The Fortieth AAAI Conference on Artificial Intelligence (AAAI-26)

Faster Symmetry Breaking Constraints for Abstract Structures

Özgür Akgün[1], **Mun See Chang**[1], **Ian P. Gent**[1], **Christopher Jefferson**[2]
[1] School of Computer Science, University of St Andrews, KY16 9SX, UK
[2] Dundee International Institute of Central South University, University of Dundee, 410083, China
ozgur.akgun@st-andrews.ac.uk, msc2@st-andrews.ac.uk, ian.gent@st-andrews.ac.uk, cjefferson001@dundee.ac.uk

Abstract

In constraint programming and related paradigms, a modeller specifies their problem in a modelling language for a solver to search and return its solution(s). Using high-level modelling languages such as ESSENCE, a modeller may express their problems in terms of *abstract* structures. These are structures not natively supported by the solvers, and so they have to be transformed into or *represented* as other structures before solving. For example, nested sets are abstract structures, and they can be represented as matrices in constraint solvers. Many problems contain symmetries and one very common and highly successful technique used in constraint programming is to "break" symmetries, to avoid searching for symmetric solutions. This can speed up the solving process by many orders of magnitude. Most of these symmetry-breaking techniques involve placing some kind of ordering for the variables of the problem, and picking a particular member under the symmetries, usually the smallest. Unfortunately, applying this technique to abstract variables produces a very large number of complex constraints that perform poorly in practice. In this paper, we demonstrate a new incomplete method of breaking the symmetries of abstract structures by better exploiting their representations. We apply the method in breaking the symmetries arising from indistinguishable objects, a commonly occurring type of symmetry, and show that our method is faster than the previous methods proposed in (Akgün et al. 2025).

Code — https://github.com/stacs-cp/aaai-2026-sym-exps
Extended version — https://arxiv.org/abs/2511.11029

1 Introduction

High-level constraint modelling languages such as ESSENCE (Akgün et al. 2022) allow users to state their problems (or models) in a higher level of abstraction. For example, a user may specify their problems in terms of combinatorial structures such as partitions, relations and multisets. As constraint solvers cannot solve problems with such abstract objects directly, automatic model rewriting tools CONJURE and SAVILEROW can rewrite such a problem specification into equivalent ones suitable for a solver. Through this automated rewriting system, a domain expert no longer needs to be burdened with the task of writing long, error-prone, low-level models, and choosing among their equivalents the most performant model for the selected solver. Therefore, the solving technologies can now be made more widely available.

It is well known that the presence of symmetries can significantly impede solver performance, as the solver would spend time exploring equivalent parts of the search space. Therefore, there has been a significant amount of research on how to remove symmetries in a process called *symmetry breaking* – see (Gent, Petrie, and Puget 2006) for an overview. The most widely used method of symmetry breaking is lex-leader symmetry breaking constraints, which force assignments considered during search to be the smallest value among their symmetric equivalences.

The lex-leader constraints for a set of symmetries G are the set of constraints $X \leqslant X^g$ for every symmetry $g \in G$, where X is the list of all variables, and X^g is the result of applying the symmetry g to X. In practice, the size of the set of symmetries G is too large to add all such constraints, so we instead add constraints for some subset of G. There are a number of ways of choosing this set (Jefferson and Petrie 2011), the most notable of which is "double-lex" (Flener et al. 2002).

While we need to limit the number of constraints we generate, the individual constraints $X \leqslant X^g$ are usually expressed quite compactly. For many simple cases of symmetries, such as permutations of the elements of a list, each lex-leader constraint can be expressed as a single lexicographic ordering constraint. Other simple symmetries, such as "value symmetries" (which permute the values of variables without permuting the values of different variables), can also be expressed compactly and efficiently (Law and Lee 2006). However, when we have more complex and abstract types, such as a set of sets of integers, requiring a set to be smaller than only one symmetric equivalent still requires a large set of complex constraints. We will demonstrate in our experiments that this can slow down the solving process significantly.

In this paper, we propose a new symmetry handling technique for abstract structures, which we call "delayed symmetry application". This method has the advantage that it always produces a single lexicographic ordering constraint for each lex-leader constraint on abstract variables. While this constraint breaks less symmetries than the traditional lex-

---

leader constraints, we will show that in practice it greatly improves performance, allowing us to perform efficient symmetry breaking on a wide range of high-level types.

We apply our method to breaking the symmetries induced by *unnamed types*. Unnamed types are introduced into the high-level modelling language ESSENCE to encapsulate indistinguishable objects, one of the most common sources of symmetries. (Akgün et al. 2025) proposed methods of breaking the symmetries arising from unnamed types using the lex-leader constraints. As an ESSENCE structure can be abstract, we show that our symmetry breaking is better when we have models with abstract structures.

We begin the paper with the background on symmetry breaking in constraint programming and high-level modelling, as well as a summary of (Akgün et al. 2025). Our method requires a new ordering of abstract objects, which we shall define in Section 3. We then give our method in Section 4. We see how the method is useful for breaking the symmetries of unnamed types in Section 5.

2 Background

A *constraint satisfaction problem* (CSP) is a triple $\langle V, D, C \rangle$ such that $V = \{V_1, \ldots, V_k\}$ is a set of *variables*, each element D_i of $D = \{D_1, \ldots, D_k\}$ is a set, called the *domain* of the variable V_i, and each element C_i of $C = \{C_1, \ldots, C_l\}$, called a *constraint*, is a subset of the Cartesian product $\bigtimes_{i=1}^{k} D_i$. An element of $\bigtimes_{i=1}^{k} D_i$ is called an *assignment* of the CSP, and an assignment is called a *solution* if it is in the intersection $\bigcap_{i=1}^{l} C_i$.

