tree t for C_{Pr} is S-reduced.*

Proof. Fix a symmetry $s \in S$, let v be a leaf of t with $||v|| = \{\alpha\}$. We have to show for every node v' of t with $v' \prec_t v$ that

$$||v'|| = \{\alpha'\} \rightarrow s(\alpha') \neq \alpha.$$

Assume v' to be a node of t with $v' \prec_t v$ and $||v'|| = \{\alpha'\}$ and $s(\alpha') = \alpha$. It follows

$$\alpha \in \{s(\alpha) \mid s \in S \wedge \exists v' \prec_t v : \alpha \in ||C_{path}(v')||\},$$

and from this the contradiction $\alpha \notin ||v||$ by proposition 4.6. □

By now we understand that an S-excluding search tree may exclude more symmetries than actually declared by S directly. To face this we have to investigate the completeness property of S-excluding search trees. We already defined C_{Pr}-complete w.r.t S for symmetry groups. But for symmetry sets we have to generalize this completeness property.

Definition 4.8 *Let S be a symmetry set. A search tree t is C_{Pr}-*complete w.r.t *S, iff t is C_{Pr}-complete w.r.t S', where S' is the closure under composition of S.*

Theorem 4.9 *Let S be a symmetry set. Every S-excluding search tree t for C_{Pr} is C_{Pr}-complete w.r.t S.*

Proof. We have to show that t is C_{Pr}-complete w.r.t. to S', where S' is the closure under composition of S, i.e. for every $\alpha \in ||C_{\mathrm{Pr}}||$ there is a leaf v with

$$||v|| = \{\alpha\} \;\vee\; \exists s \in S' : ||v|| = \{s(\alpha)\}. \tag{5}$$

Hence fix an $\alpha \in ||C_{\mathrm{Pr}}||$. Then there exists a leaf v' in t with

$$\alpha \in ||C_{path}(v')||$$

from definition 4.4. There are two cases.

1. $\alpha \in ||v'||$. This yields immediately $||v'|| = \{\alpha\}$ and (5) is proven.
2. $\alpha \notin ||v'||$. By proposition 4.6 follows

   $$\alpha \in A_S^{v'},$$

   i.e., there is a $s_1 \in S$ and a $v'' \prec_t v'$ such that $\alpha = s_1(\alpha')$, where

   $$\alpha' \in ||C_{path}(v'')||.$$

   By induction we get a sequence of symmetries $s_1, \ldots, s_n \in S$, where

   $$\alpha = s_1 \circ \cdots \circ s_n(\alpha^{(n)})$$

   and

   $$\{\alpha^{(n)}\} = ||v^{(n+1)}||$$

   for a leaf $v^{(n+1)}$ of t.

   Since S' is the closure of S, claim (5) follows.

□

5 An Example with Permutation Symmetries

A interesting non geometric application of the introduced symmetry exclusion mechanism is offered by the photo problem. Here the problem is to align a set of persons in a row, such to get as much persons as possible satisfied with their neighbours. More formally the preferences of n given persons are given by a set P of two element sets $\{i, j\}$. A photo is just a permutation $X = \{x_1, ..., x_n\}$ of the n persons and we search a photo X that optimizes the number of satisfied preferences, i.e. we maximize

$$v(X) = |p(X)|,$$

where

$$p(X) = \{\{x_i, x_{i+1}\} \in P \mid 1 \leq i < n\}.$$

To state symmetry in this problem, we define to photos X and Y to be symmetric, if p(X) = p(Y), i.e. X and Y satisfy exactly the same preferences. Of course this implies $v(X) = v(Y)$, i.e., the values of both photos are equal, too.

It shows, that there are exponentially many symmetries according to this definition in our symmetry set S, namely all permutations of n numbers.

Hence, excluding all symmetric permutations seems unfeasible. Now one could think of reducing the number of symmetries to exclude at least some of them. A good choice is to handle all reversions, since first, there are only quadratically many of them and second, reversions form a generating set for the whole symmetry group.

For an implementation with reversion exclusion we describe a symmetry $s \in S$ as

`rev(X,Y)`,

i.e., a reversion between positions `X` and `Y`. Since a reversion is excluded, if it destroys any satisfied preferences, we have to check the condition

`Satisfied.(X-1).X=:0` $\wedge$ `Satisfied.Y.(Y+1)=:0`

for each symmetry to detect if it is already excluded. Here the constraint `Satisfied.X.(X+1)=:1` expresses, that there is a satisfied preference between position `X` and `X+1` on the current photo. Technically the above condition is added to $s_{\mathrm{con}}(C_p)$.

What a symmetric constraint looks like depends on the constraints c that we use to enumerate, i.e. the constraints placed as labels at the edges of our search tree. Here we want to distribute positions to our persons, hence a constraint c has the form

`Person=:Position`,

i.e., the person `Person` gets the position `Position`.

Its symmetric constraint $s_{\mathrm{con}}(c)$, where s is a symmetry s=`rev(X,Y)`, is either

`Person=:X+Y-Position`,

if `Position` is between `X` and `Y`, or it is `Person=:Position`, i.e., c, otherwise.

According to the theorems above, excluding all reversions will not necessarily exclude only reversions, but symmetries from the closure under composition, i.e. all permutations, may be excluded as well.

Now the surprise is, that in fact, i.e., at least we obeserved this for all the examples we tested, so there is a strong indication, all permutations are excluded, even when one restricts to put only reversions into S.

Unfortunately as of yet, there is no prove of this observation. This phenomenon, i.e., a symmetry group can be exluded by its generating set, does not occur generally, since contradicting examples can be constructed. But for the moment it remains an open question which special properties are responsible for it to occur. Currently we consider it being a property of the search strategy.

References

[1] Alfonso San Miguel Aguirre. How to use symmetries in boolean constraint solving. In Frdric Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming, Selected Research*, chapter 16, pages 287–306. The MIT Press, 1993.

[2] Rolf Backofen. Constraint techniques for solving the protein structure prediction problem. In *Proceedings of* 4^{th} *International Conference on Principle and Practive of Constraint Programming (CP'98)*, 1998.

[3] Belaid Benhamou. Study of symmetry in constraint satisfaction problems. In Alan Borning, editor, *Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94,*, volume 874 of *Lecture Notes in Computer Science*. Springer, 1994.

[4] R. Jagadeesan, V. Saraswat, and V. Shanbhogu. Angelic non-determinism in concurrent constraint programming. Technical report, Xerox PARC, 1991.

[5] Christian Schulte, Gert Smolka, and Jörg Würtz. Encapsulated search and constraint programming in Oz. In A.H. Borning, editor, *Second Workshop on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, vol. 874, pages 134–150, Orcas Island, Washington, USA, May 1994. Springer-Verlag.

[6] Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin, 1995.