Extending Symmetry Reduction Techniques to a Realistic Model of Computation

Alastair F. Donaldson, Alice Miller

Extending Symmetry Reduction Techniques to a Realistic Model of Computation

Alastair F. Donaldson[1],[2] Alice Miller[3]

Department of Computing Science
University of Glasgow
Glasgow, Scotland.

Abstract

Much of the literature on symmetry reductions for model checking assumes a simple model of computation where the local state of each component in a concurrent system can be represented by an integer, and where components do not hold references to one another. Symmetry reduction techniques for model checking usually require a solution to the *NP*-hard *Constructive Orbit Problem* (COP)–computing the minimum element in the equivalence class of a given state under a symmetry group. Polynomial time strategies to solve instances of the COP under the simple model of computation are known for a large class of symmetry groups. We show that these strategies are not directly applicable when the model of computation is extended to allow components to hold references to one another, and present an approach to their extension, resulting in tractable, memory optimal symmetry reduction techniques for a realistic model of computation. Experimental results using the TopSpin symmetry reduction package for the Spin model checker illustrate the effectiveness of our techniques.

Keywords: Model checking, symmetry, computational group theory, concurrency, Promela/Spin, distributed systems, Gap

1 Introduction

Over the last decade there has been a lot of interest in using symmetry reduction techniques to combat the state space explosion problem for model checking. Symmetry reduction techniques exploit the fact that a concurrent system often has replicated structure, in which case temporal properties of a model of the system can be checked over a *quotient* state space, thus avoiding redundant checking of equivalent behaviours induced by the replication. Symmetries of a model are typically induced by a group of component identifier permutations, which give rise to automorphisms of the state space when lifted to states. The standard approach to exploiting symmetry in explicit state model checking involves converting each state encountered during search to a unique representative in its equivalence class

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

[2] Email: ally@dcs.gla.ac.uk

[3] Email: alice@dcs.gla.ac.uk

A revised version of this paper will be electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

before storing it. Then when an equivalent state is reached, it will be converted to the same representative (resulting in a transition to a previously reached state) and search can backtrack.

The standard approach to computing equivalence class representatives is, given a total ordering $\sqsubseteq$ on states, to take $rep(s) = min_{\sqsubseteq}[s]_G$, where s is a state and $[s]_G$ is the equivalence class, or *orbit* of s under the group G of symmetries. The problem of computing $min_{\sqsubseteq}[s]_G$ under a simple model of computation, where the local state of each component is an integer value, components do not hold references to one another and $\sqsubseteq$ is the usual lexicographic ordering on vectors, is called the *constructive orbit problem* (COP), and is NP-hard [4]. For certain classes of symmetry groups, including fully symmetric groups and groups which can be decomposed as a disjoint/wreath product of subgroups, the COP can be solved in polynomial time [4,8]. However, rich specification languages such as Promela [11] allow components of a system to hold references to one another, and realistic models of software systems depend on this feature.

We present the Constructive Orbit Problem *with References* (COPR), and show that polynomial time strategies proposed in [4,8] for COP under the simple model of computation used in e.g. [4,9] do not directly extend to solve COPR. We then present a computational group theoretic approach which extends any strategy for solving the COP to a solution for COPR. This extension is based on the *segmented* strategy used by the SymmSpin package for the Spin model checker [1], which is a special case of our approach for full symmetry groups. Although our extension results in exact symmetry reduction at the expense of losing a polynomial time solution, experimental results using the TopSpin package for the Spin model checker [7] with various configurations of two Promela examples demonstrate that in practice our approach is significantly more efficient than enumerating $[s]_G$ to compute the minimum. We show that COPR is polynomial time equivalent to COP, and discuss the relationship between these problems and the computational group theoretic problem of finding the smallest image of a set under a group [14].

2 Models of Computation

We use *component* to refer to a process, channel or shared variable in a concurrent system. Let $I = \{1, 2, \ldots, n\}$ be the set of component identifiers for such a system. Suppose that the local state of a component is comprised of two parts, its *control* state and its *reference* state.

The control state of a component is determined by the values of all local variables of that component which are *not* references to other components, e.g. a program counter or boolean flag. Without loss of generality, we can represent a local control state abstractly as an integer in the set $L_c = \{0, 1, 2, \ldots, k\}$ for some $k \geq 0$.