A *permutation* of a set Ω is a bijective function from Ω to itself. We write permutations using the cycle notation. A *permutation group* G over Ω is a set of permutations of Ω that is closed under inverse (for all $g \in G$, its inverse g^{-1} is also in G), closed under composition (for all $g, h \in G$, its composition gh is also in G), and contains the identity permutation 1_G (the permutation that fixes all points in Ω). The *symmetric group* $\mathrm{Sym}(\Omega)$ of Ω is the permutation group consisting of all permutations of Ω. A *symmetry group* of a set X is a group G that *acts* on X. That is, the identity 1_G fixes each $x \in X$ and $(x^g)^h = x^{(gh)}$ for all $x \in X$ and $g, h \in G$, where x^g denotes the image of x under the symmetry g. Group action formalises the notion of symmetries, and its conditions establish the expected behaviour of symmetries.

In this paper, we define a symmetry group G of a CSP as a group acting on all possible assignments that preserves the solution set. This gives rise to an equivalence relation for all possible assignments (and hence the set of all solutions), where two assignments a, a' are equivalent if there exists $g \in G$ such that $a^g = a'$. A sound *symmetry-breaking constraint* is a constraint that excludes some but not all equivalent assignments. A sound symmetry-breaking constraint is *complete* if exactly one assignment per equivalence class is retained, and is said to be *incomplete* otherwise.

The *lex-leader constraint* is a general symmetry-breaking constraint that asserts that only the smallest assignment of each equivalence class should be considered. For a symmetry group G of the domain $\mathrm{Dom}(X)$ of a variable X, the full lex-leader constraint $LL_{\leqslant}(G, X)$ with respect to a total ordering $\leqslant$ of $\mathrm{Dom}(X)$ is the constraint $\forall g \in G.\, X \leqslant X^g$. Note that a symmetry-breaking constraint is sound if it is implied by the full lex-leader constraint, and is complete if the converse implication also holds. A commonly used order is the lexicographical ordering: for a total order $\leqslant$ on set S, the *lexicographical ordering* $\leqslant_{lex}$ *over* $\leqslant$ is the total ordering on the tuples over S such that $(s_1, \ldots, s_k) \leqslant_{lex} (s'_1, \ldots, s'_k)$ if and only if either $s_i = s'_i$ for all i, or there is an i such that $s_i < s'_i$ and $s_j = s'_j$ for all $j < i$.

High-level Modelling with Essence ESSENCE is a constraint modelling language that allows the *abstract type constructors* of: sets, multisets, sequences, functions, relations and partitions, as well as other *concrete type constructors* such as tuples, matrices, integers and Booleans. A type in ESSENCE can be an arbitrarily nested construction of these constructors. For example, we can have a multiset of partitions of tuples of Booleans. We say that a type is *abstract* if its construction involves at least one abstract constructor, and is *concrete* otherwise. The method proposed in this paper will only affect the symmetry breaking where we have variables of abstract type. We use the notation *matrix indexed by* $[I_1, \ldots, I_k]$ to refer to a k-dimensional ($k \geqslant 1$) matrix where the values are $m[i_1] \ldots [i_k]$, for each i_j in I_j.

Remark 2.1. We shall use (Akgün et al. 2025, Remark 1) (reproduced in Appendix C) to express the values of each ESSENCE type in terms of multisets, tuples and matrices. For example, functions $f : T_1 \to T_2$ can be seen as a set of tuples $\{(t, f(t)) \mid t \in T_1\}$. Note that this gives the semantics of types which we use in the proofs, but does not necessarily give how the values are represented in the solvers.

CONJURE is an automatic model rewriting tool that rewrites or *refines* models in ESSENCE into models in ESSENCE PRIME, a constraint modelling language which does not allow abstract types. It does so by rewriting abstract variables into their representations (a.k.a. viewpoints) as concrete types, possibly by imposing additional constraints (called *structural constraints*), and replacing existing constraints relating to these abstract variables with a set of equivalent constraints over concrete types.

Representations are well-studied, so we will not detail them here. However, we will be discussing two fundamental representations of multisets (a multiset is a generalisation of sets where element repetitions are allowed): for a multiset s over T, the *explicit representation* of s is an (ascending) ordered matrix whose elements are elements of s; the *occurrence representation* of s is a matrix indexed by T of integers, where the entry at index t is the number of occurrences of t in s. The explicit and occurrence representations of sets are similar, but we require the matrix in the explicit representation to be strictly ascending and we have Boolean elements instead of integer elements in the occurrence case.

Remark 2.2. It is important to note that representations are carefully designed not to introduce any symmetries, which gives us some useful functions. When we represent a variable X of type T into variable X' of type T', we have an injection $\psi : \mathrm{Dom}(X) \to \mathrm{Dom}(X')$ together with a structural constraint S on X' such that $t' \in \mathrm{Im}(\psi)$ if and only if t' satisfies S. Taking the values $\mathrm{Val}(T) = \mathrm{Dom}(X)$ and $\mathrm{Val}(T')$ to be the values in $\mathrm{Dom}(X')$ that satisfy its structural constraints, we have a bijection $\phi : \mathrm{Val}(T) \to \mathrm{Val}(T')$.

For example, when we represent sets using the explicit representation, $\mathrm{Dom}(X)$ and $\mathrm{Val}(T)$ consist of sets, $\mathrm{Dom}(X')$ consists of lists, and $\mathrm{Val}(T')$ consists of elements of $\mathrm{Dom}(X')$ which are strictly ordered lists. More information on the representations and refinements used in CONJURE can be found in (Akgün et al. 2022).

