Detecting and Breaking Symmetries by Reasoning on Problem Specifications

Toni Mancini and Marco Cadoli

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
{tmancini, cadoli}@dis.uniroma1.it

Abstract. In this paper we address the problem of detecting and breaking symmetries in combinatorial problems, following the approach of imposing additional symmetry-breaking constraints. Differently from other works in the literature, we attack the problem at the *specification* level. In fact, many symmetries depend on the structure of the problem, and not on the particular input instance. Hence, they can be easily detected by reasoning on the specification, and appropriate symmetry-breaking formulae generated. We give formal definitions of symmetries and symmetry-breaking formulae on specifications written in existential second-order logic, clarifying the new definitions on some specifications: Graph 3-coloring, Social golfer, and Protein folding problems. Finally, we show experimentally that, applying this technique, even if in a naive way, to specifications written in state-of-the-art languages, e.g., OPL, may greatly improve search efficiency.

1 Introduction

The presence of symmetries in constraint satisfaction problems (CSPs) has been widely recognized to be one of the major obstacles for their efficient resolution. Much work has been already done, and a wide literature is nowadays available on how symmetries can be exploited, with the aim of greatly reducing the size of the search space. There are four main approaches followed by the research community to deal with symmetries:

1. Imposing additional constraints on the problem model, which are satisfied only for one of the symmetrical points in the search space, cf., e.g., [21, 7, 9];
2. Introducing additional constraints during the search process, to avoid the traversal of symmetrical points, cf., e.g., [3, 10];
3. Defining a search strategy able to break symmetries as soon as possible (e.g., by first selecting variables involved in the greatest number of local symmetries), cf., e.g., [18];
4. Isolating subclasses of CSPs for which particular search strategies can be used in order to efficiently break their symmetries (cf., e.g., tractability of symmetry breaking for CSPs with various form of interchangeability [25]).

However, all these approaches make the assumption that symmetries of the constraint problem at hand are known. Hence, the problem of the *automatic detection* of symmetries arises. Currently, symmetry detection is either performed

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 165–181, 2005.

by hand (it is the modeller that states them, by analyzing the problem), or recognized by reducing the CSP obtained *after* instantiation to an instance of the graph automorphism problem (for which there are no polynomial time algorithms, even if there is evidence that it is not NP-complete [13]).

On the other hand, many of the available systems and languages for the solution of constraint problems (e.g., AMPL [11], OPL [24], XPRESS^{MP}[1], DLV [15], SMODELS [19], and NP-SPEC [6]) clearly separate the *specification* of a problem from its *instances*. Furthermore, symmetries often arise from the problem structure, and not from the particular instance considered. Hence, they often clearly emerge at the compact, symbolic level of the specification. Nonetheless, many of the existing approaches to automatic symmetry detection (cf., e.g., the package *Nauty* [16]) try to infer all symmetries of a constraint problem after instantiation, where many structural aspects have been irremediably hidden.

In our opinion, reasoning at the logical level of the problem specification may be much effective in order to detect those structural properties that are suitable for optimization and reformulation, as many symmetries are: problem specifications are usually much more compact, readable, and high-level modelled, hence the recognition of, e.g., structural symmetries naturally fits at this stage. Moreover, convenient symmetry-breaking formulae (cf. approach 1 in the list above) can be added to the specification itself in order to exploit them. Finally, since specifications are logical formulae, computer tools can be used to automatically or semiautomatically detect and break symmetries [5].

Such reasoning tasks have, in principle, at least two applications: *(i)* Discover new properties of a specification, and *(ii)* Validate a specification confirming the existence of some properties. In this paper, we mainly focus on the latter, giving a formal characterization of symmetries and symmetry-breaking formulae for a specification. This is a mandatory first step also to solve *(i)* (an heuristic, and incomplete, approach for detecting some symmetries on specifications is discussed in [26]).

Of course, detecting and breaking symmetries at the specification level does not rule out the possibility to compositionally use symmetry-breaking techniques at the instance level (e.g., [7, 9]), in order to deal with additional symmetries that arise from the problem instance. As an example, since some systems generate a SAT instance, e.g., [6], or an instance of integer linear programming, e.g., [24], it is possible to do symmetry breaking on such instances, using existing techniques.

2 Existential Second-Order Logic as a Modelling Language

When dealing with problem specifications, the first choice to be made is that of the modelling language to be used. Current systems and languages for declarative constraint modelling, as those listed in Section 1, have their own syntax for describing problem specifications: AMPL, OPL, and XPRESS^{MP} allow the

[1] cf. `http://www.dashoptimization.com`.

representation of constraints by using algebraic expressions, while others, e.g., DLV, SMODELS, and NP-SPEC are rule-based languages, more specifically extensions of datalog. Anyway, from an abstract point of view, all such languages are extensions of *existential second-order logic* (ESO) over finite databases, where the existential second-order quantifiers and the first-order formula represent, respectively, the *guess* and *check* phases of the constraint modelling paradigm. In particular, even if all such languages have a richer syntax and more complex constructs, in all of them it is possible to embed ESO queries, and the other way around is also possible, as long as only finite domains are considered. Hence, as we show in the remainder of this section, *ESO can be considered as the formal logical basis for virtually all available languages for constraint modelling*, being able to represent all search problems in the complexity class NP [20]. Moreover, since checking and breaking symmetries on ESO specifications reduces to check semantic properties of logical formulae, it is possible to use known results and techniques in order to automate such tasks.

Formally, an ESO specification describing a search problem π is a formula $\psi_\pi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$, where $\boldsymbol{\mathcal{R}} = \{R_1, \ldots, R_k\}$ is the input relational schema (i.e., a fixed set of relations of given arities denoting the schema for all input instances for π), and ϕ is a closed first-order formula on the relational vocabulary $\boldsymbol{\mathcal{S}} \cup \boldsymbol{\mathcal{R}} \cup \{=\}$ ("=" is always interpreted as identity). An instance $\boldsymbol{\mathcal{I}}$ of the problem is given, as it happens in current systems, as a relational database over the schema $\boldsymbol{\mathcal{R}}$, i.e., as an extension for all relations in $\boldsymbol{\mathcal{R}}$. Predicates (of given arities) in the set $\boldsymbol{\mathcal{S}} = \{S_1, \ldots, S_n\}$ are called *guessed*, and their possible extensions (with tuples on the domain given by constants occurring in $\boldsymbol{\mathcal{I}}$ plus those occurring in ϕ, i.e., the so called Herbrand universe) encode points in the search space for problem π on instance $\boldsymbol{\mathcal{I}}$. Formula ψ_π correctly encodes problem π if, for every input instance $\boldsymbol{\mathcal{I}}$, a bijective mapping exists between solutions to π and extensions of predicates in $\boldsymbol{\mathcal{S}}$ which verify $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{I}})$. More formally, the following must hold:

$$\text{For each instance } \boldsymbol{\mathcal{I}}\text{:} \quad \boldsymbol{\Sigma} \text{ is a solution to } \pi(\boldsymbol{\mathcal{I}}) \iff \{\boldsymbol{\Sigma}, \boldsymbol{\mathcal{I}}\} \models \phi.$$

It is worthwhile to note that, when a specification is instantiated, a constraint satisfaction problem (CSP) is obtained.