On the other hand, the reference state of a component is determined by the values of all local variables which *are* references to other components. For example, components in a leader election protocol may require a reference variable to (eventually) hold the identity of the leader; a user in a model of telephony may hold a reference to its current partner. Thus a reference state is a tuple in the set $L_r = (I \cup \{0\})^m$ for some $m \geq 0$. Here m is the number of references held by a component, and 0 is used as a default value (e.g. to represent that the leader is unknown). Without loss of generality we can assume that all components have exactly $m \geq 0$ reference local variables.

Thus a global state $s \in (L_c \times L_r)^n$ has the form:

$$s = (l_1, (r_{1,1}, r_{1,2}, \ldots, r_{1,m}), l_2, (r_{2,1}, r_{2,2}, \ldots, r_{2,m}), \ldots, l_n, (r_{n,1}, r_{n,2}, \ldots, r_{n,m})),$$

where $l_i \in L_c$ represents the control state of component i, and $r_{i,j} \in I \cup \{0\}$ is the value of the jth reference variable of component i ($i \in I, 1 \leq j \leq m$).

In the special case where $m = 0$, i.e. when components do not hold references to one another, L_r consists of a 0-tuple, and can thus be ignored. A state $s \in L_c^n$ then has the form $s = (l_1, l_2, \ldots, l_n)$. We refer to models of computation where $m > 0$ and $m = 0$ as a model of computation *with* and *without* references, respectively.

A Kripke structure is a pair $\mathcal{M} = (S, R)$, where $S \subseteq (L_c \times L_r)^n$ is a non-empty set of states, and $R \subseteq S \times S$ is a total transition relation. A Kripke structure expresses the semantics of the specification of a concurrent system written in a high level language such as Promela. Statements of the specification determine which transitions are possible from a given state of the Kripke structure, and the complete structure can be constructed by following all paths from some initial state.

3 Symmetry Reduction in Model Checking

3.1 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. 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 $\alpha(J) = \{\alpha(i) : i \in J\}$, and the *set stabiliser* of J in G is the subgroup $stab_G(J) = \{\alpha \in G : \alpha(J) = J\}$. If H is a subgroup of G we write $H \leq G$ (note that we also use $\leq$ to denote lexicographic ordering of vectors).

3.2 Automorphisms and Quotient Structures

In a model of computation without references, $\alpha \in S_n$ acts on $s = (l_1, l_2, \ldots, l_n) \in L_c^n$ by simply permuting control states: $\alpha(s) = (l_{\alpha^{-1}(1)}, l_{\alpha^{-1}(2)}, \ldots, l_{\alpha^{-1}(n)})$.

Now let $s \in (L_c \times L_r)^n$ be as in Section 2, and $\alpha \in S_n$. The application of α to s can be considered as a two-stage process. For each i, first the local state $(l_i, (r_{i,1}, r_{i,2}, \ldots, r_{i,m}))$ of component i is replaced by the local state of component $\alpha^{-1}(i)$. Then α is applied to each reference variable of component i (with $\alpha(0) = 0$). Thus:

$$\begin{aligned}\alpha(s) = (\ & l_{\alpha^{-1}(1)}, (\alpha(r_{\alpha^{-1}(1),1}), \alpha(r_{\alpha^{-1}(1),2}), \dots, \alpha(r_{\alpha^{-1}(1),m})),\\ & l_{\alpha^{-1}(2)}, (\alpha(r_{\alpha^{-1}(2),1}), \alpha(r_{\alpha^{-1}(2),2}), \dots, \alpha(r_{\alpha^{-1}(2),m})), \dots,\\ & l_{\alpha^{-1}(n)}, (\alpha(r_{\alpha^{-1}(n),1}), \alpha(r_{\alpha^{-1}(n),2}), \dots, \alpha(r_{\alpha^{-1}(n),m}))\ ).\end{aligned}$$

For example, consider a system comprised of 4 components, where the state of each component consists of its control state and one reference variable. In this case $n = 4$, $m = 1$ and $L_r = \{0,1,2,3,4\}^1 = \{0,1,2,3,4\}$. Suppose that $L_c = \{0,1,2\}$. Let $s = (\mathbf{1},4,\mathbf{2},3,\mathbf{0},0,\mathbf{0},4) \in (L_c \times L_r)^4$ (control states are distinguished by bold type). If $\alpha = (3\ 4)$ then $\alpha(s) = (\mathbf{1},\alpha(4),\mathbf{2},\alpha(3),\mathbf{0},\alpha(4),\mathbf{0},\alpha(0)) = (\mathbf{1},3,\mathbf{2},4,\mathbf{0},3,\mathbf{0},0)$.