Breaking the Symmetries of Indistinguishable Objects

A commonly occurring source of symmetries is indistinguishable objects. These are objects where permuting the objects gives an equivalent solution. These occur in many problems: e.g. in a delivery problem, we may treat trucks and drivers as indistinguishable; when setting up a rota for a whole work week, we may also treat days of the week as indistinguishable. ESSENCE allows one to express the notion of indistinguishable objects using *unnamed types*, which allows correct symmetry-breaking constraints to be *automatically* generated without requiring expert knowledge.

An unnamed type T can be thought of as an enumerated type together with a symmetry group that allows all permutations of its values. (Multiple) unnamed types can be used in nested constructions of types, e.g. `matrix indexed by [T,U] of set (size 2) of T`, where T and U are distinct unnamed types. More information about unnamed types can be found in (Akgün et al. 2025). It it important to note that unnamed type symmetries do not fall neatly into well-studied symmetry classes such as variable and value symmetries, and go beyond those investigated in the literature such as (Walsh 2006; Law and Lee 2004).

(Akgün et al. 2025) shows how the symmetries of unnamed types can be consistently broken by adding constraints of the form $X \leqslant X^g$ where X is a variable, g is a symmetry of unnamed types, X^g represents the variable after applying the induced symmetry of g and $\leqslant$ is an ordering of the possible values of X. This required defining, for each possible (nested) type, (1) the symmetry of its values induced by the symmetries of unnamed types (in Def 2), and (2) a total order for its values (in Def 4 & Rem 2). Recall Remark 2.1. We reproduce slightly generalised versions here for completeness.

Definition 2.1. Let G be a symmetry group of the values of a type T, and U be any type (whose construction may contain T). Then the *naturally induced action of* G on $\mathrm{Val}(U)$ is the action defined by: for all $g \in \mathrm{Sym}(T)$ and $u \in \mathrm{Val}(U)$, (a) if $T = U$, then u^g is the image under the action on $\mathrm{Val}(T)$; (b) if u not constructed from T, then $u^g = u$; (c) if u is a matrix indexed by $[I_1, \ldots, I_k]$ of E, then the image u^g is a matrix where its i-th element $u^g[i]$ is $(u[i^{(g^{-1})}])^g$, where $i^{(g^{-1})}$ denotes the preimage of i under g; (d) if u is a multiset $\{u_1, \ldots, u_k\}$, then $u^g = \{u_1^g, \ldots, u_k^g\}$; (e) if u is a tuple $(u_1, \ldots, u_k)$, then $u^g = (u_1^g, \ldots, u_k^g)$.

Definition 2.2. We define a total ordering $\preceq_T$ for values $\mathrm{Val}(T)$ of a type T recursively by: (a) if $\mathrm{Val}(T)$ consists of integers, we take $\preceq_T$ to be $\leqslant$ on integers; (b) if $\mathrm{Val}(T)$ consists of Boolean, we use `false` $\prec_T$ `true`; (c) if $\mathrm{Val}(T)$ consists of enumerated types, then $x \prec_T y$ if x occurs before y in the definition of the enumerated type; (d) if $\mathrm{Val}(T)$ consists of matrices or tuples of an inner type S, then take $\preceq_T$ to be lexicographical order $\preceq_{Slex}$ over an order $\preceq_S$ for the inner type; (e) if $\mathrm{Val}(T)$ consists of multisets of type S, take $\preceq_T$ to be such that $x \preceq_T y$ if and only if $[-\mathrm{freq}(x, s_i) \mid 1 \leqslant i \leqslant |S|] \preceq_{Slex} [-\mathrm{freq}(y, s_i) \mid 1 \leqslant i \leqslant |S|]$, where $\mathrm{freq}(z, a)$ denote the frequency of a in set z, the ordering $\preceq_{Slex}$ is the lexicographical order over a total order $\preceq_S$ of S, and each s_i is the i-the smallest value in S w.r.t. $\preceq_S$.

CONJURE rewrites a model with unnamed types into an ESSENCE PRIME model using these definitions of symmetries and ordering, in conjunction with the existing rewriting methods. The method is illustrated in the following example.

Example 2.1. Suppose that we have a variable X of type `set (size 3)` of unnamed type T of size 4. We consider the values $\mathrm{Val}(T)$ of T as 'tagged integers' $1_T, 2_T, 3_T, 4_T$. We have put a label on the values and hence distinguished them, so any permutation of $1_T, 2_T, 3_T, 4_T$ is a symmetry of the values $\mathrm{Val}(T)$.

These symmetries of $\mathrm{Val}(T)$ induce symmetries on the values of X. For example, consider the symmetry $g := (1_T, 2_T)(3_T, 4_T)$, which swaps 1_T and 2_T, and simultaneously swaps 3_T and 4_T. The image under the symmetry of a possible value $x = \{1_T, 2_T, 3_T\}$ of X is $x^g := \{1_T^g, 2_T^g, 3_T^g\} = \{2_T, 1_T, 4_T\}$, which is $\{1_T, 2_T, 4_T\}$, as order does not matter in sets.

We add symmetry-breaking constraints for X consisting of constraints of the form $X \leqslant X^s$ for some total ordering $\leqslant$ of the domain of X and symmetry s. To check if the constraint is falsified for the value x of X and symmetry g, we need to check if $\{1_T, 2_T, 3_T\} \leqslant \{1_T, 2_T, 4_T\}$. Since $[-\mathrm{freq}(x, i_T) \mid 1 \leqslant i \leqslant 4] = [-1, -1, -1, 0] \preceq_{lex} [-1, -1, 0, -1] = [-\mathrm{freq}(x^g, i_T) \mid 1 \leqslant i \leqslant 4]$, the constraint $X \leqslant X^s$ is satisfied.

Finally, we present a complication when breaking the symmetries of nested abstract objects. In the rest of this paper, we will more formally discuss the issues raised in Example 2.2, and how we solve them.

