Exact and Approximate Strategies for Symmetry Reduction in Model Checking

Alastair F. Donaldson[⋆] and Alice Miller

Department of Computing Science
University of Glasgow
Glasgow, Scotland
{ally, alice}@dcs.gla.ac.uk

Abstract. Symmetry reduction techniques can help to combat the state space explosion problem for model checking, but are restricted by the hard problem of determining equivalence of states during search. Consequently, existing symmetry reduction packages can only exploit *full* symmetry between system components, as checking the equivalence of states is straightforward in this special case. We present a framework for symmetry reduction with an *arbitrary* group of structural symmetries. By generalising existing techniques for efficiently exploiting symmetry, and introducing an approximate strategy for use with groups for which fast, exact strategies are not available, our approach allows for significant state-space reduction with minimal time overhead. We show how computational group theoretic techniques can be used to analyse the structure of a symmetry group so that an appropriate symmetry reduction strategy can be chosen, and we describe a symmetry reduction package for the Spin model checker which interfaces with the computational algebra system Gap. Experimental results on a variety of Promela models illustrate the effectiveness of our methods.

Keywords: Promela/Spin, model checking, symmetry, computational group theory, Gap.

1 Introduction

Symmetry reduction techniques can help to combat the state space explosion problem when model checking systems with replicated structure. Replication of components in a concurrent system frequently induces replication, or *symmetry*, in a Kripke structure modelling the system, which allows the set of states of the model to be partitioned into equivalence classes. To model check temporal logic properties it is often sufficient to search one state per equivalence class, potentially resulting in more efficient verification. Given a symmetry group G, a common approach to ensure that equivalent states are recognised during search is to convert each newly encountered state s into $min[s]_G$ the *smallest* state in its equivalence class (under a suitable total ordering) before it is stored. However,

[⋆] Supported by the Carnegie Trust for the Universities of Scotland.

the problem of computing $min[s]_G$ for an arbitrary group, called the *constructive orbit problem* (COP), is NP-hard [5].

Existing symmetry reduction packages, such as SymmSpin [1] and SMC [18], are limited as they can only exploit *full* symmetry between identical components of a system. Such symmetry arises in systems where all components of the same type are interchangeable, and has been of primary interest since the COP can be efficiently solved in this special case. However, many other kinds of symmetry commonly occur in models of concurrent systems with a regular structure. For example, cyclic/dihedral groups are typically associated with systems which have uni-/bi-directional ring structures, and wreath product groups occur when dealing with tree topologies.

In this paper we generalise existing techniques for efficiently exploiting symmetry under a simple model of computation, and give an approximate strategy for use with symmetry groups for which fast, exact strategies cannot be found. We use computational group theory to automatically determine the structure of a group before search so that an appropriate symmetry reduction strategy can be chosen, and give encouraging experimental results to support our techniques using TopSpin, a new symmetry reduction package for the Spin model checker [14] which interfaces with the Gap computational algebra system [12]. We then illustrate the problems associated with extending our model of computation to apply to Promela specifications, where full symmetry reduction may no longer be guaranteed.

2 Symmetry in Model Checking

2.1 Model of Computation

We use a simple model to represent the computation of a system comprised of n communicating components, interleaving concurrently [10,11]. Let $I = \{1, 2, \ldots, n\}$ be the set of component identifiers, and for some $k > 0$, let $L = \{1, 2, \ldots, k\}$ denote the possible local states of the components. A Kripke structure is a pair $\mathcal{M} = (S, R)$, where $S \subseteq L^n$, is a non-empty set of states, and $R \subseteq S \times S$ is a total transition relation. The lexicographical ordering of vectors in L^n provides a total ordering on S. If $s = (l_1, l_2, \ldots, l_n) \in S$ then we use $s(i)$ to denote l_i, the local state of component i.

This model of computation allows us to reason about concurrent systems consisting of processes and channels, since a positive integer can be assigned to each valuation of the local variables of a process or the contents of a channel. We assume that the local variables of components do not refer to component identifiers. We discuss the implications of lifting this assumption in Section 5.1.

2.2 Group Theoretic Notation

We assume some knowledge of basic group theory, but recap some notation here. Let G be a group, and let $\alpha_1, \alpha_2, \ldots, \alpha_n \in G$. The smallest subgroup of G containing the elements $\alpha_1, \ldots, \alpha_n$ is denoted $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$, and is called the subgroup *generated* by $\alpha_1, \alpha_2, \ldots, \alpha_n$. The elements α_i $(1 \leq i \leq n)$ are called *generators* for this subgroup. Let $X = \{\alpha_1, \ldots, \alpha_n\}$ be a finite subset of G. Then we use $\langle X \rangle$ to denote $\langle \alpha_1, \ldots, \alpha_n \rangle$, the subgroup generated by X. Let H be a subgroup of G, denoted $H \leq G$, and let $\alpha \in G$. The set $H\alpha = \{\beta\alpha : \beta \in H\}$ is called a (right) *coset* of H in G. The set of all cosets of H in G provides a partition of G into disjoint equivalence classes.

The set of all permutations of I forms a group under composition of mappings, denoted S_n (the symmetric group on n points). If $J \subseteq I$ and $\alpha \in S_n$, then $\alpha(J) = \{\alpha(i) : i \in J\}$. For $\alpha \in S_n$ and $H \leq S_n$, define $moved(\alpha) = \{i \in I : \alpha(i) \neq i\}$, and $moved(H) = \{i \in I : i \in moved(\alpha)$ for some $\alpha \in H\}$. For $i \in I$, the *stabiliser* of i under H is the subgroup $stab_H(i) = \{\alpha \in H : \alpha(i) = i\}$, and the *orbit* of i under H is the set $orb_H(i) = \{\alpha(i) : \alpha \in H\}$. The orbit $orb_H(i)$ is *non-trivial* if $|orb_H(i)| > 1$, and H is said to act *transitively* on I if it induces a single orbit.