A permutation $\alpha \in S_n$ is an *automorphism* of a Kripke structure $\mathcal{M} = (S, R)$ if α preserves the transition relation R. That is, for each transition $(s,t) \in R$, $(\alpha(s), \alpha(t)) \in R$. The set of all automorphisms of $\mathcal{M}$ forms a group under composition of mappings, denoted $Aut(\mathcal{M})$. If G is a subgroup of $Aut(\mathcal{M})$ then G induces an equivalence relation on S, where the equivalence class or *orbit* of $s \in S$ is the set $[s]_G = \{\alpha(s) : \alpha \in G\}$. If *rep* is a function which maps every state to a unique representative from its equivalence class, then the *quotient* Kripke structure of $\mathcal{M}$ by G can be defined as follows: $\mathcal{M}_G = (S_G, R_G)$ where $S_G = \{rep(s) : s \in S\}$, $R_G = \{(rep(s), rep(t)) : (s,t) \in R\}$. 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 [5].

Theorem 3.1 *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*

$$\mathcal{M}, s \models \phi \ \textit{ iff } \ \mathcal{M}_G, rep(s) \models \phi$$

3.3 Constructive Orbit Problems

Let $\sqsubseteq$ be a total ordering on S. Then for any $s \in S$, $rep(s)$ can be taken as $min_{\sqsubseteq}[s]_G$, the smallest element of $[s]_G$ with respect to the ordering $\sqsubseteq$. If $\mathcal{M} = (S, R)$ has initial state s_0, Algorithm 1 (adapted from [12]) can be used to explore $\mathcal{M}_G$. The efficiency of the algorithm depends on the complexity of computing $min_{\sqsubseteq}[s]_G$. Assuming a model of computation without references, so that $S \subseteq L_c^n$, we can take $\sqsubseteq$ to be $\leq$, the usual lexicographic ordering on vectors.[4] Then we have

Definition 3.2 The Constructive Orbit Problem (COP) [4] Given a group $G \leq S_n$ and a state $s \in L_c^n$, find $min_{\leq}[s]_G$, the lexicographically least element in the orbit of s under G.

Theorem 3.3 [4] *The COP is NP-hard.*

Despite this discouraging result, for a large class of commonly occurring symmetry groups, it is possible to solve the COP in polynomial time. We outline these

[4] Note that choosing $rep(s) = min_{\leq}[s]_G$ is only one convenient way to choose a representative, but it is a method that is commonly used. Using another distinguished element would be equivalent, but would not allow us to adapt existing algorithms without conversion.

special cases in Section 4.1. Additionally, an approximate solution to the COP has been proposed, and shown to be effective [8].

We now turn to the more realistic model of computation where references are permitted. In order to define a total ordering $\preceq$ on $S \subseteq (L_c \times L_r)^n$, we define two projection mappings, *ctrl* and *ref*, projecting a state on to its control and reference parts respectively. For a state $s = (l_1, (r_{1,1}, r_{1,2}, \ldots, r_{1,m}), l_2, (r_{2,1}, r_{2,2}, \ldots, r_{2,m}), \ldots, l_n, (r_{n,1}, r_{n,2}, \ldots, r_{n,m}))$, $ctrl(s) = (l_1, l_2, \ldots, l_n)$ and $ref(s) = (r_{1,1}, r_{1,2}, \ldots, r_{1,m}, \ldots\ldots, r_{n,m})$.

Definition 3.4 For $s, t \in S$, $s \preceq t$ if either $s = t$; $ctrl(s) < ctrl(t)$; or $ctrl(s) = ctrl(t)$ and $ref(s) < ref(t)$. Here $ref(s)$ and $ref(t)$ are compared using the usual lexicographic ordering on vectors (similarly $ctrl(s)$ and $ctrl(t)$).

It is clear that $\preceq$ is a total ordering on states. We write $s \prec t$ if $s \preceq t$ and $s \neq t$. We now extend the COP to the model of computation with references:

Definition 3.5 The COP with References (COPR) Given a group $G \leq S_n$ and a state $s \in (L_c \times L_r)^n$, find $min_{\preceq}[s]_G$, the $\preceq$-least element in the orbit of s under G.