Example 2.2. Extending Example 2.1, consider a variable Y of type `set (size 2) of set (size 3) of T`, an assignment $v := \{\{2_T, 3_T\}, \{2_T, 4_T\}, \{1_T, 3_T\}\}$ of Y and the permutation $g := (1_T, 2_T)$ that swaps the values 1_T and 2_T. Let us assume that we represent Y using the explicit representation for the outer set, and the occurrence representation for the inner sets.

So v is represented as $[[0,1,0,1],[0,1,1,0],[1,0,1,0]]$. Its image v^g is $\{\{1_T, 3_T\}, \{1_T, 4_T\}, \{2_T, 3_T\}\}$, which is represented by $[[0,1,1,0],[1,0,0,1],[1,0,1,0]]$. Note that, as g maps the smaller set in v to the larger set in v^g and explicit representations require their elements to be ordered, we have to re-order the inner sets. This re-ordering of the inner sets is what makes implementing $Y \leqslant Y^g$ difficult. This constraint is currently implemented by splitting it into two constraints, $Y \leqslant Z$ and $Z = Y^g$. The first $Y \leqslant Z$ can be implemented as a single lexicographic ordering constraint for any type in ESSENCE, but $Z = Y^g$ is more difficult. In practice, we impose constraints of the form "for every $y \in Y$, there exists $z \in Z$, such that $y^g = z$". These constraints are required to correctly find the image of Y under g, but greatly increase the size of our model, slowing down the solving process.

3 Representation-Dependent Ordering

The lex-leader method uses the conjunction of constraints of the form $X \leqslant X^g$, where $\leqslant$ is a total order of $\mathrm{Dom}(X)$. The lex-leader method correctly breaks all symmetries as long as we have a total order, and does not restrict the ordering to be used. When using pre-determined ordering of abstract objects from the literature (such as in (Frisch et al. 2003; Akgün et al. 2025)), determining if a nested set is smaller than the other can be slow. This is because these orderings of multisets are essentially the occurrence representations of multisets. If we choose to represent multisets in another way, e.g., as explicit ordered lists, then we effectively have two representations (one for solving and one for ordering), which may not be what we want. Further, if the multisets are deeply nested, the representation may be very long.

Example 3.1. Consider the Balanced Incomplete Block Design problem (Prestwich (no date)), where one must find an arrangement of v distinct objects into b blocks of size k satisfying certain constraints. We may model this using a decision variable `bibd` of type `set (size b) of set (size k) of T` where `T` is an unnamed type of size v. To break the symmetries of the unnamed type `T`, we add constraints of the form `bibd` $\leq_N$ `bibd`^{g} for $g \in \mathrm{Sym}(\mathrm{Val}(T))$, where $\leq_N$ is an ordering of nested sets.

As a toy example, suppose that $b = 3, k = 2$ and $v = 4$. Now, to determine if `bibd` $\leq_N$ `bibd`^{g} for some symmetry g, if we use the method from (Akgün et al. 2025), we need the frequency tuple $[-\mathrm{freq}(v, \omega_i) \mid 1 \leqslant i \leqslant |\Omega|]$, where Ω is the set consisting of all 2-sets of $\mathrm{Val}(T)$. Since elements of Ω are also sets, we again use the frequency tuples of these inner sets to give an ordering of Ω. This ordering has $\{1,2\} \leq \{1,3\} \leq \{1,4\} \leq \{2,3\} \leq \{2,4\} \leq \{3,4\}$. For example, the frequency tuple of $v := \{\{2_T, 3_T\}, \{2_T, 4_T\}, \{1_T, 3_T\}\}$ is $[0, -1, 0, -1, -1, 0]$.

However, abstract objects would need to be represented to be fed into solvers. Suppose that we represent the variable `bibd` as an explicit representation of occurrence representation. This means that `bibd` is replaced with a variable `m`, of type `matrix indexed by [int(1..b), T] of Bool`, a 2-dimensional matrix where each row of `m` represents one inner set s_i of `bibd`, and `m`$[i][j]$ is true if s_i contains `j`. Then, to determine the first element of the frequency tuple of `bibd`, we need to know if $\{1_T, 2_T\}$ is an element of `bibd`, which means that we need to know if $\bigvee_{1 \leqslant i \leqslant 3}(m[i][1_T] \wedge m[i][2_T])$. This means that in the worst case, we need to know half of the values of m before we can use the lex-leader to break symmetries.

Suppose further that we have a symmetry $g := (1_T, 3_T)(2_T, 4_T)$, and we want to check if `bibd` $\leq_N$ `bibd`^{g}. Then, to determine the first element of the frequency tuple of `bibd`^{g}, again we need to know if $\{1_T, 2_T\} \in$ `bibd`^{g}, but this is only true if $\{3_T, 4_T\} \in$ `bibd`. Similarly to before, to determine this last inclusion, we need to know if $\bigvee_{1 \leqslant i \leqslant 3}(m[i][3_T] \wedge m[i][4_T])$. So, in the worst case, we need to know the other half of m to determine the first value of the frequency tuple of `bibd`^{g}, and so we need to have all elements of m to determine if `bibd` $\leq_N$ `bibd`^{g}.

In this paper, we introduce an alternative ordering of abstract types that depends on the representation used, thereby giving us a representation-dependent ordering that is dynamic during compilation, but not during search. We now assume that we have constraints of the form $X \leq_T Y$ where both sides of the inequality are of *abstract* type T. These constraints would need to eventually be rewritten into equivalent constraints in terms of concrete types only, according to a choice of representation. We assume that the choice of representation for X and Y is the *same*. This is the case when we want to break the symmetries of unnamed types, as Y is always the image X^g of X under some symmetry g.