2.3 Symmetry Reduction

Let $\mathcal{M} = (S, R)$ be a Kripke structure, and let $\alpha \in S_n$. Then α acts on a state $s = (l_1, l_2, \ldots, l_n) \in S$ in the following way: $\alpha(s) = (l_{\alpha^{-1}(1)}, l_{\alpha^{-1}(2)}, \ldots, l_{\alpha^{-1}(n)})$. If $(\alpha(s), \alpha(t)) \in R \; \forall \, (s,t) \in R$, α is an *automorphism* of $\mathcal{M}$. The set of all automorphisms of $\mathcal{M}$ forms a group $Aut(\mathcal{M}) \leq S_n$ under composition of mappings.

A subgroup $G \leq Aut(\mathcal{M})$ induces an equivalence relation $\equiv_G$ on the states of $\mathcal{M}$ thus: $s \equiv_G t \Leftrightarrow s = \alpha(t)$ for some $\alpha \in G$. The equivalence class under $\equiv_G$ of a state $s \in S$, denoted $[s]_G$, is called the *orbit* of s under the action of G (so G induces orbits on both the set I of component identifiers and the set S of states), and $min[s]_G$ denotes the *smallest* element of $[s]_G$ under the total ordering discussed in Section 2.1. The quotient Kripke structure for $\mathcal{M}$ with respect to G is a pair $\mathcal{M}_G = (S_G, R_G)$ where $S_G = \{min[s]_G : s \in S\}$, and $R_G = \{(min[s]_G, min[t]_G) : (s,t) \in R\}$. In general $\mathcal{M}_G$ is a smaller structure than $\mathcal{M}$, but $\mathcal{M}_G$ and $\mathcal{M}$ are equivalent in the sense that they satisfy the same set of logic properties which are *invariant* under the group G (that is, properties which are "symmetric" with respect to G). For a proof of the following theorem, together with details of the temporal logic CTL^*, see [6].

Theorem 1. *Let $\mathcal{M}$ be a Kripke structure, G a subgroup of $Aut(\mathcal{M})$ and ϕ a CTL^* formula. If ϕ is invariant under G then*

$$\mathcal{M}, s \models \phi \Leftrightarrow \mathcal{M}_G, min[s]_G \models \phi$$

Thus by choosing a suitable symmetry group G, model checking can be performed over $\mathcal{M}_G$ instead of $\mathcal{M}$, often resulting in considerable savings in memory and verification time [2,11]. Algorithm 1, adapted from [15], explores a quotient Kripke structure given an initial state s_0. An extension of this algorithm for on-the-fly model checking of LTL properties using a nested depth first search is presented in [1].

In practice, for an arbitrary group G, it may be infeasible to implement the function min exactly. In such cases the requirements of min can be relaxed so

Algorithm 1. Algorithm to explore a quotient Kripke structure

```
$reached := \{min[s_0]_G\}$;
$unexplored := \{min[s_0]_G\}$;
while $unexplored \neq \emptyset$ do
  remove a state $s$ from $unexplored$;
  for all successor states $t$ of $s$ do
    if $min[t]_G$ is not in $reached$ then
      add $min[t]_G$ to $reached$;
      add $min[t]_G$ to $unexplored$;
    end if
  end for
end while
```

that $min[s]_G$ yields *some* state $t \in [s]_G$ with $t \leq s$. This does not compromise the safety of symmetry reduced model checking since at least one state per orbit is searched, but does not result in memory-optimal verification. However, an efficient implementation of *min* which maps any element s to one of a small number of orbit representatives can result in fast verification, maintaining a significant reduction in model states (this use of *multiple representatives* is employed in e.g. [2,5]). We refer to such an implementation of *min* as an *approximate* symmetry reduction strategy, whereas a true implementation of *min* is an *exact* strategy. Note that exact verification results are still obtained using an approximate symmetry reduction strategy, if enough memory is available.

Throughout the rest of the paper, let G be a subgroup of $Aut(\mathcal{M})$, where M models a concurrent system comprised of n components.

2.4 Symmetry Detection

In this paper, we are concerned with techniques for *exploiting* component symmetries during model checking, rather than *detecting* symmetry before search. Structural symmetries of a model $\mathcal{M}$ are typically inferred by extracting a *communication graph* from the initial specification. The vertex set of this graph is the set I, representing the components of the system. Provided that the specification obeys certain restrictions ensuring that components of the same type are not explicitly distinguished, automorphisms of the communication graph induce automorphisms of $\mathcal{M}$. Since the communication graph is typically small, these automorphisms can be computed automatically using a package such as *saucy* [7]. Practical examples of communication graphs include the *static channel diagram* of a Promela specification [8], and the *coloured hypergraph* [5] of a shared variable concurrent program.

For illustration, throughout the paper we consider a system with a three-tiered architecture consisting of a *database*, a layer of *server* components, and a layer of *client* components, each of which communicates with exactly one server. Figure 1 shows a possible communication graph for this system, which we assume has been extracted from a specification of the system by some symmetry detection tool. Let $\mathcal{M}_{3T}$ be a model of the system. Using the *saucy* program, we compute generators for G_{3T}, the automorphism group of the communication graph:

$$\begin{aligned} G_{3T} = \langle &(1\ 2), (2\ 3), (4\ 5), (5\ 6), (7\ 8), (8\ 9), (10\ 11), \\ &(12\ 13)(1\ 4)(2\ 5)(3\ 6), (13\ 14)(4\ 7)(5\ 8)(6\ 9)\rangle. \end{aligned}$$

Note that the last two elements of the generating set of G_{3T} are products of transpositions. We assume that $G_{3T} \leq Aut(\mathcal{M}_{3T})$, and will use this group and its subgroups as examples to illustrate some of our techniques.