It is clear that COPR is a generalisation of COP – in the special case where $m = 0$ COP and COPR are identical. Since COP is *NP*-hard (Theorem 3.3), COPR is *NP*-hard by restriction. In fact, the two problems can be shown to be polynomial time equivalent. An instance of COP is trivially an instance of COPR, and an instance of COPR can be converted, in quadratic time, to an instance of COP. The latter is achieved by replacing each component id reference $r_{i,j}$ by a vector of n binary values, which are all 0 unless $r_{i,j} = l > 0$, in which case the binary value l places from the right is 1. For example, if $n = 8$ and $r_{i,j} = 5$, the value of $r_{i,j}$ is converted to the binary sequence $\underbrace{0,0,0}_{n-5},\underbrace{1,0,0,0,0}_{5}$. The variables introduced to hold these values are modelled as components with binary valued local state. If *convert* denotes a function which performs this conversion, then placing the value 1 l places to the right ensures that, for states s and t, $s \preceq t$ iff $convert(s) \leq convert(t)$. Elements of the symmetry group G must also be transformed appropriately, so that if s is a state and $\alpha \in G$, the transformed element α' must satisfy $convert(\alpha(s)) = \alpha'(convert(s))$.

4 Symmetry Reduction Strategies

Let $S \subseteq L_c^n$, and let $G \leq S_n$. A symmetry reduction *strategy* for a group $G \leq S_n$ is a function $f : S \to S$ with the property that $f(s) = min_{\leq}[s]_G$ [1]. Application of such a function typically involves repeated application of elements from G (the product of which is an element of G). We can equivalently define a symmetry reduction strategy with respect to a group G as follows:

Definition 4.1 An (exact) COP strategy for $G \leq S_n$ is a function $f : S \to G$ such that, for all $s \in S$, if $\alpha = f(s)$ then $\alpha(s) = min_{\leq}[s]_G$.

In other words, f applied to s yields an element of G which minimises s.

Algorithm 1 Algorithm to explore a quotient Kripke structure, given total ordering $\sqsubseteq$ on states.

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

Because of the difficulty of solving COP, some symmetry reduction approaches map states to a small number of representatives rather than a single representative [1,4]. This is known as the *multiple orbit representatives* approach. COP strategies which use multiple orbit representatives are said to be *approximate* [8].

Definition 4.2 An *approximate* COP strategy for $G \leq S_n$ is a function $f : S \to G$ such that, for all $s \in S$, if $\alpha = f(s)$ then $\alpha(s) \leq s$.

A good approximate strategy yields elements of G which map the orbit $[s]_G$ on to a small number of representatives. Exact and approximate strategies for the COPR are defined analogously, using the total ordering $\preceq$.

4.1 Summary of Polynomial Time Exact Strategies

Enumeration If $|G|$ is polynomial in n (e.g. when G is a cyclic or dihedral group arising from a uni/bi-directional ring network topology), $f(s)$ can be computed by enumerating the elements of G and returning an element α such that $\alpha(s) \leq \beta(s)$ for all $\beta \in G$ [4]. Clearly this strategy is not feasible if $|G|$ is exponential in n.

Symmetric groups When $G = S_n$, $f(s)$ can be taken to be any permutation which sorts s in increasing order [1,4]. For example, if $G = S_4$ and $s = (5, 2, 4, 3) \in \{0, 1, \ldots, 5\}^4$, sorting s gives $min_{\leq}[s]_G = (2, 3, 4, 5)$. Since sorting can be performed in polynomial time, this leads to a polynomial time exact COP strategy for S_n. Algorithm 2 gives such a strategy based on selection sort. The idea of solving COP by sorting is generalised in [8] to a class of groups which are isomorphic to S_m for some $m \leq n$.

Disjoint/wreath products If G is the *disjoint product* of subgroups H and K, and h and k are polynomial time strategies for COP for H and K respectively, then the strategy f defined by $f(s) = k(s)h(s)$ is a polynomial time strategy for G [4]. In other words, COP can be solved for G by applying in sequence permutations which minimise s with respect to H and K respectively. Disjoint product groups occur frequently in practice when there is full symmetry between multiple component types in a system. A similar approach can be used to obtain a polynomial time COP strategy when G is the *wreath product* of subgroups for which polynomial time strategies are available [4]. Wreath product groups typically occur when a system has a tree topology.

Algorithm 2 A COP strategy for S_n based on selection sort.

```
$\alpha := id$
**for all** $i \in [1, \ldots, n-1]$ **do**
  $\beta := id$
  **for all** $j \in [i+1, \ldots, n]$ **do**
    **if** $(i\ j)\alpha(s) < \beta\alpha(s)$ **then**
      $\beta := (i\ j)$
    **end if**
  **end for**
  $\alpha := \beta\alpha$
**end for**
return $\alpha$
```

4.2 Problems With References

The strategies summarised above were proposed for a model of computation without references. Clearly the strategy based on enumeration extends immediately to a model of computation with references, if $|G|$ is polynomial in n. However, the other strategies are not immediately applicable. We show this for the COP strategy where $G = S_n$ and representatives are computed by sorting. Similar arguments can be applied for the other strategies.