We would want to rewrite the constraint $X \leq_T Y$ into an equivalent constraint $X' \leq_{T'} Y'$ for some ordering $\leq_{T'}$ over values of T', where X and Y are represented by expressions X' and Y' respectively. For these to be equivalent, for any $x, y \in \mathrm{Dom}(X)$, we need $x \leq_T y$ exactly when $x' \leq_{T'} y'$, where $x', y' \in \mathrm{Dom}(X')$ are the corresponding values representing x and y respectively. Here, we shall define a total ordering $\leq_T$ for $\mathrm{Dom}(X)$ as *exactly* the ordering that satisfies this requirement. Recall Remark 2.2.

Definition 3.1. For a variable X of abstract type T, we define a total order $\dot{\leqslant}_T$ of $\mathrm{Dom}(X)$ by: if X is to be represented with a variable X' of type T' via an injection $\psi : \mathrm{Dom}(X) \to \mathrm{Dom}(X')$, then, for $a, b \in \mathrm{Dom}(X)$, take $a \dot{\leqslant}_T b$ if and only if $\psi(a) \dot{\leqslant}_{T'} \psi(b)$, where $\dot{\leqslant}_{T'}$ is a total order of $\mathrm{Dom}(X')$.

The fact that $\dot{\leqslant}_T$ is well-defined since the domain of an injective function f induces a total ordering of its image. The full proof can be found in Appendix A.

As long as we have defined total ordering for concrete types, we obtain total orderings on all types using Definition 3.1 It is crucial that all representations of abstract variables do not introduce any symmetries, which is indeed true for CONJURE.

Example 3.2. Let variable X be of type sets of size 3 over $\{1, \ldots, 5\}$. Suppose that we take the explicit representation of sets, so a value of X is represented as an ordered matrix over $\{1, \ldots, 5\}$. Then the ordering $\dot{\leqslant}$ over $\mathrm{Dom}(X)$ is defined as the ordering satisfying: $\{x_1, x_2, x_3\} \dot{\leqslant} \{y_1, y_2, y_3\}$ whenever $[x_1, x_2, x_3] \leqslant_{lex} [y_1, y_2, y_3]$, assuming $x_1 \leqslant x_2 \leqslant x_3$ and $y_1 \leqslant y_2 \leqslant y_3$. In particular, the set $\{1, 2, 3\}$ is less than the set $\{1, 3, 4\}$ because $[1, 2, 3] \leqslant_{lex} [1, 3, 4]$.

To see why it is important that rewriting does not introduce any symmetries, suppose instead that we try to represent sets as unordered matrices. Then $[2, 1, 3]$ and $[1, 2, 3]$ both represent the set $\{1, 2, 3\}$. Now $[1, 2, 4] \leqslant_{lex} [2, 1, 3]$ gives $\{1, 2, 4\} \dot{\leqslant} \{1, 2, 3\}$, but $[1, 2, 3] \leqslant_{lex} [1, 2, 4]$ gives $\{1, 2, 3\} \dot{\leqslant} \{1, 2, 4\}$, a contradiction.

4 Delayed Symmetry Application

Now that the ordering is in terms of the representation used, it can still be difficult to establish if an abstract object is smaller than the other:

Example 4.1. Recall Example 3.1. As before, we add lex-leader constraints of the form `bibd<=bibd`^{g}, but this time, we have `<=` to denote the representation-dependent total ordering of the possible values of `bibd`.

When CONJURE rewrites `bibd` into its representation `m`, we introduced new symmetries of `m` – the symmetry on the first index. Typically, such a symmetry is always broken by adding constraints to order the elements of `m`. This is indeed what is automatically done by CONJURE upon the introduction of `m`. This ensures that we have an underlying bijection from possible values of `bibd` to the possible values of `m` which satisfies this additional ordering constraint.

Let us first work out, theoretically, if a value $v := \{\{2_T, 3_T\}, \{2_T, 4_T\}, \{1_T, 3_T\}\}$ of `bibd` satisfies `bibd <= bibd`^{g} for $g := (1_T, 2_T)$. As seen in Example 2.2, the representation v' of v is $[[0,1,0,1], [0,1,1,0], [1,0,1,0]]$, which is lexicographically smaller than the representation $(v^g)' = [[0,1,1,0], [1,0,0,1], [1,0,1,0]]$ of v^g, so the constraint is satisfied.

When solving the problem using a constraint solver, we have to work with the lower-level Boolean matrices. So applying the symmetry g to `bibd` corresponds to swapping the first two columns of `m`. However, we now need to reorder the rows of `m` so that its elements are in increasing order. In our example, permuting the first two columns of v' gives $v'^g := [[1,0,0,1], [1,0,1,0], [0,1,1,0]]$, and reordering gives exactly $(v^g)'$. This means that determining if `bibd<=bibd`^{g} by considering the representative values is quite difficult: we need to apply the symmetry to the representative value and then reorder. In CONJURE, we implement this by introducing an auxiliary copy `ma` of `m` and constraints that enforce `ma` is a correct refinement of `bibd`^{g}, where g acts as described above – including reordering the rows. This creates a significantly larger model, in terms of both variables and constraints.

Instead, we propose not to reorder after applying the symmetries to the representative value. We will show that not sorting the rows after applying g to each row of `m` creates a sound but incomplete symmetry-breaking constraint. This is equivalent to applying the permutation to the representation v' of the set v to obtain v'^g, instead of applying the permutation to the set itself before taking its representation $(v^g)'$, effectively delaying the symmetry application until after taking representations. Notice that $(v^g)'$, which is sorted, cannot be greater than v'^g. This means that each ordering constraint `m <= m`^{g} is implied by the ordering constraint `bibd <= bibd`^{g}. So the constraint $LL_{\leqslant}(\texttt{m}, G)$, being implied by the full lex-leader $LL_{\leqslant}(\texttt{bibd}, G)$, is a sound symmetry breaking constraint.