In order to facilitate the writing of specifications, several built-in constructs are provided by current languages, in particular those for typed relations, functions (cf., e.g., arrays), bounded integers and arithmetics over them. Hence, to ease expressions, and to make specifications more compact and closer to their counterparts in state-of-the-art languages, in this paper we consider an enriched ESO. In particular, we assume that:

1. Guessed predicates may be typed: we write $\exists S \in type_S^1 \times \cdots \times type_S^k$, where each $type_S^i$ is a monadic relation in $\boldsymbol{\mathcal{R}}$ that represents the domain of the i-th argument of S. (For simplicity of notation, given a relation S of arity k, we denote with $type(S)$ the domain of tuples that belong to S, i.e., the set $type_S^1 \times \cdots \times type_S^k$.)
2. Guessed predicates that encode functions can be natively expressed in the language (we write $\exists S \in type_S^1 \times \cdots \times type_S^j \rightarrow type_S^{j+1} \times \cdots \times type_S^k$ for some $j \in [1, k-1]$. Total functions will be denoted by "(total)").

3. Bounded integers and arithmetics over them are available.

We note that such additions do not change the expressive power of the language. Types and (total) functions can be simulated in ESO by means of monadic predicates in $\boldsymbol{\mathcal{R}}$ and first-order constraints, respectively. The same holds for bounded integers and arithmetics (that can be pre-interpreted).

Formally, we denote the set of monadic relations that encode types as $\boldsymbol{\mathcal{T}}$ (with $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{R}}$). Hence, a specification in the enriched language is of the kind:

$$\exists S_1 \in type(S_1), \ldots \exists S_n \in type(S_n) \quad \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{R}}) \tag{1}$$

where $type(S_i) = type^1_{S_i} \times \cdots \times type^{ar(S_i)}_{S_i}$, with all $type^j_{S_i} \in \boldsymbol{\mathcal{T}}$.

Since $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{R}}$, we normally omit $\boldsymbol{\mathcal{T}}$ as argument of ϕ, even if, in some cases, in order to emphasize the occurrence of types relations in some formulae, we state it explicitly.

Example 1 (Graph k-coloring). Given an undirected graph and a set of k colors as input, this problem amounts to decide whether it is possible to give each of its nodes one out of the colors, in such a way that adjacent nodes (not including self-loops) are never colored the same way. The problem is well-known to be NP-complete for $k \geq 3$, and can be specified in ESO by, e.g., the following formula over relations in $\boldsymbol{\mathcal{R}} = \{node(\cdot), edge(\cdot,\cdot), color(\cdot)\}$, listing the graph nodes, edges and the available colors, respectively. The set of types $\boldsymbol{\mathcal{T}}$ is given by $\{node, color\}$. In particular, relation $color(\cdot)$ will have exactly k tuples. We also assume that $node(\cdot)$ and $color(\cdot)$ have no tuples in common.

$$\exists Col \ \in \ node \rightarrow color \text{ (total)} \tag{2}$$

$$\forall X, Y, C, C' \quad edge(X,Y) \wedge X \neq Y \wedge Col(X,C) \wedge Col(Y,C') \ \rightarrow \ C \neq C'. \tag{3}$$

Part (2) of the above specification defines *Col* as a total function assigning a color to each node, while (3) is the good coloring constraint. It is worth noting that the specification above is very close to that written in available languages, e.g., the following one in OPL (initializations are omitted):

```
range node 1..n_nodes;  range color 1..n_colors;
var color Col[node];
solve { forall (e in edges: e.start<>e.end) Col[e.start]<>Col[e.end]; };};
```

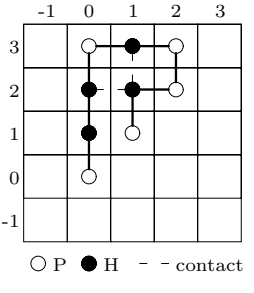
Another assumption that we make in this paper is that the set of guessed predicates $\boldsymbol{\mathcal{S}}$ is partitioned in two parts: *output* and *auxiliary* guessed predicates, denoted, respectively, as $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{A}}$ (with $\boldsymbol{\mathcal{A}}$ possibly empty). Output guessed predicates conceptually denote the search space, while auxiliary predicates are used internally to maintain and/or compute additional information needed to express and evaluate the constraints. This is a very common necessity in declarative languages, as forthcoming Example 2 shows.

When such a partition is made, a solution is completely characterized by the extensions of predicates in $\boldsymbol{\mathcal{O}}$ and *not* by those of predicates in $\boldsymbol{\mathcal{A}}$. Hence, the general form of a problem specification in ESO is as follows:

$$\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \quad \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \tag{4}$$

where predicates in $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{A}}$ may have an associated type, that can (in general) be represented with a first-order formula over $\boldsymbol{\mathcal{T}}$.

Example 2 (HP 2D-Protein folding [14]). This specification models a simplified version of one of the most important problems in computational biology. It consists in finding the spatial conformation of a protein (i.e., a sequence of amino-acids) with minimal energy. The simplifications are twofold: the amino-acids alphabet is reduced to just H (*hydrophobic*) and P (*polar*), and the protein is forced to fold in a 2D discrete space. However, the simplified problem is known to be NP-complete [8]. Given the sequence of amino-acids of the protein, i.e., a string over {H,P} of length n, the problem aims to find a connected shape for it on a 2D grid (with coordinates in $[-(n-1), (n-1)]$, starting at $(0,0)$), non-crossing, and such that the number of "contacts", i.e., the number of non-sequential pairs of Hs for which the Euclidean distance of the positions is 1 is maximized (the overall energy is the opposite of the number of contacts). The figure below shows a possible conformation of the protein "PHHPHPPHP", with overall energy -2.



Different alternatives for the search space obviously exist: as an example, we can guess the position on the grid of each amino-acid, and then force the shape to be connected, non-crossing, and with minimal energy. However, a preferred approach that reduces the size of the search space (4^n points versus $(2n)^{2n}$) is to guess the shape of the protein as a connected path starting at $(0,0)$, by guessing, for each index $i \in [1, n-1]$, the direction that the $(i+1)$-th amino-acid assumes wrt the i-th one (directions can only be North, South, East, West).