The proof that the COP for $G = S_n$ can be solved by sorting a state s is based on the following lemma.

Lemma 4.3 *In the simple model of computation, there are no $i_1, j_1, i_2, j_2 \in I$ where $i_1 < j_1$, $i_2 < j_2$, $(i_2\ j_2)(s) < s$ and $(i_1\ j_1)(s) \geq s$, but $(i_2\ j_2)(i_1\ j_1)(s) < (i_2\ j_2)(s)$.*

However, this result does not hold in the presence of references.

Lemma 4.4 *Lemma 4.3 does not hold for the model of computation with references where the ordering $\leq$ is replaced with $\preceq$.*

Proof. We prove Lemma 4.4 by counterexample. Suppose $n = 3$, and consider $s = (\mathbf{1}, 0, \mathbf{0}, 2, \mathbf{0}, 2)$. Take $i_1 = 2$, $j_1 = 3$, $i_2 = 1$ and $j_2 = 3$. Then we have $(i_2\ j_2)(s) = (\mathbf{0}, 2, \mathbf{0}, 2, \mathbf{1}, 0) \prec s$, $(i_1\ j_1)(s) = (\mathbf{1}, 0, \mathbf{0}, 3, \mathbf{0}, 3) \succ s$. But $(i_2\ j_2)(i_1\ j_1) = (1\ 3\ 2)$, and $(1\ 3\ 2)(s) = (\mathbf{0}, 1, \mathbf{0}, 1, \mathbf{1}, 0) \prec (i_2\ j_2)(s)$. □

This counterexample for the case $n = 3$ can be extended to give a counterexample for any $n \geq 3$ – consider i_1, j_1, i_2 and j_2 as above, and $s = (\mathbf{1}, 0, \mathbf{0}, 2, \mathbf{0}, 2, \mathbf{0}, 0, \ldots, \mathbf{0}, 0)$.

Applying Algorithm 2 with $\leq$ replaced by $\preceq$ to $s = (\mathbf{1}, 0, \mathbf{0}, 2, \mathbf{0}, 2)$ gives the element $(1\ 3)$ which does not minimise s, whereas enumeration of S_3 gives $(1\ 3\ 2)$, which does. Thus this adaptation of Algorithm 2 does not yield an exact COPR strategy.

Suppose $G \leq S_n$ is a symmetry group and $G' \leq S_{n'}$ is the group isomorphic to G obtained by the conversion of a COPR instance to a COP instance discussed in Section 3.3 (here $n' \geq n$ is the number of components used in the representation of converted states). A polynomial time COP strategy for G does not in general yield a corresponding COPR strategy for G', as the action of G' on $\{1, 2, \ldots, n'\}$ may be fundamentally different to that of G on $\{1, 2, \ldots, n\}$. For example, if G is the disjoint product of subgroups H and K then G' is the *direct* product of subgroups H' and K', but it may not be the case that H' and K' act disjointly on $\{1, 2, \ldots, n'\}$.

We now show how a polynomial time exact COP strategy can be extended to an exact COPR strategy. The result is not a polynomial time strategy, but may be significantly more efficient than the enumeration strategy if G is large.

5 Segmentation: Extending Strategies to a Model of Computation With References

Our approach to extending a strategy for COP to one for COPR works by constructing a *partition* of I from a given state, and enumerating the *stabiliser* of this partition.

5.1 Partitions and Stabilisers

A *partition* of I is a set $\mathcal{X} = \{I_1, I_2, \ldots, I_d\}$, where $d > 0$, $I_j \subseteq I$ $(1 \leq j \leq d)$, $I = \bigcup_{j=1}^{d} I_j$, and $I_i \cap I_j = \emptyset$ for $1 \leq i \neq j \leq d$.

Definition 5.1 Let $\mathcal{X}$ be a partition of I, and let $G \leq S_n$. The (partition) stabiliser of $\mathcal{X}$ in G is the subgroup $stab_G(\mathcal{X}) = \bigcap_{J \in \mathcal{X}} stab_G(J)$.

5.2 Segmenting a State

We define a subset of $[s]_G$ whose elements have minimal control states.

Definition 5.2 Let $small_G(s) = \{t \in [s]_G : ctrl(t) \leq ctrl(u) \;\forall\; u \in [s]_G\}$.