We formalise this observation in Lemma 4.1, whose proof is a straightforward application of the assumptions.

Lemma 4.1. *Recalling Remark 2.2, suppose that a variable X of type T is to be represented as a variable X' of type T' using an underlying bijection $\phi : \mathrm{Val}(T) \to \mathrm{Val}(T')$. Let $\leq_T$ and $\leq_{T'}$ be total orderings of $\mathrm{Dom}(X)$ and $\mathrm{Dom}(X')$ respectively, such that $x \leq_T y \Leftrightarrow \phi(x) \leq_{T'} \phi(y)$. Let G be a symmetry group of the values of a type U and consider the naturally induced actions of G. If ϕ satisfies:*

$$\phi(x^g) \leq_{T'} \phi(x)^g \text{ for all } g \in G \text{ and } x \in \mathrm{Val}(T), \quad (1)$$

then for any $g \in G$, the constraint $X' \leq_{T'} (X')^g$ is implied by the constraint $X \leq_T X^g$. Further, if equality is attained in Equation (1), then the two constraints are equivalent.

The condition $x \leq_T y \Leftrightarrow \phi(x) \leq_{T'} \phi(y)$ holds for orderings from Definition 3.1. If Equation (1) also holds, then the constraint $X' \leq_{T'} (X')^g$ can replace the constraint $X \leq_T X^g$ as a possibly weaker symmetry-breaking constraint. Note that no constraints are added or removed in this replacement.

Lemma 4.2 shows that Equation (1) is satisfied for the two common representations of multisets. A slight generalisation of the example shows that the condition holds for the explicit representation. For the occurrence representation, we have the stronger property of $\phi(x^g) = \phi(x)^g$, which follows from Definition 2.1. See Appendix B for the full proof.

Lemma 4.2. *Let G be a symmetry group of the values $\mathrm{Val}(U)$ of a type U. For a multiset type T, considering the naturally induced actions of G, and letting $\leq_S$ be a total order for the inner type.*

1. *if we represent T using the explicit representation where the elements are ordered using $\leq_S$. taking $\leq_{T'}$ to be the lexicographical order over $\leq_S$, then $\phi(x^g) \leq_{T'} \phi(x)^g$ for all $g \in G$ and $x \in \mathrm{Val}(T)$.*
2. *if we represent T using the occurrence representation where we order the indices using $\leq_S$, then $\phi(x^g) = \phi(x)^g$ for all $g \in G$ and $x \in \mathrm{Val}(T)$.*

It is important to be careful with the ordering used, as there is a risk of incompatible orderings. We avoid this complication in our implementation by always using the natural, ascending order.

Example 4.2. Let x be the set $\{1, 2\}$ and g be the permutation $(1, 3)$ that swaps 1 and 3. Suppose that we represent sets using the explicit representation. Then $x^g = \{2, 3\}$ and so $\phi(x^g) = [2, 3]$, while $\phi(x) = [1, 2]$ and so $\phi(x)^g = [3, 2]$. In this case $\phi(x^g) \leqslant_{lex} \phi(x)^g$.

Let $\leqslant_{\overline{lex}}$ be the lexicographical order that compares the elements in descending order. Then clearly it is not true that $\phi(x^g) \leqslant_{\overline{lex}} \phi(x)^g$. This means that, if we choose to order lists using this ordering instead of the usual lex-ordering, our method may not work. Similarly, if the refinement represents sets as descending ordered lists, then $\phi(x^g) = [3, 2]$, while $\phi(x) = [2, 1]$ and so $\phi(x)^g = [2, 3]$. Here again it is not true that $\phi(x^g) \leqslant_{lex} \phi(x)^g$.

5 Breaking Unnamed Type Symmetries

We apply our method to breaking the symmetries induced by unnamed types in ESSENCE. Other technologies can follow a similar method to handle symmetries of indistinguishable objects.

5.1 Method Summary

We start with an ESSENCE model M with unnamed types $T_1, T_2, \ldots, T_k$. We obtain an ESSENCE PRIME model where the unnamed type symmetries are soundly (but possibly incompletely) broken using the following steps.

1. Replace each unnamed type T_i by tagged integers of values $1_{T_i}, 2_{T_i}, \ldots, |T_i|_{T_i}$.
2. Letting $V_1, V_2, \ldots, V_n$ be the decision variables of M, let X be a new decision variable that represents the tuple $(V_1, V_2, \ldots, V_n)$.
3. Add constraints of the form $X \,\mathtt{.<=}\, \mathtt{transform}(gs, X)$ for some permutation combinations $gs = (g_1, g_2, \ldots, g_k)$, where each $g_i \in \mathrm{Sym}(\mathrm{Val}(T_i))$. Here, `.<=` is $\dot{\leqslant}_T$ from Definition 3.1 for the appropriate type T, and $\mathtt{transform}(g, X)$ represents the image of X under the symmetry g from Definition 2.1.
4. Go through CONJURE refinements to rewrite abstract variables into concrete variables, according to some representation.
   (a) If X is abstract and is refined to an expression X', then the constraint $X \,\mathtt{.<=}\mathtt{transform}(gs, X)$ is rewritten to $X' \,\mathtt{.<=}\mathtt{transform}(gs, X')$.
   (b) If X is concrete, then it is rewritten to $X \mathtt{<=} \mathtt{transform}(gs, X)$, for an appropriate concrete ordering <=. The expressions of form $\mathtt{transform}(g, X)$ are subsequently removed using the definition of symmetry application on concrete type from Definition 2.1.