The extension of *Move* for the shape in the figure is: $\{\langle 1, N\rangle, \langle 2, N\rangle, \langle 3, N\rangle, \langle 4, E\rangle, \ldots\}$. However, choosing the latter model is not completely satisfactory: to express the non-crossing constraint, and to compute the number of contacts, absolute coordinates of each amino-acid must be computed and maintained. An ESO specification for this problem, where, for simplicity, we assume to deal with its decisional version, and to have (pre-interpreted) bounded integers and arithmetics in the range $[-(n-1), n-1]$, is as follows ($\boldsymbol{\mathcal{R}} = \{index(\cdot), elem(\cdot,\cdot)\}$, with $elem(i,a)$ stating that the i-th element of the protein is $a \in \{H, P\}$):

$$\exists Move \ \in \ index \rightarrow \{N, S, E, W\} \text{ (total)}, \tag{5}$$

$$\exists X, Y \ \in \ index \rightarrow [-n+1, n-1] \text{ (total)} \tag{6}$$

$$X(0,0) \ \wedge \ Y(0,0) \ \wedge \tag{7}$$

$$\begin{aligned} &\forall I, I' \ index(I) \wedge index(I') \wedge I' = I - 1 \ \rightarrow \\ &\quad \forall D, X, Y, X', Y' \ Move(I', D) \wedge X(I, X) \wedge X(I', X') \wedge Y(I, Y) \wedge Y(I', Y') \rightarrow \\ &\qquad D = N \ \rightarrow X = X' \wedge Y = Y' + 1 \wedge \\ &\qquad D = S \ \rightarrow X = X' \wedge Y = Y' - 1 \wedge \\ &\qquad D = E \ \rightarrow X = X' + 1 \wedge Y = Y' \wedge \\ &\qquad D = W \rightarrow X = X' - 1 \wedge Y = Y' \wedge \end{aligned} \tag{8}$$

$$\begin{aligned} &\forall I, I', X, X', Y, Y' \\ &\quad I \neq I' \wedge X(X, I) \wedge X(X', I') \wedge Y(Y, I) \wedge Y(Y', I') \ \rightarrow \ X \neq X' \vee Y \neq Y' \wedge \end{aligned} \tag{9}$$

$$\left| \left\{ \begin{array}{c} \langle I, I' \rangle \mid index(I) \wedge index(I') \wedge (I+1 < I') \wedge elem(I, H) \wedge elem(I', H) \wedge \\ \forall X, X', Y, Y' \ X(X, I) \wedge X(X', I') \ \wedge \\ [3pt] Y(Y, I) \wedge Y(Y', I') \ \wedge \ |X - X'| + |Y - Y'| = 1 \end{array} \right\} \right| \geq k \tag{10}$$

Constraints (5) and (6) declare guessed predicates *Move*, X and Y as total functions assigning, respectively, a value in $\{N, S, E, W\}$, and a value in $[-(n-1), n-1]$ to each amino-acid.[2] Furthermore, (7) forces the first amino-acid to be placed in $(0, 0)$, while (8) defines the absolute position of each amino-acid starting from that of the previous one and the move. Finally, (9) is the non-crossing constraint, and (10) forces the number of contacts to be at least k (integer k is assumed to be fixed). The above specification is very similar to that given in available languages, e.g., OPL (cf. [4]).

From the problem description, *Move* is the output guessed predicate, while X and Y are auxiliary: a solution is completely characterized by the sole extension of *Move*. However, it is a matter of choice and responsibility of the modeler to state which guessed predicates are output and which others are auxiliary, and, of course, it is always possible to consider all guessed predicates as output ones (hence, $\boldsymbol{\mathcal{A}}$ can always be empty). Indeed, in the following we show that this "conceptual" partition plays an important role in detecting structural properties, e.g., symmetries, that may be exploited to improve efficiency. We also observe that, in this example, X and Y are functionally dependent on *Move* (cf. [4]).

3 Symmetries on Problem Specifications

In this section we define the concepts of *transformation* and *symmetry* on a specification, and investigate interesting specializations of them. In Section 2, we presented some syntactic sugar that can be added to ESO in order to have more compact and readable specifications. However, we also noticed that such constructs can always be regarded as additional constraints. Hence, for what concerns the reasoning tasks that we describe from this section on, we consider the basic ESO framework. Hence, all such additional constraint will be considered as integral part of the ϕ part of a specification having the general form (4).

Definition 1 (Transformation). *Given a specification* $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$*, a* transformation *for* $\boldsymbol{\mathcal{O}}$ *and* $\boldsymbol{\mathcal{A}}$ *is a family of functions, one for each possible finite Herbrand domain* $\mathcal{H}$*, of the kind* $\tau_{\mathcal{H}}$*:*$\{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\} \rightarrow \{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$*, where* $\{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$ *is the set of all possible extensions of predicates in* $\boldsymbol{\mathcal{O}}$ *and* $\boldsymbol{\mathcal{A}}$ *with elements in* $\mathcal{H}$*.*

Intuitively, a transformation is a mapping from and to all points in the search space defined by all the guessed predicates in the specification, for any $\mathcal{H}$. For the sake of simplicity, and with a little abuse of notation, in what follows we denote a transformation as a single function τ: $\{ext(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\} \rightarrow \{ext(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$, obtained by collapsing all the $\tau_{\mathcal{H}}$, which is defined on all finite Herbrand domains $\mathcal{H}$.

By focusing only on the set $\boldsymbol{\mathcal{O}}$, the following definition holds:

[2] Actually, *Move* should be not defined for the last amino-acid. However, the proposed simpler specification remains correct, with the last move having no meaning.

Definition 2 (Symmetry). *Given a problem specification* $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ *as above, a* symmetry *is an* invertible *transformation for* $\boldsymbol{\mathcal{O}}$*, i.e., an invertible function* $\tau : \{ext(\boldsymbol{\mathcal{O}})\} \rightarrow \{ext(\boldsymbol{\mathcal{O}})\}$ *such that, for every input instance* $\boldsymbol{\mathcal{I}}$*, and every extension* $\boldsymbol{\Omega}$ *for relations in* $\boldsymbol{\mathcal{O}}$*, the following holds:*

$$\boldsymbol{\Omega}, \boldsymbol{\mathcal{I}} \models \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \iff \tau(\boldsymbol{\Omega}), \boldsymbol{\mathcal{I}} \models \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}). \tag{11}$$

The distinction between output and auxiliary guessed predicates here becomes more clear: a symmetry is a transformation of the sole output predicates such that, if an extension of $\boldsymbol{\mathcal{O}}$ may lead to a solution (with appropriate extensions for auxiliary predicates $\boldsymbol{\mathcal{A}}$), then its transformation must also lead to a solution (even if the corresponding extensions for predicates in $\boldsymbol{\mathcal{A}}$ change) and vice versa. As an example, in the Protein folding problem, given a solution, i.e., a move in $\{N, S, E, W\}$ for each element of the sequence such that all constraints are satisfied, we can uniformly change N with S, and/or E with W and obtain another solution, even if the corresponding extensions for X and Y change.

Definition 1 is about transformations in general, but does not limit in any way the kind of functions τ. By imposing some restrictions on τ, interesting specializations arise. In this paper, we consider functions τ that focus on a single output guessed predicate, being the identity function on the others (we call them *single-predicate transformations*). They are of special interest, because of the usual structure of constraint problems, in which transformations we are interested in (i.e., candidate symmetries) often are internal to a guessed predicate.