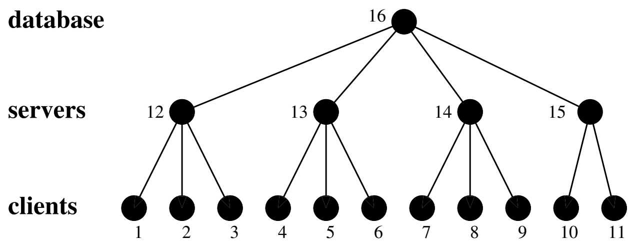

Fig. 1. Communication structure for a three-tiered architecture

3 Exploiting Basic Symmetry Groups

3.1 Enumerating Small Groups

The most obvious strategy for computing $min[s]_G$ is to consider each state in $[s]_G$, and return the smallest. This can be achieved by *enumerating* the elements $\alpha(s)$, $\alpha \in G$. If G is small then this strategy is feasible in practice, and provides an exact symmetry reduction strategy. The SymmSpin package provides an enumeration strategy for fully symmetric groups, which is optimised by generating permutations incrementally by composing successive transpositions. This is more efficient than applying permutations to s directly.

We generalise this optimisation for arbitrary groups using *stabiliser chains*. A stabiliser chain for G is a series of subgroups of the form $G = G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(k)} = \{id\}$, for some $k > 1$, where $G^{(i)} = stab_{G^{(i-1)}}(x)$ for some $x \in moved(G^{(i-1)})$ $(2 \leq i \leq k)$. If $U^{(i)}$ is a set of representatives for the cosets of $G^{(i)}$ in $G^{(i-1)}$ $(2 \leq i \leq k)$, then each element of G can be uniquely expressed as a product $u_{k-1}u_{k-2}\ldots u_1$, where $u_i \in U^{(i)}$ $(1 \leq i < k)$ [3]. Permutations can be generated incrementally using elements from the coset representatives, and the set of images of a state s under G computed using a sequence of partial images (see Algorithm 2). To ensure efficient application of permutations, the coset representatives are stored as a list of transpositions, applied in succession. GAP provides functionality to efficiently compute a stabiliser chain and associated coset representatives for an arbitrary permutation group. Although this approach still involves enumerating the elements $\alpha(s)$ for every $\alpha \in G$ (and is

Algorithm 2. Computing $min[s]_G$ using a stabiliser chain

```
$min[s]_G := s$
**for all** $u_1 \in U_1$ **do**
   $s_1 := u_1(s)$
   **for all** $u_2 \in U_2$ **do**
      $s_2 := u_2(s_1)$
      ⋮
      **for all** $u_k \in U_k$ **do**
         $s_k := u_k(s_{k-1})$
         **if** $s_k < min[s]_G$ **then**
            $min[s]_G := s_k$
         **end if**
      **end for**
      ⋮
   **end for**
**end for**
```

thus infeasible for large groups), calculating each $\alpha(s)$ is faster. The experimental results of Section 5.3 show an improvement over basic enumeration. Additionally, it is only necessary to store coset representatives, rather than all elements of G.

3.2 Minimising Sets for G if $G \cong S_m$ $(m \leq n)$

For systems where there is full symmetry between components, the smallest state in the orbit of $s = (l_1, l_2, \ldots, l_n)$ can be computed by *sorting* the tuple s lexicographically. [2,5]. For example, for a system with four components, sorting equivalent states $(3, 2, 1, 3)$ and $(3, 3, 2, 1)$ yields the state $(1, 2, 3, 3)$, which is clearly the smallest state in the orbit. Since sorting can be performed in polynomial time, this provides an efficient solution for the COP for this group.

Recall the group G_{3T} of automorphisms of the communication graph of Figure 1. Consider the subgroup

$$H = \langle (12\ 13)(1\ 4)(2\ 5)(3\ 6), (13\ 14)(4\ 7)(5\ 8)(6\ 9) \rangle.$$

This group permutes *server* components 12, 13 and 14, with their associated blocks of *client* components. It is clear that H is isomorphic to S_3, the symmetric group on 3 objects. However, we cannot compute $min[s]_H$ by sorting s in the obvious way, since this is equivalent to applying an element $\alpha \in S_{16}$ to s, which may not belong to H.

In some cases we can deal with groups of this form using a *minimising set* for G. Using terminology from [11], G is said to be *nice* if there is a small set $X \subseteq G$ such that, for any $s \in S$, $s = min[s] \Leftrightarrow s \leq \alpha(s)\ \forall\ \alpha \in X$. In this case we call X a *minimising set* for G. If a small minimising set X can be found for a large group G, then computing the representative of a state involves iterating over the small set X, minimising the state until a fix-point is reached. At this point, no element of the minimising set maps the state to a smaller image, thus the minimal element has been found.

We show that, for a large class of groups which are isomorphic to S_m for some $m \leq n$, a minimising set with size polynomial in m can be efficiently found. This minimising set is derived from the swap permutations used in a selection sort algorithm.

Theorem 2. *Suppose that, for each $i \in I$ such that $orb_G(i)$ is non-trivial, $stab_G(i)$ fixes exactly one element from each non-trivial orbit of G acting on I, and that $G \cong S_m$, where $m = |orb_G(i)| > 1$ for some $i \in I$. Then there is an isomorphism $\theta : S_m \to G$ such that $\{(i\ j)^\theta : 1 \leq i < j \leq m\}$ is a minimising set for G.*

Proof. Since for each $i \in I$ such that $|orb_G(i)| > 1$ the set of elements fixed by $stab_G(i)$ contains exactly one element from each orbit, there is a set of *columns* $C_1, C_2, \ldots, C_m$ such that each column contains one element from each orbit of G, and G permutes the columns. There is an isomorphism θ from G' (the action of G on the columns) to G acting on I. Since $G \cong S_m$, G' contains all column transpositions $(i\ j)$ where $i < j$, so $(i\ j)^\theta \in G$. The element $(i\ j)^\theta$ maps all elements of column i to elements of column j.