Clearly $min_{\preceq}[s]_G \in small_G(s)$. Given a state s, the vector $ctrl(s)$ can be viewed as a state under a model of computation without references. The following result is a consequence of this observation and Definition 5.2:

Lemma 5.3 *For $s \in S$, $t \in small_G(s) \Leftrightarrow ctrl(t) = min_{\leq}[ctrl(s)]_G$.*

For $0 \leq i \leq k$, let $s^{(i)} = \{j \in I : l_j = i\}$, i.e. the set of indices of components which have control state i in s. Define the function *seg* acting on states by

$$seg(s) = \{s^{(0)}, s^{(1)}, \ldots, s^{(k)}\}.$$

Then clearly, for any state s, $seg(s)$ is a partition of I.

5.3 Symmetry Reduction via Segmentation

Lemma 5.4 *If $t \in small_G(s)$ and $\alpha(t) < t$ for some $\alpha \in G$ then $\alpha \in stab_G(seg(t))$.*

Proof. Since $t \in small_G(s)$ and $\alpha(t) < t$, by Definition 5.2 clearly $ctrl(t) = ctrl(\alpha(t))$ and $ref(t) > ref(\alpha(t))$. Since $ctrl(t) = ctrl(\alpha(t))$, $t^{(i)} = \alpha(t)^{(i)}$ for $1 \leq i \leq k$, i.e. $seg(t) = seg(\alpha(t))$. Thus α preserves $seg(t)$, i.e. $\alpha \in stab_G(seg(t))$.□

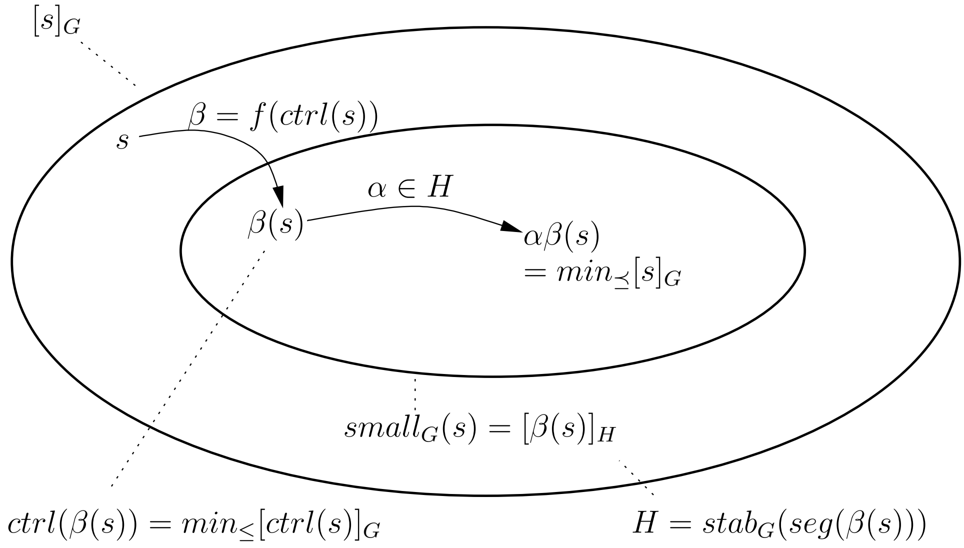
$[s]_G$
s
$\beta = f(ctrl(s))$
$\alpha \in H$
$\beta(s)$
$\alpha\beta(s)$
$= min_{\preceq}[s]_G$
$small_G(s) = [\beta(s)]_H$
$ctrl(\beta(s)) = min_{\leq}[ctrl(s)]_G$
$H = stab_G(seg(\beta(s)))$

Fig. 1. Symmetry reduction by segmentation.

Thus, if a state $t \in small_G(s)$ is not the smallest element in $[s]_G$ under $\preceq$ then search for a minimising element of G can be restricted to $stab_G(seg(t))$. Note that if component indices $i, j \in X \in seg(t)$, it is still necessary to consider elements of G which map i to j. Thus we cannot treat the elements of $seg(t)$ as sequences and compute their pointwise stabiliser (which would be computationally easier).