Definition 3 (Single-predicate transformation). *A transformation is* single-predicate *if there exist* $O \in \boldsymbol{\mathcal{O}}$ *and a function* $\tau_O : \{ext(O)\} \rightarrow \{ext(O)\}$ *such that, for all extensions* $\Omega_1, \ldots, \Omega, \ldots \Omega_n$ *for* $O_1, \ldots, O, \ldots, O_n$ *(for any finite* $\mathcal{H}$*), we have that* $\tau(\Omega_1, \ldots, \Omega, \ldots \Omega_n) = \langle \Omega_1, \ldots, \tau_O(\Omega), \ldots, \Omega_n \rangle$.

A single-predicate transformation over $O \in \boldsymbol{\mathcal{O}}$ is completely characterized by giving τ_O. Further specializations of single-predicate transformations are *column* (definition omitted) and *uniform column transformations*.

Definition 4 (Uniform column transformation (UCT)). *A single-predicate transformation* τ_O *is a UCT if there exists a partition of the indexes of arguments of* O *in two (disjoint) sets,* D *and* C*, such that, for each extension* Ω *of* O*, we have that* $\tau_O(\Omega) = \Omega'$*, where:*

$$\forall \boldsymbol{\delta} \quad \boldsymbol{\delta} \in \Omega \quad \leftrightarrow \quad \langle \boldsymbol{\delta}[D], \sigma(\boldsymbol{\delta}[C]) \rangle \in \Omega'$$

where $\sigma : type(\underset{C}{\pi}(O)) \rightarrow type(\underset{C}{\pi}(O))$ *is a total invertible function on (i.e., a* permutation *of) the domain values of arguments of* O *in set* C.

A UCT that is a symmetry is called *uniform column symmetry* (UCS). Intuitively, UCTs and UCSs change only the C components of tuples in an extension of O, leaving the others (i.e., D) unchanged. Hence, they are completely described by a permutation σ from and to the type of the C columns of O. It is worth noting that σ is *uniform*, i.e., its behavior on a tuple $\boldsymbol{\delta} \in O$ depends only on $\boldsymbol{\delta}[C]$, and not on $\boldsymbol{\delta}[D]$. A (non-uniform) column transformation/symmetry, instead, is described by a function which behavior on tuple $\boldsymbol{\delta}$ depends also on $\boldsymbol{\delta}[D]$.

Example 3 (Graph k-coloring (Example 1 continued)). We have that $\boldsymbol{\mathcal{O}} = \{Col\}$, and $\boldsymbol{\mathcal{A}} = \emptyset$. By focusing on Col, with $D = \{1\}$ and $C = \{2\}$, all permutations $\sigma : color \rightarrow color$ (where $color$ is $type(\underset{C}{\pi}(Col))$) are UCSs.

These symmetries are uniform because their σs map a given color (e.g., red) always to the same color, independently on the nodes (i.e., values in column 1).

If, after instantiation, we define the corresponding CSP with one variable for each node, with domain $[1, k]$, symmetries defined above become *uniform value symmetries* (in the sense of [17]).

Example 4 (HP 2D-Protein folding (Example 2 continued)). Let us consider the UCTs that focus on $Move$, with $D = \{1\}$ and $C = \{2\}$, i.e., permutations σ of $\{N, S, E, W\}$. As an example, the following ones are UCSs:

$\sigma(N) = N,\ \sigma(S) = S,\ \sigma(E) = W,\ \sigma(W) = E$ (flip horizontally)
$\sigma(N) = S,\ \sigma(S) = N,\ \sigma(E) = E,\ \sigma(W) = W$ (flip vertically)
$\sigma(N) = S,\ \sigma(S) = N,\ \sigma(E) = W,\ \sigma(W) = E$ (flip horizontally & vertically)
$\sigma(N) = E,\ \sigma(S) = W,\ \sigma(E) = S,\ \sigma(W) = N$ (rotation 90° clockwise)

while others are not, e.g., σ such that:

$$\sigma(N) = N, \quad \sigma(S) = E, \quad \sigma(E) = W, \quad \sigma(W) = S.$$

It is worth noting that, if we consider also X and Y as output guessed predicates, the above transformations are no longer UCSs, moving to the more general class of *multiple-predicate symmetries* (definition omitted). In fact, when permuting directions in $Move$, extensions for X and Y must change accordingly.

4 Symmetry Checking

In the previous section, we considered transformations and symmetries as functions from and to extensions of predicates in $\boldsymbol{\mathcal{O}}$. Nonetheless, in order to practically deal with transformations and symmetries, we are interested into finite representations of such functions. To this end, in what follows we assume that τ is finitely representable, e.g., in first-order logic, and, with a little abuse of notation, we will denote with $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ a logical representation of it.[3] Such a representation will contain also occurrences of types in $\boldsymbol{\mathcal{T}}$. However, for simplicity, we do not explicitly write such types as arguments of τ.

Theorem 1. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, and τ an invertible transformation for $\boldsymbol{\mathcal{O}}$. τ is a symmetry for ψ if and only if the following formula is valid:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \quad \rightarrow \quad [\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \ \leftrightarrow\ \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})]\,. \tag{12}$$

It is worth noting that the above formula is second-order, even if τ is first-order. This is because of the presence of auxiliary guessed predicates $\boldsymbol{\mathcal{A}}$, which

[3] Given extensions $\boldsymbol{\Omega}$ and $\boldsymbol{\Omega}_\tau$ for $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{O}}_\tau$ respectively, $\tau(\boldsymbol{\Omega}, \boldsymbol{\Omega}_\tau)$ is `true` iff $\boldsymbol{\Omega}_\tau$ is the output of function τ when applied to $\boldsymbol{\Omega}$.

extensions may not coincide when applying the transformation (cf., e.g., Example 4). However, in the important case of $\boldsymbol{\mathcal{A}} = \emptyset$, the above formula reduces to a first-order one.

The above theorem naturally specializes in case of single-predicate symmetries and UCSs. In the latter case, the following holds:

Theorem 2. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\, \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, and let τ_O be a UCT on $O \in \boldsymbol{\mathcal{O}}$ with σ the relative permutation of the domain values of arguments of O in set C. τ_O is a symmetry for ψ if and only if the following formula is valid:*

$$\tau(O, O_\tau) \;\rightarrow\; \exists \boldsymbol{\mathcal{A}}\; \phi(O_1,\ldots,O,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \leftrightarrow \exists \boldsymbol{\mathcal{A}}\; \phi(O_1,\ldots,O_\tau,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \qquad (13)$$

with τ being: $\forall \boldsymbol{X_D}, \boldsymbol{X_C}, \boldsymbol{X_C^\sigma} \quad O(\boldsymbol{X_D}, \boldsymbol{X_C}) \wedge \sigma(\boldsymbol{X_C}, \boldsymbol{X_C^\sigma}) \;\leftrightarrow\; O_\tau(\boldsymbol{X_D}, \boldsymbol{X_C^\sigma})$, and σ a finite representation of the permutation over type$\left(\underset{C}{\pi}(O)\right)$.