Now consider states s and s', where $s' = \alpha(s)$ for some $\alpha \in G$. Let i be the smallest index for which $s(i) \neq s'(i)$. Let j be the index such that $j = \alpha^{-1}(i)$. All of the elements in the column containing j (column j' say) are mapped via α to the column containing i (column i' say). Then $s' < s$ iff $(i'\ j')^\theta s < s$. Hence s is minimal in its orbit iff $(i\ j)^\theta(s) \geq s$ for all $i < j$. So the set $\{(i\ j)^\theta : 1 \leq i < j \leq m\}$ is a minimising set for G.

Note that the minimising set is much smaller than G, and the conditions of Theorem 2 can be easily checked using GAP. It may seem that these conditions are unnecessary, and that, given any isomorphism $\theta : S_m \to G$, the set $\{(i\ j)^\theta : 1 \leq i < j \leq m\}$ is a minimising set for G. However, consider the group G below, which is a subgroup of the symmetry group of a hypercube (see Section 5.3).

$$G = \langle (1\ 2)(5\ 6)(9\ 10)(13\ 14), (1\ 2\ 4\ 8)(3\ 6\ 12\ 9)(5\ 10)(7\ 14\ 13\ 11) \rangle \leq S_{14}$$

G is isomorphic to S_4, with an isomorphism $\theta : S_4 \to G$ is defined on generators by $(1\ 2\ 3\ 4)^\theta = (1\ 2\ 4\ 8)(3\ 6\ 12\ 9)(5\ 10)(7\ 14\ 13\ 11)$, $(1\ 2)^\theta = (4\ 8)(5\ 9)(6\ 10)(7\ 11)$. The state $s = (6, 10, 3, 6, 3, 5, 7, 10, 4, 8, 2, 1, 9, 3) \in \{1, 2, \ldots, 10\}^{14}$ *cannot* be minimised using the set $\{(i\ j)^\theta : 1 \leq i < j \leq 4\}$.

3.3 Local Search for Unclassifiable Groups

If G is large group then computing $min[s]_G$ by enumeration of the elements of G may be infeasible, even with the group-theoretic optimisations discussed in Section 3.1. If no minimising set is available for G, and G cannot be classified as a composite symmetry group (see Section 4) then we must exploit G via an approximate symmetry reduction strategy.

We propose an approximate strategy based on *hillclimbing local search*, which has proved successful for a variety of search problems in artificial intelligence [17, Chapter 4]. In this case the function *min* works by performing a local search of $[s]_G$ starting at s, using the generators of G as operations from which to compute a successor state. The search starts by setting $t = s$, and proceeds iteratively. On each iteration, $\alpha(t)$ is computed for each generator α of G. If $t \leq \alpha(t)$ for all α then a local minimum has been reached, and t is returned as a representative for $[s]_G$. Otherwise, t is set to the smallest image $\alpha(t)$, and the search continues. In Section 5.3 we show that this local search algorithm is effective when exploring the state spaces of various configurations of message routing in a hypercube network.

There are various local search techniques which could be employed to attempt to improve the accuracy of this strategy. *Random-restart* hill-climbing [17] involves the selection of several random elements of $[s]_G$ in addition to s, and performing local search from each of them, returning the smallest result. In our case we could apply such a technique by finding the image of a state s under distinct, random elements of G (Gap provides functionality for generating random group elements). Another potential improvement would be to use *simulated annealing* [16] to escape local minima.

4 Exploiting Composite Symmetry Groups

Certain kinds of symmetry groups can be decomposed as a product of subgroups. In this case solving the COP separately for each subgroup provides a solution to the COP for the whole group. In particular, if a symmetry group permutes disjoint sets of components independently then the group can be described as the *disjoint product* of the groups acting on these disjoint sets. On the other hand, if the symmetry group partitions the components into subsets such that there is analogous symmetry within each subset, and symmetry between the subsets, then the group can be described as the *wreath product* of the group which acts on one of the subsets, and the group which permutes the subsets. It has been shown that, if G is known to be a disjoint or wreath product of subgroups, then the COP can be solved for G by restricting attention to these subgroups [5]. We now present solutions to the problem of detecting, before search, whether or not G can be decomposed.

4.1 Disjoint Products

Definition 1. *Let $H \leq S_n$. Suppose that $H_1, H_2, \ldots, H_k$ are subgroups of H ($1 \leq i \leq k$, $k > 1$). If $H = H_1 H_2 \ldots H_k = \{\alpha_1 \alpha_2 \ldots \alpha_k : \alpha \in H_i \ (1 \leq i \leq k)\}$ then H is called the product of the H_i. If $moved(H_i) \cap moved(H_j) = \emptyset$ for all $1 \leq i \neq j \leq k$ then H is written $H_1 \bullet H_2 \bullet \cdots \bullet H_k$, and called the* disjoint product *of the H_i. The disjoint product is said to be non-trivial if $H \neq H_i \neq \{id\}$ for all $1 \leq i \leq k$.*

Disjoint products occur frequently in model checking problems. For example, the symmetry group associated with a model of the *readers writers* problem [10] may be a disjoint product of two groups, which independently permute *reader* and *writer* components respectively. In our three-tiered architecture example (see Section 2.4), the group G_{3T} can be shown to decompose as a disjoint product $G_{3T} = H_1 \bullet H_2$ where:

$$\begin{aligned} H_1 &= \langle (1\ 2), (2\ 3), (4\ 5), (5\ 6), (7\ 8), (8\ 9), \\ &\quad (12\ 13)(1\ 4)(2\ 5)(3\ 6), (13\ 14)(4\ 7)(5\ 8)(6\ 9) \rangle \\ H_2 &= \langle (10, 11) \rangle. \end{aligned}$$

If G is a disjoint product of subgroups $H_1, H_2, \ldots, H_k$ then $min[s]_G = min[\ldots min[min[s]_{H_1}]_{H_2} \ldots]_{H_k}$ [5], so the COP for G can be solved by considering each subgroup H_i in turn. This result is only useful when designing a fully automatic symmetry reduction package if it is possible to efficiently determine, before search, whether or not G decomposes as a disjoint product of subgroups. We present two solutions to this problem.