Steps 1–3 follow the procedure in (Akgün et al. 2025, Proposition 1) to obtain a model without unnamed types. However, we replace the static ordering $\preceq$ from Definition 2.2 with our representation-dependent ordering $\dot{\leqslant}$ from Definition 3.1. This is correct because lex-leader expressions require a total order, but it does not matter which. If T' is concrete, we replace $\dot{\leqslant}_{T'}$ with an ordering of concrete type (Step 4b), which should natively exist in most solvers. We never need to implement an ordering for abstract types, in contrast to the methods in (Akgün et al. 2025).

In Step 4(a), constraints of the form $X \dot{\leqslant}_T Y$, are then rewritten (and possibly weakened) into $X' \dot{\leqslant}_{T'} X'^g$ using Lemma 4.1. The condition "$x \preceq_T y \Leftrightarrow \phi(x) \preceq_{T'} \phi(y)$" is attained since we use the ordering from Definition 3.1. However, for this method to be sound, we need to show that *all* representations in CONJURE satisfy Equation (1):

Lemma 5.1. *Let G be a symmetry group of the values* $\mathrm{Val}(U)$ *of a type U. For each representation in* CONJURE, *letting the underlying bijection be $\phi : \mathrm{Val}(T) \to \mathrm{Val}(T')$ and considering the natural induced actions of G, we have $\phi(x^g) \dot{\leqslant}_{T'} \phi(x)^g$ for all $x \in \mathrm{Val}(T)$ and $g \in G$, where $\dot{\leqslant}_{T'}$ is as defined in Definition 3.1.*

More details on the proof of this claim can be found in Appendix C. As observed in (Akgün et al. 2025, Remark 1), all types in CONJURE can be expressed in terms of multisets (which are abstract), matrices and tuples (which are concrete and hence do not have to be represented). So every representation of abstract types used in CONJURE is a generalisation of the two representations of multisets from Lemma 4.2.

5.2 Experiments

We compare the performance of our method against that in (Akgün et al. 2025). All experiments are run on Core i7-13700HX, Ubuntu 25.04 with 32 GB RAM. See the linked code repository to see how to reproduce the results.

So we can focus on the performance of symmetry breaking, rather than considering complete problems, we will consider finding all assignments to a single abstract variable, which is constructed from unnamed types. We also do not dictate which representations to use, but instead let CONJURE decide the most suitable ones, and report those that use a representation where our method produces a different constraint, such as when we have explicit representations of sets. We believe these experiments show the potential benefits of our new technique.

We summarise the results of our experiment in Table 1. The '**Old**' and '**New**' columns denote the method in (Akgün et al. 2025) and our new method, respectively. We consider six different high-level variables, each built using a single unnamed type. For each problem, we select the smallest instance where the 'Old' technique takes more than a second. The time taken varies between experiments because the set of solutions to each of our problems grows extremely rapidly as the parameters are increased. The exact instances used are provided in the supplementary data.

All experiments are run with the `Consecutive` and `Independently` options, the best performing options from (Akgün et al. 2025). The `Consecutive` option generates lex-leader for only the permutations that swap consecutive values of the unnamed type. We do not report the timings for `AllPairs` (which uses all permutations that swap 2 values), as they show a similar pattern. We also do not report the timings when using all permutations (which gives the full lex-leader with 'Old') because it is too slow, especially for the 'Old' method, as it introduces an exponential number of symmetry-breaking constraints. Our experiments also do not include models with more than one unnamed type, as the number of unnamed types is irrelevant to our new method. We have performed further tests with multiple unnamed types, and the results are similar to those presented here. As these experiments only include one unnamed type, the other incomplete options of `Independently` and `Altogether` from (Akgün et al. 2025) are equivalent.

The problems are solved using the constraint solver MINION (Gent, Jefferson, and Miguel 2006), and SAVILEROW (Nightingale et al. 2017) is used to convert the ESSENCE PRIME outputted by CONJURE into input for MINION. The 3 inner rows denote the total number of solutions, the number of nodes in MINION and the total time taken, in seconds.

The main takeaway from our results is that while our method usually produces more solutions, the time taken is significantly shorter, often by orders of magnitude. This pattern continues for larger instances; we found larger instances of our problems, which could be solved in a few seconds by our new method, but timed out after a day with the old method. Furthermore, and more surprisingly, the number of search nodes is often smaller by orders of magnitude as well. This is, however, not true in the single FUNCTION case as it is the simplest type we consider, and so the gain of our technique is smallest here – although it still outperforms the old method in terms of runtime.

Instance	FUNCTION		SET OF FUNCTION		SET OF MATRIX	
	Old	New	Old	New	Old	New
Solutions	1,469,103	1,492,818	25,612	34,790	5,621	9,979
Nodes	2,982,777	4,512,188	637,244	70,570	526,917	19,970
Time	27.	15.	7.2	0.07	3.5	0.01

Instance	SET OF MULTISET		SET OF RELATION		SET OF SET	
	Old	New	Old	New	Old	New
Solutions	18	21	5,761,575	8,775,909	1,019	3,897
Nodes	91,358	52	114,372,858	17,551,836	388,034	7,944
Time	1.70	0.00	895.	1.78	3.2	0.00

Table 1: Experimental results comparing the old and the new methods for various variable types

Parameter	High New		High Old		High NoSym		Matrix DoubleLex	
	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
14-6-2	$1.04e2$	0.00	-	-	$1.11e8$	$6.67e2$	$1.30e3$	0.01
8-4-3	35	0.00	$2.78e6$	$1.41e2$	$2.51e2$	0.00	$1.02e2$	0.00
8-4-4	$1.86e3$	0.00	$1.13e5$	$3.97e1$	$1.13e5$	0.29	$1.14e4$	0.02