Suppose that we have an exact COP strategy f for G. Let $\beta = f(ctrl(s))$, so that $\beta(ctrl(s)) = min_{\leq}[ctrl(s)]_G$. Clearly $\beta(ctrl(s)) = ctrl(\beta(s))$, and therefore by Lemma 5.3, $\beta(s) \in small_G(s)$. By Lemma 5.4, the group $H = stab_G(seg(\beta(s)))$ can now be enumerated to find an element α such that $\alpha\beta(s) \preceq \delta\beta(s)$ for all $\delta \in H$. Thus we have proved the following:

Theorem 5.5 *Let $s \in S$, $G \leq Aut(\mathcal{M})$, and let f be an exact COP strategy for G. Then Algorithm 3 is an exact COPR strategy for G.*

Algorithm 3 Extending an exact COP strategy f for a group G to an exact COPR strategy.

```
$\beta := f(ctrl(s))$
$H := stab_G(seg(\beta(s)))$
$\alpha = id$
for all $\delta \in H$ do
  if $\delta\beta(s) \prec \alpha\beta(s)$ then
    $\alpha := \delta$
  end if
end for
return $\alpha\beta$
```

Figure 1 illustrates graphically the relationship between $[s]_G$ (represented by the outer ellipse) and its subset $small_G(s)$ (represented by the inner ellipse), and the process of computing an element of G which minimises s. We illustrate the approach further with an example.

Let n, m, L_c and L_r and G be as in the example in Section 4.2. Let $s = (\mathbf{1},2,\mathbf{0},1,\mathbf{0},1,\mathbf{2},1)$. Then $ctrl(s) = (1,0,0,2)$, and applying Algorithm 2, we find that $\beta = (1\ 3)$ satisfies $\beta(ctrl(s)) = min_{\leq}[ctrl(s)]_G$. Applying β to s gives $t = (\mathbf{0},3,\mathbf{0},3,\mathbf{1},2,\mathbf{2},3)$, and $seg(t) = \{\{1,2\},\{2\},\{3\}\}$. It is easy to check that $stab_G(seg(t)) = \langle(1\ 2)\rangle$, a group of order 2, and that applying $(1\ 2)$ to t gives $min_{\preceq}[s]_G = (\mathbf{0},3,\mathbf{0},3,\mathbf{1},1,\mathbf{2},3)$. For this example, the application of 6 group elements is required by Algorithm 2, followed by enumeration of a group of order 2. Computing $min_{\preceq}[s]_G$ by basic enumeration would have required the application of all 24 elements of G to s.

6 Efficiency

Assuming that f can be computed in polynomial time (using strategies described in [4,8]), the efficiency of Algorithm 3 is dominated by computation of and iteration over H.

Computing $H = stab_G(seg(s))$ is equivalent to computing the stabiliser of a set in a group. The most efficient algorithms available for computing set stabilisers involve backtrack search of the group using a base and strong generating set [2,16]. Typically this search can be heavily pruned using both problem-independent heuristics, and heuristics based on properties of set stabilisers. Thus, despite the fact that no polynomial time algorithm is known for computing set stabilisers, the associated overhead is not large. Furthermore, as the experimental results of Section 7.1 show, the set $\{seg(s) : s \in S\}$ of all partitions of I which must be considered during search, is often much smaller than the number of possible partitions of I.[5] Thus, re-computation of partition stabilisers can be avoided by caching partition-stabiliser pairs.

In the worst case, H may have size $|G|$ (e.g. when $|seg(s)| = 1$), and $|G|$ may be as large as $n!$ (in the case where $G = S_n$). However, if the number of distinct component control states is reasonably large, many states s will have the property that $|seg(s)| = n$, in which case $stab_G(seg(s))$ is the trivial group.

7 Implementation for the Spin Model Checker

We have implemented the approach presented in this paper as an addition to TopSpin [7], a symmetry reduction package for the Spin model checker [11]. Computation of partition stabilisers during search is performed via the computational algebra system Gap [10]. Given a Promela specification, Spin produces C code for a corresponding executable verifier. TopSpin automatically detects component symmetries from the Promela specification, and (using a method similar to that of SymmSpin [1]) adds symmetry reduction algorithms to the verifier according to one of 4 options specified by the user.

If the *enumerate* option is selected then memory optimal symmetry reduction will be applied during verification, with a runtime overhead proportional to the size of the symmetry group G (so this is useful only for small groups, or for comparison

[5] The number of such partitions is B_n, the nth Bell number, which is defined recursively by $B_0 = 1$ and $B_n = \sum_{k=0}^{n-1} \binom{n}{k} B_k$ for $n > 0$ [15].

with more efficient options). If the *fast* or *segment* option is chosen then TopSpin uses Gap to analyse the structure of G in an attempt to find a polynomial time exact COP strategy, as described in [8]. With the *fast* option, such a strategy will be used directly. This results in memory optimal verification if components of the specification do not hold references to one another; otherwise model checking may involve the use of multiple representatives from each orbit. Choosing the *segment* option means that the COP strategy will be extended to a memory optimal COPR strategy using the method presented in this paper. In this case, the verifier must currently be instantiated from within the Gap system. Finally, the *hillclimbing* option can be selected, in which case an approximate symmetry reduction strategy based on hillclimbing local search will be applied [8]. This strategy will be used by default if no polynomial time COP strategy for G can be found with the *fast* or *segment* options.