Efficient, Sound, Incomplete Approach

Let $G = \langle X \rangle$ for some $X \subseteq G$ with $id \notin X$. Define a binary relation $B \subseteq X^2$ as follows: for all $\alpha, \beta \in X$, $(\alpha, \beta) \in B \Leftrightarrow moved(\alpha) \cap moved(\beta) \neq \emptyset$. Clearly B is symmetric, and since for any $\alpha \in G$ with $\alpha \neq id$, $moved(\alpha) \neq \emptyset$, B is reflexive. It follows that the transitive closure of B, denoted B^*, is an equivalence relation on X. We now show that if B^* has multiple equivalence classes then each class generates a subgroup of G which is a non-trivial factor for a disjoint product decomposition of G.

Lemma 1. *Suppose that $\alpha, \beta \in X$, and that $(\alpha, \beta) \notin B^*$. Then $moved(\alpha) \cap moved(\beta) = \emptyset$ and α and β commute.*

Theorem 3. *Suppose $C_1, C_2, \ldots, C_k$ are the equivalence classes of X under B^* where $k \geq 2$. For $1 \leq i \leq k$ let $H_i = \langle C_i \rangle$. Then $G = H_1 \bullet H_2 \bullet \cdots \bullet H_k$, and $H_i \neq \{id\}$ $(1 \leq i \leq k)$.*

Proof. Clearly $H_1 H_2 \ldots H_k \subseteq G$. If $\alpha \in G$ then $\alpha = \alpha_1 \alpha_2 \ldots \alpha_d$ for some $\alpha_1, \alpha_2, \ldots, \alpha_d \in X$, $d > 0$. By Lemma 1 we can arrange the α_l so that elements of C_i appear before those of C_j whenever $i < j$. It follows that $G = H_1 H_2 \ldots H_k$. By Lemma 1, $moved(H_i) \cap moved(H_j) = \emptyset$ for $1 \leq i \neq j \leq k$ and so $G = H_1 \bullet H_2 \bullet \cdots \bullet H_k$, where (since $id \notin X$) the H_i are non-trivial.

The approach is incomplete as it does not guarantee the finest decomposition of G as a disjoint product. However, in practice we have not found a case in which the finest decomposition is not detected when generators have been computed by a graph automorphism program. The approach is very efficient as it works purely with the generators of G, of which there are typically few.

Sound and Complete Approach
It is straightforward to show that if $G = H \bullet K$, then H and K are *normal* subgroups of G. Thus a complete method for determining whether or not G is a non-trivial disjoint product of subgroups H and K involves the computation of all normal subgroups of G and searching for a pair such that $G = H \bullet K$. This method could be applied recursively to the factors of the disjoint product to compute the finest disjoint product decomposition of G. Although for certain groups (e.g. abelian groups), all subgroups are normal, in many cases the number of normal subgroups of a group is significantly smaller than the number of arbitrary subgroups.

4.2 Wreath Products

Definition 2. *For $r > 1$ let $B_1, B_2, \ldots, B_r$ be disjoint subsets of I, where $B_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,m}\}$ for some $m > 1$. Let $H \leq S_n$ with moved$(H) \subseteq B_1$. For any $\alpha \in H$ and $1 \leq i \leq r$, define $\alpha^{(i)}$ by: $\alpha^{(i)}(x) = x$ if $x \notin B_i$; $\alpha^{(i)}(b_{i,j}) = b_{i,l}$ where $\alpha(b_{1,j}) = b_{1,l}$. For $\beta' \in S_r$, define $\beta \in S_n$ by: $\beta(x) = x$ if $x \notin \bigcup_{1 \leq i \leq r} B_i$ and $\beta(b_{i,j}) = b_{\beta'(i),j}$. Let $K = \langle \beta_1, \beta_2, \ldots, \beta_d \rangle$ where $d > 0$ and each β_i is derived from some $\beta'_i \in S_r$). If every element of G can be expressed in the form $\beta\alpha_r^{(r)} \ldots \alpha_2^{(2)}\alpha_1^{(1)}$, where $\beta \in K$ and $\alpha_1, \alpha_2, \ldots, \alpha_r \in H$, then G is the* wreath product *of H and K, denoted $H \wr K$.*

Intuitively, an element of G applies a permutation to each set B_i, then applies a permutation which permutes the sets. This definition of wreath products is specific to those that occur in model checking problems, typically when a system has a tree structure. For a more general definition, see [4]. In Section 4.1, we showed that the group G_{3T} decomposes as a disjoint product. Consider the factor H_1 of this product. This group itself decomposes as a wreath product $H_1 = H \wr K$ where:

$$\begin{aligned} H &= \langle (1\ 2), (2\ 3) \rangle \\ K &= \langle (12\ 13)(1\ 4)(2\ 5)(3\ 6), (13\ 14)(4\ 7)(5\ 8)(6\ 9)(10, 11) \rangle. \end{aligned}$$

Here, the sets are $B_1 = \{1, 2, 3, 12\}$, $B_2 = \{4, 5, 6, 13\}$ and $B_3 = \{7, 8, 9, 14\}$. If $G = H \wr K$ then, for $1 \leq i \leq k$ define H_i by $\{\alpha^{(i)} : \alpha \in K$. Then $min[s]_G = min[min[\ldots min[min[s]_{H_1}]_{H_2} \cdots]_{H_r}]_K$ [5].