However, the problem of checking symmetries is undecidable. To show this, we focus on the most restricted case of first-order definable UCTs, when $\boldsymbol{\mathcal{A}} = \emptyset$.

Theorem 3. *Checking whether a first-order definable UCT τ_O is a symmetry is undecidable, even if $\boldsymbol{\mathcal{A}} = \emptyset$.*

Of course, decidable subcases for this problem may exist, and can be possibly derived by decidability results already known in first-order and second-order logic (cf., e.g., [2]). Additionally, decidable heuristic approaches, similar to those already presented in [25, 26] can be used. However, these issues are left for future research.

Often, constraint problems exhibit *many* symmetries. In order to make the relevant checks, the procedure suggested above by Theorems 1 and 2 needs to be invoked for all of them. However, when a set of symmetries can be finitely characterized, Theorems 1 and 2 can be restated with τ being the finite representation of the *whole* set of symmetries. In the particular case of UCSs, Theorem 2 can be restated with σ being the finite representation of the whole set of permutations over type$\left(\underset{C}{\pi}(O)\right)$ that are symmetries. In these cases, $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ holds iff $\boldsymbol{\mathcal{O}}_\tau$ is the result of applying *any* symmetry in the set to $\boldsymbol{\mathcal{O}}$ (hence, it models a relation, and not a single function any more). The same holds for σ in case of UCSs.

Example 5 (Social golfer (`www.csplib.org`, prob. 10)). Given a set of players, a set of groups, and a set of weeks, encoded in relations $\boldsymbol{\mathcal{R}} = \{player(\cdot), group(\cdot), week(\cdot)\}$ respectively, this problem amounts to decide whether there is a way to arrange a scheduling for all weeks in *week*, such that *(i)* For every week, players are divided into equally sized groups; *(ii)* Two different players don't play in the same group more than once. A specification for this problem (assuming $|player|/|group|$, i.e., the group size, integral) is the following ($Play(P, W, G)$ states that player P plays in group G on week W):

$$\exists Play \ \in \ player \times week \rightarrow group \ \ \text{(total)} \tag{14}$$

$$\forall P,P',W,W',G,G' \quad (P \neq P' \wedge W \neq W' \wedge PLAY(P,W,G) \wedge PLAY(P',W,G)) \rightarrow \neg[Play(P,W',G') \wedge Play(P',W',G')] \ \wedge \tag{15}$$

$$\forall G,G',W,W' \ group(G) \wedge group(G') \wedge week(W) \wedge week(W') \ \rightarrow \ |\{P : Play(P,W,G)\}| = |\{P : Play(P,W',G')\}|. \tag{16}$$

Relation *Play* is declared as a total function assigning a group to each player on each week (14); moreover, (15) is the *meet only once* constraint, while (16) forces groups to be equally sized. The last constraint can be written in ESO using standard techniques, essentially by means of an auxiliary guessed predicate *Aux* –hence $\boldsymbol{\mathcal{A}} = \{Aux\} \neq \emptyset$– forced to encode a set of bijective functions, one between tuples of any pair of sets defined in the specification.

The following sets of UCTs that focus on *Play* are all UCSs:

1. With $D = \{1,2\}$, $C = \{3\}$, all permutations $\sigma : group \rightarrow group$ of groups;
2. With $D = \{1,3\}$, $C = \{2\}$, all permutations $\sigma : week \rightarrow week$ of weeks;
3. With $D = \{2,3\}$, $C = \{1\}$, all permutations $\sigma : player \rightarrow player$ of players.

Let us consider the set of UCSs described in point 1. A finite representation for them exists, in the form of $\tau_G(Play, Play_{\tau_G})$, defined as:

$$\forall P,W,G,G^{\sigma} \ Play_{\tau_G}(P,W,G^{\sigma}) \ \leftrightarrow \ Play(P,W,G) \wedge perm(\sigma, group) \wedge \sigma(G,G^{\sigma}),$$

with $perm(\sigma, group)$ being a first-order formula stating that σ is a permutation of domain values in *group*, i.e., $type(\underset{C}{\pi}(O)))$. A formulation for *perm* is as follows:

$$perm(\sigma,R) \ \doteq \ \forall \boldsymbol{X}, \boldsymbol{X^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \ \rightarrow R(\boldsymbol{X}) \ \wedge \ R(\boldsymbol{X^{\sigma}}) \ \wedge \tag{17}$$

$$\forall \boldsymbol{X} \ R(\boldsymbol{X}) \ \rightarrow \ \exists \boldsymbol{X^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \ \wedge \tag{18}$$

$$\forall \boldsymbol{X}, \boldsymbol{X^{\sigma}}, \boldsymbol{X'^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \wedge \sigma(\boldsymbol{X},\boldsymbol{X'^{\sigma}}) \rightarrow \boldsymbol{X^{\sigma}} = \boldsymbol{X'^{\sigma}} \ \wedge \tag{19}$$

$$\forall \boldsymbol{X^{\sigma}} \ R(\boldsymbol{X^{\sigma}}) \ \rightarrow \ \exists \boldsymbol{X} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}). \tag{20}$$

In the important case of a set of UCSs, the following specialized result holds:

Corollary 1. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, $O \in \boldsymbol{\mathcal{O}}$, and D and C a partition of its argument indexes. A set of permutations σ over $type(\underset{C}{\pi}(O))$, finitely characterized by the additional conditions encoded in a formula $\gamma(\sigma, \boldsymbol{\mathcal{T}})$, are all UCSs for ψ iff the following formula (open wrt $\boldsymbol{\mathcal{O}}, O_\tau, \boldsymbol{\mathcal{R}}, \sigma$)[4] is valid:*

$$\tau(O,O_\tau) \ \rightarrow \ \exists \boldsymbol{\mathcal{A}} \ \phi(O_1,\ldots,O,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \leftrightarrow \exists \boldsymbol{\mathcal{A}} \ \phi(O_1,\ldots,O_\tau,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \tag{21}$$

with τ being:

$$perm(\sigma, type(\underset{C}{\pi}(O))) \wedge \gamma(\sigma,\boldsymbol{\mathcal{T}}) \ \wedge \ \forall \boldsymbol{X_D}, \boldsymbol{X_C}, \boldsymbol{X_C^{\sigma}} \quad O_\tau(\boldsymbol{X_D}, \boldsymbol{X_C^{\sigma}}) \ \leftrightarrow \ O(\boldsymbol{X_D}, \boldsymbol{X_C}) \wedge \sigma(\boldsymbol{X_C}, \boldsymbol{X_C^{\sigma}})$$

[4] With σ being a predicate of arity $|type(\underset{C}{\pi}(O))|$.