Parameter	Low New		Low Old		Low NoSym		Matrix NoSym	
	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
14-6-2	81	0.00	81	0.01	-	-	-	-
8-4-3	22	0.00	22	0.00	$6.80e2$	0.00	$6.80e2$	0.00
8-4-4	$8.40e3$	0.01	$8.40e3$	0.01	-	-	-	-

Table 2: Solving BIBD with a variety of symmetry breaking methods, where '-' denotes timeout after 3600 seconds

5.3 Case Study: BIBD

As a complete example, we consider finding one solution for the Balanced Incomplete Block Design problem (for the definition, see (Prestwich (no date)). Given BIBD parameters (v-k-λ), we model the central BIBD variable in 3 ways:

- **High**: A `set (size b) of set (size k) of Obj` for a new unnamed type `Obj` of size `v`.
- **Low**: A Boolean 2-D matrix indexed by two unnamed types, `Obj` of size `v` and `Blocks` of size `b`.
- **Matrix**: A 2-D Boolean matrix indexed by integer ranges, `1..v` and `1..b`.

We consider breaking symmetries of the first two models using `Consecutive` and `Independently` generation, with both the **Old** and our **New** symmetry-breaking method, or with no added symmetry-breaking constraints (**NoSym**). In the 3rd model, we consider using "double-lex" (**DoubleLex**), or no symmetry breaking (**NoSym**). The models can be found in the supplementary data, and the results are given in Table 2. We see the following results:

- Low NoSym and Matrix NoSym do worst, as these are equivalent. This shows that any symmetry breaking is better than no symmetry breaking.
- High NoSym does better than other forms of no symmetry breaking because Conjure always breaks symmetry whenever a set is turned into a matrix – this cannot be turned off.
- In the Low model, the New and Old symmetry breaking are identical – this is expected, as this model contains no abstract types.
- When using "New" symmetry breaking, High now produces equally good results to the other models (both Low models) that are equivalent to DoubleLex. The variations come about from variables and constraints being outputted in different orders.

These results show the efficiency of double lex – when it is being used, all models finish in almost no time. We could look at larger experiments to see where double-lex performs worse, but this would not give any insights for this paper, and is already a very well-studied problem.

6 Conclusion and Future Work

In this paper, we show that symmetry breaking for abstract types can be quite difficult, and propose a new symmetry-breaking method using two innovations: using representation-specific ordering instead of a pre-defined ordering, and breaking symmetries using the images of *representations* of abstract values instead of the images of the abstract values themselves. We show how the method can be combined with the methods in (Akgün et al. 2025) to give faster symmetry breaking of unnamed types, which represent the commonly occurring indistinguishable objects. Our method also does not require implementation of the symmetry application and ordering of *abstract* types given in (Akgün et al. 2025), simplifying implementations. While the method is not complete, we show that it can be much faster.

This method gives another dimension of variability for the incomplete symmetry breaking for indistinguishable objects. Comparing these choices and deciding the level of symmetry breaking for a given problem is future work. Another interesting direction is to investigate whether we can use the method to find or count all solutions modulo symmetries.

Acknowledgements

This work was supported by the International Science Partnerships Fund (ISPF) and the grant was awarded by the Engineering and Physical Sciences Research Council on behalf of UK Research and Innovation (UKRI) [EP/Y000609/1].

References

Akgün, Ö.; Chang, M. S.; Gent, I. P.; and Jefferson, C. 2025. Breaking the Symmetries of Indistinguishable Objects. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 152–168. Springer Nature Switzerland.

Akgün, Ö.; Frisch, A. M.; Gent, I. P.; Jefferson, C.; Miguel, I.; and Nightingale, P. 2022. Conjure: Automatic generation of constraint models from problem specifications. *Artificial Intelligence*, 310.

Flener, P.; Frisch, A. M.; Hnich, B.; Kiziltan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. 2002. Breaking row and column symmetries in matrix models. In *Principles and Practice of Constraint Programming*, 462–477. Springer.

Frisch, A. M.; Miguel, I.; Kiziltan, Z.; Hnich, B.; and Walsh, T. 2003. Multiset ordering constraints. In *International Joint Conferences on Artificial Intelligence*, volume 3, 221–226.

Gent, I. P.; Jefferson, C.; and Miguel, I. 2006. MINION: A Fast, Scalable, Constraint Solver. In *European Conference on Artificial Intelligence*, 98–102. IOS Press. ISBN 1586036424. Software available at https://github.com/minion/minion.

Gent, I. P.; Petrie, K. E.; and Puget, J.-F. 2006. Chapter 10: Symmetry in Constraint Programming. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 329 – 376. Elsevier.

Jefferson, C.; and Petrie, K. E. 2011. Automatic Generation of Constraints for Partial Symmetry Breaking. In *Principles and Practice of Constraint Programming*, 729–743. Springer Berlin Heidelberg. ISBN 978-3-642-23786-7.

Law, Y. C.; and Lee, J. H. M. 2004. Global Constraints for Integer and Set Value Precedence. In *Principles and Practice of Constraint Programming*, 362–376. Springer Berlin Heidelberg. ISBN 978-3-540-30201-8.

Law, Y. C.; and Lee, J. H. M. 2006. Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction. *Constraints*, 11(2): 221–267.

Nightingale, P.; Akgün, O.; Gent, I. P.; Jefferson, C.; Miguel, I.; and Spracklen, P. 2017. Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251: 35–61.

Prestwich, S. (no date). CSPLib Problem 028: Balanced Incomplete Block Designs. http://www.csplib.org/Problems/prob028.

Walsh, T. 2006. General Symmetry Breaking Constraints. In *Principles and Practice of Constraint Programming*, 650–664. Springer Berlin Heidelberg. ISBN 978-3-540-46268-2.