We sketch an approach for detecting whether an arbitrary group is a wreath product of subgroups. If G acts transitively on I, a subset B of I is a *block* if, for any $\alpha \in G$, $\alpha(B) = B$ or $B \cap \alpha(B) = \emptyset$. The set $\mathcal{B} = \{\alpha(B) : \alpha \in G\}$ is a *block system* for G. Given block systems $\mathcal{B}, \mathcal{C}$ for G, $\mathcal{C}$ is *strictly coarser* than $\mathcal{B}$ if $\forall\ B \in \mathcal{B}\ \exists\ C \in \mathcal{C}$ such that $B \subset C$, and $\mathcal{B}$ is *maximal* for G if each $B \in \mathcal{B}$ is a proper subset of I, and the only block system strictly coarser than $\mathcal{B}$ is the trivial system $\{I\}$.

If $\{B_1, B_2, \ldots, B_r\}$ is a block system for a transitive group G, where $B_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,m}\}$ then $G \leq H \wr K$, where $H = \bigcap_{i \notin B_1} stab_G(i)$, and $K = \bigcap_{1 \leq i \leq m} stab_G(\{b_{1,i}, b_{2,i}, \ldots, b_{r,i}\})$ [4]. To check whether or not $G = H \wr K$ it is sufficient to compare orders, and it can be shown that $|H \wr K| = |H|^r|K|$. However, in general G does not act transitively on I. We solve the general problem of determining whether or not G is a wreath product of subgroups by considering the action of G separately on each non-trivial orbit of I.

Lemma 2. *If $G = H \wr K$ then each non-trivial orbit O of I under G has a single maximal block system: $\{O \cap moved(H_i) : 1 \leq i \leq r)\}$.*

If G can be shown to have exactly one maximal block system per orbit, then candidate groups H and K can be constructed. Suppose the non-trivial orbits are $O_1, O_2, \ldots, O_d$. For $1 \leq i \leq d$, the group K_i is computed as follows: let $\{B_1, B_2, \ldots, B_u\}$ be the maximal block system for O_i, where each B_j has the form $\{b_{j,1}, b_{j,2}, \ldots, b_{j,v}\}$ (for some $u, v > 0$). Then $K_i = \bigcap_{1 \leq l \leq v} stab_G(\{b_{1,l}, b_{2,l}, \ldots, b_{u,l}\})$. The candidate group K is the intersection of the K_i. Candidate group H is initially set to G. For each orbit O_i a maximal block B is chosen such that $B \subseteq moved(H)$. Then H is recomputed as $\bigcap_{j \in O_i \setminus B} stab_H(j)$. We now have groups H and K with $G \leq H \wr K$, and we can check whether $G = H \wr K$ by comparing orders, as before.

4.3 Choosing a Strategy for *G*

The strategies which we have presented for minimising a state with respect to basic and composite groups can be combined to yield a symmetry reduction strategy for the arbitrary group G by classifying the group using a top-down recursive algorithm.

The algorithm starts by searching for a minimising set for G of the form prescribed in Theorem 2, so that $min[s]_G$ can be computed as described in Section 3.2. If no such minimising set can be found, a decomposition of G as a disjoint/wreath product is sought. In this case the algorithm is applied recursively to obtain a minimisation strategy for each factor of the product so that $min[s]_G$ can be computed using these strategies as described in Sections 4.1 and 4.2 respectively. If G remains unclassified and $|G|$ is sufficiently small, enumeration is used, otherwise local search is selected.

5 Symmetry Reductions in Practice

5.1 Extending the Model of Computation

When components do not hold references to other components, the simple model of computation and the action of a permutation on a state (described in Sections 2.1 and 2.3 respectively) are sufficient to reason about concurrent systems, since it is always possible to represent the local state of a component using an integer. However, if components *can* hold references to one another then any permutation that moves component i will also affect the local state of any components which refer to component i.

Sophisticated specification languages, such as Promela, include special datatypes to represent process and channel identifiers. An extended model of computation for Promela is presented in [8]. Both the results presented in [5] on solving the COP for groups which decompose as disjoint/wreath products, and our results on minimising sets for fully symmetric groups (see Section 3.2) do not hold in general for this extended model of computation.

Thus for Promela specifications where local variables refer to process and channel identifiers, the efficient symmetry reduction strategies presented above are not always exact—in some cases they may yield an *approximate* implementation of the function *min*, as discussed in Section 2.3. This does not compromise the safety of symmetry reduced model checking, and in any case, for a large model, there will be many states for which the strategies *will* give exact representatives in an extended model of computation as the experimental results in Section 5.3 show.

For the simple case of full symmetry between identical components, the SymmSpin package deals with local variables which are references to component identifiers by dividing the local state of each component into two portions, one which does not refer to other components (the *insensitive* portion say), and another which consists entirely of such references (the *sensitive* portion). A state is minimised by first sorting it with respect to the insensitive portion. Then, for each subset of components with identical insensitive portions, every permutation of the subset is considered, and the permutation which leads to the smallest image is applied. This is known as the *segmented* strategy. Our approach using minimising sets is similar to the *sorted* strategy which SymmSpin also provides. Here a state is minimised only with respect to the insensitive portions of the local states. This strategy is much faster than the segmented strategy, but is approximate. It may be possible to extend our approach to be exact by generalising the segmented strategy.

5.2 A Symmetry Reduction Package for Spin

We have implemented the strategies discussed in Sections 3 and 4 as TopSpin, a fully automatic symmetry reduction package for Spin [9]. In order to check properties of a Promela specification, Spin first converts the specification into a C source file, `pan.c`, which is then compiled into an executable verifier. The state space thus generated is then searched. If the property being checked is proved to be false, a counterexample is given. TopSpin follows the approach used by the SymmSpin symmetry reduction package, where `pan.c` is generated as usual by Spin, and then converted to a new file, `sympan.c`, which includes algorithms for symmetry reduction. With TopSpin, because we allow for arbitrary system topologies, symmetry must be detected before `sympan.c` can be generated. This is illustrated in Figure 2.