In general, formula (21) is second-order, because of the presence of "$\exists \boldsymbol{\mathcal{A}}$", and because $\gamma(\sigma, \boldsymbol{\mathcal{T}})$ can be second-order. Of course it reduces to a first-order formula when $\boldsymbol{\mathcal{A}} = \emptyset$ and $\gamma(\sigma, \boldsymbol{\mathcal{T}})$ is also first-order.

All the UCSs described in Examples 3 and 5 can be checked in one step by using Corollary 1, with $\gamma \equiv \mathsf{true}$. As another example, let us consider a variation of Social golfer where the following constraint is added:

$$\forall P, W, G \qquad P = p_1 \wedge \mathit{Play}(P, W, G) \;\rightarrow\; G = g_1 \tag{22}$$

forcing a particular player (denoted by the constant "p_1") to play always in the same group (denoted by constant "g_1"). In the new specification, not all UCTs denoted with *1.* in Example 5 are symmetries any more. In particular, only those permutations of groups σ such that $\sigma(g_1) = g_1$ remain symmetries. The whole set of such permutations is finitely representable as $\gamma(\sigma, \boldsymbol{\mathcal{T}}) \doteq \sigma(g_1, g_1)$, thus, they can be all verified at once by using Corollary 1.

5 Symmetry Breaking

In Section 4, we showed how logically representable sets of "structural" symmetries can be checked by reasoning on the problem specification. Here we show how such knowledge can be used in order to *modify* the specification, in order to exclude from the search space (some of) the symmetrical points. Such modifications can of course be made by working only on the specification, since they will be valid whatever instance we will consider in a later stage.

Actually, several approaches to symmetry breaking have been described in Section 1. In this paper, we focus on the first one (i.e., the addition of symmetry-breaking constraints) but, differently from other works in the literature (e.g., [7, 9]), we attack this problem at the logical level of the specification.

Definition 5 (Symmetry-breaking formula). *Given a specification $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$, and a logical representation $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ of a set of symmetries, a symmetry-breaking formula for them is a closed (except for $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{T}}$) formula $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ –in general in second-order logic– such that the new specification*

$$\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \quad \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \;\wedge\; \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$$

satisfies the following two requirements (we call them Conditions 1 and 2):

1. *The set of transformations τ is not a set of symmetries for the new problem any more: hence, the following formula (negation of (12)), is satisfiable:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \;\wedge\; [\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}}) \;\not\Rightarrow\; \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{T}})]\,.$$

2. *Every model of $\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ (i.e., every solution for any input instance) can be obtained by those of $\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ by applying transformations in τ an arbitrary number of times:*

$$\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \;\models\; \exists \boldsymbol{\mathcal{O}}_\beta\ \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{T}}) \wedge \bigvee_{i \geq 0} \tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$$

where $\tau^0(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$ *is defined as* $\boldsymbol{\mathcal{O}}_\beta \equiv \boldsymbol{\mathcal{O}}$,[5] *and* $\tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$ *(*$i > 0$*) as* $\exists \boldsymbol{\mathcal{O}}' \; \tau(\boldsymbol{\mathcal{O}}', \boldsymbol{\mathcal{O}}) \wedge \tau^{i-1}(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}}')$*, with* $\boldsymbol{\mathcal{O}}'$ *a fresh set of variables.*

Lemma 1 (Alternative formulation for Conditions of Definition 5). *Formula* $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ *is symmetry-breaking for set of symmetries* $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ *iff it satisfies the following two, alternative, conditions:*

1. *The following formula is satisfiable:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \; \wedge \; \exists \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \; \wedge \; [\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}}) \; \not\leftrightarrow \; \beta(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{T}})] \, . \tag{23}$$

2. *It holds that:*

$$\exists \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \quad \models \quad \exists \boldsymbol{\mathcal{O}}_\beta \; \beta(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{T}}) \; \wedge \; \bigvee_{i \geq 0} \tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}}). \tag{24}$$

If $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ respects the above conditions, we are entitled to solve the problem $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ instead of the original one $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$. In fact, Condition 1 states that formula $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ actually breaks τ, since, by Theorem 1, transformations in τ are not all symmetries of the rewritten problem. Furthermore, Condition 2 states that every solution of $\phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ can be obtained by repeatedly applying transformations in τ to some solutions of $\phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$. Hence, all solutions are preserved in the rewritten problem, up to symmetric ones.

It can be observed that Condition 1, even if it behaves well when τ describes a single symmetry, is quite weak when used with a set of symmetries. This is because it is enough, for a formula β, to break just one of the symmetries in τ to satisfy it. A stronger characterization of Condition 1 for the case of τ representing a set of symmetries is currently under investigation.

As for Condition 2, it is worthwhile noting that in formula (24) i ranges over the (infinite) set of positive integers. However, once the (always finite) Herbrand universe $\mathcal{H}$ has been fixed, the number of consecutive applications of τ that lead to different extensions for predicates in $\boldsymbol{\mathcal{O}}$ is always finite (even if this value actually depends on $\mathcal{H}$). Furthermore, when dealing with UCSs on guessed predicate $O \in \boldsymbol{\mathcal{O}}$, i is bound by $n!$, where n is $\left| type(\underset{C}{\pi}(O)) \right|$, and C the set of indexes where τ focuses on, since this is the maximum number of successive applications of τ that can lead to all different permutations. However, in the following we show that in many practical circumstances, either i is bound to a known value because the value for n is known (cf., e.g., Example 2), or many interesting symmetry-breaking formulae satisfy Condition 2 of Definition 5 by design, with a very low i.

We observe that breaking a symmetry is sound, i.e., it preserves at least one solution, as shown by the following theorem:

[5] In general, given two vectors of variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ of the same length n, by $\boldsymbol{X} \equiv \boldsymbol{Y}$ we denote the formula $\bigwedge_{i=1}^{n} (X_i \leftrightarrow Y_i)$.

Theorem 4 (Symmetry-breaking formulae preserve satisfiability). *Let ψ, τ, and β as in Definition 5. For each input instance $\boldsymbol{\mathcal{I}}$, if $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{I}})$ has solutions, then also $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{I}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ has solutions.*

Example 6 (HP 2D-Protein folding (Example 4 continued)). Let us consider the following UCS τ that focuses on the output guessed predicate *Move*, with $D = \{1\}$ and $C = \{2\}$, characterized by the following permutation σ of $\{N, S, E, W\}$:

$$\sigma(N) = N,\ \sigma(S) = S,\ \sigma(E) = W,\ \sigma(W) = E \text{ (flip horizontally)}$$

The following formula $\beta^{E,W}_{\text{least}}(\mathit{Move}, \mathit{index})$ is symmetry-breaking for it:

$$\beta^{E,W}_{\text{least}} \doteq \forall I\ \mathit{index}(I) \wedge \mathit{Move}(I, W) \rightarrow \exists I'\ \mathit{index}(I') \wedge (I' \leq I) \wedge \mathit{Move}(I', E) \quad (25)$$

since it forces the protein shape to move East before moving West. Condition 1, i.e., formula (23) is satisfied by, e.g., the instance $[H, H]$, and the extension $\{\langle 1, E\rangle\}$ for *Move*. As for Condition 2, it holds even by limiting i to only 0 and 1.