First, the *static channel diagram* (SCD) of the Promela specification is extracted by the SymmExtractor tool, which is described in detail in [8]. The SCD is a graphical representation of potential communication between components of the specification. The group of symmetries of the SCD, $Aut(SCD)$, is computed using the *saucy* tool [7], which we have extended to handle directed graphs. The generators of $Aut(SCD)$ are checked against the Promela specification for validity (an assurance that they induce symmetries of the underlying state space).

TopSpin uses Gap to compute, from the set of valid generators, the largest group $G \leq Aut(SCD)$ which can be safely used for symmetry-reduced model checking. Gap is then used to classify the structure of G in order to choose an efficient symmetry reduction strategy. The chosen strategy is merged with `pan.c` to form `sympan.c`, which can be compiled and executed as usual. In order to compare strategies it is possible to manually select the strategy used (rather than let TopSpin choose the most efficient). For experimental purposes, TopSpin also allows generators of an arbitrary group of component symmetries to be specified manually, as long as the group elements do not permute components with different types.

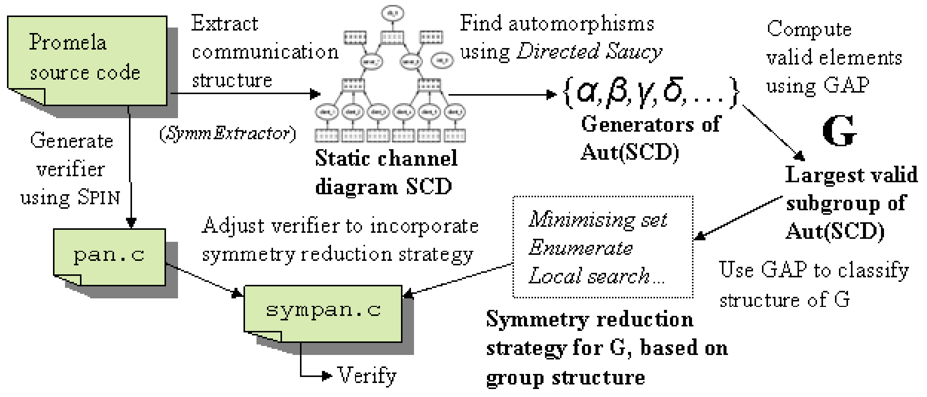
Promela source code
Extract communication structure
(SymmExtractor)
Static channel diagram SCD
Find automorphisms using Directed Saucy
$\{\alpha,\beta,\gamma,\delta,\ldots\}$
Generators of Aut(SCD)
Compute valid elements using GAP
G
Largest valid subgroup of Aut(SCD)
Use GAP to classify structure of G
Minimising set
Enumerate
Local search...
Symmetry reduction strategy for G, based on group structure
Generate verifier using SPIN
pan.c
Adjust verifier to incorporate symmetry reduction strategy
sympan.c
Verify

Fig. 2. The symmetry reduction process

5.3 Experimental Results

In Table 1 we present experimental results applying TopSpin to four families of Promela specifications. For each specification, we give the number of model states without symmetry reduction (**orig**), with full symmetry reduction (**red**), and using the strategy chosen by TopSpin (**best**). If the latter two are equal, '=' appears for the TopSpin strategy. The use of state compression, provided by Spin, is indicated by the number of states in italics. For each strategy (**basic** for enumeration without the optimisations described in Section 3.1, **enum** for optimised enumeration, and **best** for the strategy chosen by TopSpin), and when symmetry reduction is not applied (**orig**), we give the time taken for verification (in seconds). Verification attempts which exceed available resources, or do not terminate within 5 hours, are indicated by '-'. All experiments are performed on a PC with a 2.4GHz Intel Xeon processor, 3Gb of available main memory, running Spin version 4.2.3.

The first family of specifications model flow of control for a system similar to the three-tiered architecture example of Section 2.4, but with a layer of p servers with q clients connected to each server (a D-S-C system). Here models exhibit wreath product symmetry: there is full symmetry between the q clients in each block, and the blocks of clients, with their associated servers, are interchangeable. A configuration with p servers and q clients per server is denoted p/q. The second family of specifications model a resource allocator process which controls access to a resource by a competing set of prioritised clients (an R-C system). Models of these specifications exhibit disjoint product symmetry: there is full symmetry between each set of clients with the same priority level. A configuration with p_i clients of priority level i is denoted $p_1, \ldots, p_k$, where k is the number of priority levels. The next family consists of specifications which model an email system where *client* processes exchange messages via a *mailer* process. The symmetries of models of these specifications permute the client processes, simultaneously permuting their input channels, and can be handled using a minimising set. A configuration with p clients is simply denoted p. Finally, we consider specifications modelling message routing in an n dimensional hypercube network (an HC system). The symmetry group here is isomorphic to the group of geometrical symmetries of a n dimensional hypercube, which cannot be decomposed as a product of subgroups, and thus must be handled using either the *enumeration* or *local search* strategies. An n-dimensional hypercube specification is denoted nd. For all specifications, we verify deadlock freedom, and check the satisfaction of basic safety properties expressed using assertions.