A different symmetry-breaking formula for the same symmetry is:

$$\beta^{E,W}_{\leq} \doteq |\{i : \mathit{Move}(i, E)\}| \leq |\{i : \mathit{Move}(i, W)\}|, \quad (26)$$

that forces the protein "head" to move West at least the same number of times it moves East.

Example 7 (Social golfer (Example 5 continued)). Let us consider all UCSs that focus on the output guessed predicate *Play*, with $D = \{1, 2\}$ and $C = \{3\}$, i.e., all permutations of groups. The following formula (where we assume that a total ordering is given on tuples of relations in $\boldsymbol{\mathcal{T}}$, hence also on their Cartesian product) is symmetry-breaking (according to Definition 5) for all of them:

$$\begin{aligned}\beta_{\text{least}}(\mathit{Play}, \mathit{player}, \mathit{week}, \mathit{group}) \doteq\ & \forall G, G'\ \mathit{group}(G) \wedge \mathit{group}(G') \wedge (G \leq G') \rightarrow \\ & \forall P, W, P', W'\, \mathit{least}((P, W), G) \wedge \mathit{least}((P', W'), G') \rightarrow (P, W) \leq_{PW} (P'W')\end{aligned} \quad (27)$$

with $\leq_{PW}$ the total order derived from $\leq$ on players and weeks. It forces the group assignment to be such that, for all G, G' such that $G \leq G'$, the least pairs P, W and P', W' such that $\mathit{Play}(P, W, G)$ and $\mathit{Play}(P', W', G')$ are such that $(P, W) \leq_{PW} (P', W')$. [6] As a consequence, we have that the first player always plays in the first group. We can break other symmetries (e.g., permutations of weeks or players) in a similar way, and get the symmetry-breaking constraints described in [23].

Social golfer is well known also because it is one of the prototypical examples of problems having a 2D *matrix model* (where rows are players, columns are weeks, and entries are groups) exhibiting all row and column symmetries. For these problems, the lex^2 symmetry-breaking constraint, that forces a lexicographic

[6] $\overline{\mathit{least}((P, W), G)}$ can be written in first-order logic as: $\mathit{Play}(P, W, G) \wedge \forall \overline{P}, \overline{W}\ \mathit{Play}(\overline{P}, \overline{W}, G) \rightarrow (P, W) \leq_{PW} (\overline{P}, \overline{W})$.

ordering on both rows and columns of the matrix has been proposed [9]. It is possible to show that the lex^2 symmetry-breaking constraint can be formulated in ESO by, e.g., a formula $\beta_{lex^2}(Play, player, week, group) \doteq \beta^P_{lex} \wedge \beta^W_{lex}$ where $\beta^P_{lex}(Play, player, week, group)$ is given by:

$$\begin{aligned}
\forall P, P' \quad & player(P) \wedge player(P') \wedge P < P' \ \rightarrow \\
& \exists \overline{W} \ week(\overline{W}) \ \wedge \ \forall W \ week(W) \ \wedge W < \overline{W} \ \rightarrow \\
& \quad \forall G, G' \ \big(Play(P, W, G) \wedge Play(P', W, G')\big) \ \rightarrow \ G = G' \ \wedge \\
& \quad \forall G, G' \ \big(Play(P, \overline{W}, G) \wedge Play(P', \overline{W}, G')\big) \ \rightarrow \ G < G' \ \vee \\
& \forall W, G, G' \ \big(Play(P, W, G) \wedge Play(P', W, G')\big) \ \rightarrow \ G = G'
\end{aligned}$$

that forces a lexicographic ordering among the rows of the matrix, and β^W_{lex} by a similar formula, that forces a lexicographic ordering among the columns.

It is worth noting that from the above formulae, it is straightforwardly possible to derive general schemas, that can be used to break symmetries on many different specifications. To this end, we note that formulae of the kind β_{least}, $\beta_{\leq}$, and β_{lex^2} make the right part of (24) a tautology (it is enough to consider, e.g., in the first two cases, $i \in \{0, 1\}$), and hence they are guaranteed to respect Condition 2 of Definition 5, independently on the specification constraints. This kind of schemas for βs can be used as a library, thus making a first step towards the automatic generation of guaranteed correct symmetry-breaking formulae (cf., e.g., the nature of symmetry-breaking constraints added to CSPs in [7]).

6 Experiments

In this section we show that in many cases, even if applying the technique proposed in Section 5 naively, impressive speed-ups in performances can be obtained on different problems. To this end, we show the results of the following experiments, performed with Ilog OPLSTUDIO, using state-of-the-art solvers CPLEX (a MP solver) and SOLVER (a general CP one):

- Graph k-coloring, on instances from the DIMACS repository; we broke UCSs in Example 3 with β_{least} and $\beta_{\leq}$ (using CPLEX and SOLVER);
- Social golfer, on several negative instances, with β_{lex^2} (SOLVER);
- Protein folding, by using a composition of $\beta^{E,W}_{\text{least}}$ and $\beta^{N,S}_{\text{least}}$, (SOLVER), on several benchmark instances (some of them from [12]).

Results are often good: as for k-coloring using CPLEX (cf. Table 1(a)), speed-ups up to 90% have been observed for many instances (especially when using β_{least}), even if for some others the overhead of adding such constraints leads to poorer performances (cf. also [22]). As for Social golfer instead (cf. Table 1(b)), adding β_{lex^2} leads to impressive time savings on negative instances, usually around 99%. A similar behavior has been observed for Protein folding (cf. Table 1(c)) –we solved the optimization version– with savings up to 73% (often more than 50%).

7 Conclusions

In this paper we dealt with symmetry checking and breaking at the logical level of the specification. We observed that in many cases, symmetries arise from the structure of the problem, and not from input data. Hence, from a methodological point of view, it makes sense inferring such symmetries by reasoning on the problem model. Furthermore, since specifications can be regarded as logical formulae, such tasks reduce to tautology or satisfiability checking, and hence they can be automated by computer tools (even if the general problem is undecidable). To this end, in [5] we show with several examples how first-order theorem provers and finite model finders can be effectively and efficiently used in the important case where formulae to be checked are first-order.

As for symmetry-breaking, adding constraints to the specification may, in general, lead to some overhead, and we don't exclude that, for some problems,

Table 1. Solving times (seconds) for k-coloring (CPLEX) (a), Social golfer (SOLVER) (b), and Protein folding (SOLVER) (c). '–' means that the solver did not terminate in one hour

			CPLEX				
			No s.b.	β_{least}		$\beta_<$	
Instance	k	Sol?	Time	Time	% sav.	Time	% sav.
DSJC1000.1	24	N	–	368.46	>89.77	–	–
DSJC125.5	8	N	15.32	13.21	13.77	10.51	31.40
DSJC125.5	25	Y	–	2337.29	>35.08	2177.21	39.52
DSJC125.9	21	N	1408.23	2080.21	-47.72	1088.65	22.69
DSJC250.5	10	N	–	2158.75	>40.03	2432.55	32.43
DSJC500.1	11	N	2.53	–	$-\infty$	–	$-\infty$
fpsol2.i.2	21	N	139.80	43.70	68.70	102.20	26.90
fpsol2.i.2	31	Y	–	397.61	>88.96	–	–
fpsol2.i.3	31	Y	–	330.22	>90.83	–	–
le450_25a	21	N	84.73	95.32	-12.50	46.51	45.11
le450_25a	25	Y	3536.41	–	<-1.80	1783.23	49.58
miles500	19	N	2.31	–	$-\infty$	1.67	27.71
mulsol.i.1	30	N	–	10.61	>99.71	–	–
mulsol.i.1	49	Y	–	311.12	>91.36	–	–
mulsol.i.2	30	N	–	10.98	>99.70	–	–
mulsol.i.2	31	Y	26.75	48.67	-81.94	–	$-\infty$
mulsol.i.3	30	N	–	10.78	>99.70	–	–
mulsol.i.3	31	Y	55.77	43.65	21.73	284.32	-409.81
mulsol.i.4	30	N	–	10.99	>99.69	–	–
mulsol.i.4	31	Y	47.46	14.25	69.97	–	$-\infty$
mulsol.i.5	30	N	–	11.12	>99.69	–	–
mulsol.i.5	31	Y	166.85	20.68	87.61	64.56	61.31
myciel4	4	N	5.22	0.87	83.33	8.46	-62.07

(a)

Instance				SOLVER		
Plrs	Wks	Grps	Solv?	No s.b.	β_{lex^2}	% sav.
6	6	3	N	2267.35	2.12	99.91
6	7	3	N	273.53	4.23	98.45
6	8	3	N	96.67	10.31	89.33
9	5	3	N	–	1.05	>99.97
9	6	3	N	342.24	3.86	98.87

(b)

Instance		SOLVER		
Length	Contacts	No s.b.	$\beta^{E,W}_{least} \wedge \beta^{N,S}_{least}$	% sav.
14	5	45.38	15.1	66.73
14	2	34.29	10.05	70.69
16	7	23.95	13.27	44.59
16	6	124.12	44.21	64.38
17	6	2788.05	746.97	73.21
17	6	311.78	117.68	62.26
18	8	–	1660.35	>53.88
18	4	547.84	370.38	32.39
18	9	–	1830.02	>49.17

(c)

this technique may be worse than making symmetry breaking after instantiation (cf., e.g., [22]) or during search. However, in our approach, we make a strong decoupling between symmetry detection and breaking: all techniques for symmetry-breaking need to know the symmetries of the problem, and detecting structural ones at the model level can be a common task for all of them. In particular, detected symmetries can be broken, in principle, either by adding symmetry-breaking constraints to the specification, or by instructing search algorithms to break them during search (cf. Section 1 for references). Understanding which technique is better for a given specification is topic for future work.

As for the experiments presented in Section 6, it is worth noting that our goal is not to compare specification-level versus instance-level symmetry-breaking, but to give evidence that even a naive implementation of the proposed symmetry-breaking techniques may lead to consistent time savings. In our opinion, this is a very interesting point, since the required reasoning can be effectively automated in many practical circumstances. As an example, in [5] we present experimental results on using first-order theorem provers for automating these tasks. Moreover, we recall –cf. the end of Section 5– that well-behaved symmetry-breaking templates do exist, that satisfy Condition 2 of Definition 5 by design. Hence, in many practical circumstances, only Condition 1 of Definition 5 should be checked for a given specification, and this can be done very efficiently by using a finite model finder (cf. [5]).

References

1. *Proc. of CP 2001*, v. 2239 of *LNCS*, 2001. Springer.
2. E. Börger, E. Gräedel, and Y. Gurevich. *The Classical Decision Problem*. Perspect. in Math. Logic. Springer, 1997.
3. C. A. Brown, L. Finkelstein, and P. W. Purdom. Backtrack searching in the presence of symmetry. In *Proc. of Intl. Conf. on Applied Algebra, Algebraic Algorithms and Error Correcting codes*, v. 357 of *LNCS*, pp. 99–110, 1988. Springer.
4. M. Cadoli and T. Mancini. Exploiting functional dependencies in declarative problem specifications. In *Proc. of JELIA 2004*, v. 3229 of *LNAI*, 2004. Springer.
5. M. Cadoli and T. Mancini. Using a theorem prover for reasoning on constraint problems. In *Proc. of Intl. W. on Modelling and Reformulating CSPs: Towards Systematisation and Automation, in conj. with CP 2004*, 2004.
6. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. *Artif. Intell.* To appear.
7. J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR'96*, pp. 148–159, 1996. Morgan Kaufmann.
8. P. Crescenzi, D. Goldman, C. H. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *J. of Comp. Biology*, 5(3):423–466, 1998.
9. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proc. of CP 2002*, v. 2470 of *LNCS*, page 462 ff., 2002. Springer.
10. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. of CP 2001* [1], pp. 77–92.

11. R. Fourer, D. M. Gay, and B. W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming*. Intl. Thomson Publ., 1993.
12. W. Hart and S. Istrail. HP Benchmarks. Available at `http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html`.
13. J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its computational complexity*. Birkhauser Press, 1993.
14. K. F. Lau and K. A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.
15. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. on Comp. Logic*. To appear.
16. B. D. McKay. *Nauty* user's guide (version 2.2). Available at `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`, 2003.
17. P. Meseguer and C. Torras. Solving strategies for highly symmetric CSPs. In *Proc. of IJCAI'99*, pp. 400–405, 1999. Morgan Kaufmann.
18. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artif. Intell.*, 129:133–163, 2001.
19. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Math. and Artif. Intell.*, 25(3,4):241–273, 1999.
20. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., 1994.
21. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proc. of ISMIS'93*, v. 689 of *LNCS*, pp. 350–361, 1993. Springer.
22. A. Ramani, F. A. Aloul, I. L. Markov, and K. A. Sakallak. Breaking instance-independent symmetries in exact graph coloring. In *Proc. of DATE 2004*, pp. 324–331, 2004. IEEE Comp. Society Press.
23. B. M. Smith. Dual model of permutation problems. In *Proc. of CP 2001* [1], pp. 615–619.
24. P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.
25. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Tractable symmetry breaking for CSPs with interchangeable values. In *Proc. of IJCAI 2003*, pp. 277–282, 2003. Morgan Kaufmann.
26. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Compositional derivation of symmetries for constraint satisfaction. TR 2004-022, Dep. of Inf. Tech., Uppsala Univ., Uppsala, Sweden, 2004.