In all cases, basic enumeration is significantly slower optimised enumeration, which is in turn slower than the strategies chosen by TopSpin. For the three-tiered and resource allocator configurations the strategies chosen by TopSpin, which

Table 1. Experimental results for various configurations of the three-tiered (D-S-C), resource allocator (R-C), email (email) and hypercube (HC) specifications

system	config.	states orig	time orig	$\lvert G\rvert$	states red	time basic	time enum	states best	time best
D-S-C	2/3	103105	5	72	2656	7	4	=	2
D-S-C	2/4	1.1×10^6	37	1152	5012	276	108	=	2
D-S-C	3/3	*$2.54{\times}10^7$*	4156	1296	50396	4228	1689	=	19
D-S-C	3/4	-	-	82944	-	-	-	130348	104
R-C	3,3	16768	0.2	36	1501	0.9	0.3	=	0.1
R-C	4,4	199018	2	576	3826	57	19	=	0.4
R-C	5,5	2.2×10^6	42	14400	8212	4358	1234	=	2
R-C	4,4,4	2.39×10^7	1587	13824	84377	-	12029	=	17
R-C	5,5,5	-	-	1728000	-	-	-	254091	115
email	3	23256	0.1	6	3902	0.9	0.8	3908	0.2
email	4	852641	9	24	36255	13	6	38560	2
email	5	*$3.04{\times}10^7$*	3576	120	265315	679	253	315323	40
email	6	-	-	720	1.7×10^6	-	13523	2.3×10^6	576
email	7	-	-	5040	-	-	-	1.53×10^7	6573
HC	3d	13181	0.3	48	308	0.6	0.3	468	0.2
HC	4d	380537	18	384	1240	58	34	6986	13
HC	5d	*$9.6{\times}10^6$*	2965	3840	3907	7442	5241	90442	946

decompose the symmetry group as a wreath/disjoint product of groups which are then handled by minimising sets, provide exact symmetry reduction, despite the potential problems discussed in Section 5.1. This is not the case for email configurations, for which TopSpin uses minimising sets. Nevertheless, a large factor of reduction is still obtained, and verification is fast. For hypercube configurations, TopSpin chooses local search, requiring more states than enumeration, but still resulting in a greatly reduced state space.

6 Related Work

The simple model of computation which we have used throughout the paper is common to numerous works on symmetry reductions for model checking [5,10,11], and is adequate for reasoning about input languages where components do not individually hold references to other components, e.g. the input languages of SMC [18] and `SYMM` [5], or where components are specified using *synchronisation skeletons* [10]. The problem of extending symmetry reduction techniques to a model of computation where such references are allowed is tackled, for the simple case of full symmetry between identical components, by the *segmented* strategy of the SymmSpin package.

Methods for exploiting the disjoint/wreath product structure of symmetry groups were proposed in [5], but this work did not investigate the problem of classifying the structure of an arbitrary group, as we have done. Stabiliser chains (see Section 3.1) are used extensively in computational group theory [3,12], and have been utilised in symmetry breaking approaches for constraint programming [13]. This paper is, to our knowledge, the first to apply these techniques to model checking. The construction of minimising sets for fully symmetric groups which we presented in Section 3.2 builds on the concept of a *nice* group [11], and generalises the idea of computing orbit representatives by sorting [2,5,11].

7 Conclusions and Future Work

In this paper, we have proposed exact and approximate strategies for tackling the NP-hard problem of computing orbit representatives in order to exploit symmetry when model checking concurrent systems, and we have generalised existing results in this area. We have applied techniques from computational group theory to speed up representative computation, and to classify the structure of a symmetry group as a disjoint/wreath product of subgroups before search. We have described TopSpin, a fully automatic symmetry reduction package for Spin, and presented encouraging experimental results for a variety of Promela examples.

We are currently investigating further the use of local search techniques as an approximate symmetry reduction strategy. We are also developing an approach to generalise the *segmented* strategy used by the SymmSpin package to overcome potential inefficiencies associated with extending our simple model of computation to the Promela language. TopSpin is currently limited to verifying the absence of deadlock and the satisfaction of safety properties of Promela specifications. Future work includes extending TopSpin to allow symmetry-reduced verification of temporal properties with weak fairness, as described in [1].

References

1. D. Bosnacki. A light-weight algorithm for model checking with symmetry reduction and weak fairness. In *SPIN'03*, LNCS 2648, pages 89–103. Springer, 2003.
2. D. Bosnacki, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):65–80, 2002.
3. G. Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *LNCS*. Springer-Verlag, 1991.
4. P.J. Cameron. *Permutation Groups*. Cambridge University Press, 1999.
5. E.M. Clarke, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model checking. In *CAV'98*, LNCS 1427, pages 147–158. Springer, 1998.
6. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.
7. P.T. Darga, M.H. Liffiton, K.A. Sakallah, and I.L. Markov. Exploiting structure in symmetry detection for CNF. In *DAC'04*, pages 530–534. ACM Press, 2004.
8. A.F. Donaldson and A. Miller. Automatic symmetry detection for model checking using computational group theory. In *FM'05*, LNCS 3582, pages 418–496. Springer, 2005.
9. A.F. Donaldson and A. Miller. A computational group theoretic symmetry reduction package for the Spin model checker. In *AMAST'06*, LNCS 4019, pages 374–380. Springer, 2006.
10. E.A. Emerson and R.J. Trefler. From asymmetry to full symmetry: new techniques for symmetry reduction in model checking. In *CHARME'99*, LNCS 1703, pages 142–156. Springer, 1999.
11. E.A. Emerson and T. Wahl. Dynamic symmetry reduction. In *TACAS'05*, LNCS 3440, pages 382–396. Springer, 2005.
12. The Gap Group. *GAP–Groups Algorithms and Programming, Version 4.2*. Aachen, St. Andrews, 1999. http://www-gap.dcs.st-and.ac.uk/~gap.
13. I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: symmetry breaking during search. In *CP'02*, LNCS 2470, pages 415–430. Springer, 2002.
14. G. J. Holzmann. *The SPIN model checker: primer and reference manual.* Addison Wesley, 2003.
15. C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2): 41–75, 1996.
16. K.S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–680, 1983.
17. S. Russel and P. Norvig. *Artificial Intelligence, a Modern Approach.* Prentice Hall, 1995.
18. A.P. Sistla, V. Gyuris, and E.A. Emerson. SMC: a symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology*, 9(2):113–166, 2000.