7.1 Experimental Results

We illustrate the variation in memory requirements and verification time for the *enumerate*, *fast* and *segment* strategies using various configurations of two Promela examples: an email system, and a loadbalancer which forwards requests from a pool of clients to a pool of servers in a fair manner.

The email example is adapted from [3], and is used as a case study for symmetry reduction in [8]. A configuration of the system consists of p *client* processes, which communicate by sending messages to a *mailer* process via a *network* channel component. The client components are instantiations of the same parameterised process and thus behave identically, so there is full symmetry between clients. Components in a Promela specification of the system use reference variables to keep track of the sender and recipient of a given message. A configuration of the email example with p clients is denoted *email* p. Components in a configuration of the loadbalancer example are a set of p *server* and q *client* processes with associated communication channels, and a *loadbalancer* process (with a dedicated input channel). The *load* of a server is the number of messages queued on its input channel. Client processes send requests to the loadbalancer, and if some *server* channel is not full, the loadbalancer forwards a request nondeterministically to one of the least loaded *server* queues. Each request contains a reference to the input channel of its associated client process, and the server designated by the loadbalancer uses this channel to service the request. A configuration with p clients and q is denoted p/q. There is full symmetry between the p servers and also between the q clients, thus a p/q configuration has a disjoint product symmetry group of order $p!q!$. For both examples, we verify safety properties embedded in the specification as assertions.

Table 1 contains experimental results for various configurations of the email and loadbalancer examples. For each configuration, we give the number of model states without symmetry reduction (**orig**), with memory optimal symmetry reduction via the *enumerate* or *segment* options (**red**), and with symmetry reduction via the *fast* option (**fast**). The use of state compression, an option provided by Spin, is indicated by the number of states in italics. This option was selected for two configurations to allow verification without symmetry reduction by storing states more efficiently, with an associated time overhead. Verification times (in seconds) are given for the

system	states orig	time orig	$\lvert G\rvert$	states red	time enum	time seg	ptns	states fast	time fast
email 3	23256	0.1	6	3902	0.8	0.2	5	3908	0.2
email 4	852641	9	24	36255	6	4	7	38560	2
email 5	*3.04×10*^{7}	3576	120	265315	253	71	9	315323	40
email 6	-	-	720	1.7×10^6	13523	1600	11	2.3×10^6	576
email 7	-	-	5040	$9.3{\times}10^6$	-	50970	13	1.53×10^7	6573
lb 2/6	*2.37×10*^{7}	1585	1440	23474	265	28	94	31066	5
lb 2/7	-	-	10080	44137	4376	266	259	61245	16
lb 3/6	-	-	4320	125126	5024	271	330	256204	57
lb 3/7	-	-	30240	293657	-	2722	451	685167	213
lb 4/6	-	-	17280	527548	-	2378	884	$1.7{\times}10^6$	487
lb 4/7	-	-	120960	$1.2{\times}10^6$	-	29779	1296	$3.7{\times}10^6$	1583

Table 1
Experimental results for configurations of the email and loadbalancer (lb) specifications.

enumerate (**enum**), *sement* (**seg**) and *fast* (**fast**) options, as well as for the case where symmetry reduction is not applied (**orig**). The size of the symmetry group ($|G|$) and the number of partitions which arise using the *segment* option (**ptns**) are also given. Verification attempts which exceed available resources, or do not terminate within 15 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.

For both examples, and especially for the loadbalancer, the use of symmetry reduction techniques allows the verification of larger configurations – even using state compression, memory requirements were quickly exceeded when symmetry reduction was not applied. When G is large, enumeration is not a feasible technique, but with the *segment* option it was possible to perform model checking over quotient structures using unique representatives. Although this is slower than using the *fast* option, the reduction in states for large configurations of the loadbalancer example is encouraging. As discussed in Section 6, this time overhead is mostly due to the final step of minimising a state, which involves enumeration of a (potentially large) subgroup of G.

8 Related Work

The COP for symmetry reduction in model checking was investigated in [4], and polynomial time strategies for certain classes of groups, summarised in Section 4.1, were proposed.

The problem of performing symmetry reduction in the presence of inter-component references is investigated in [1], where the *segmented* strategy provides memory optimal symmetry reduction. This strategy applies when $G = S_n$, and is informally described as the process of applying every possible permutation which sorts the *main array* of a state, and selecting the permutation which results in the smallest image. The main array of a state s is analogous to $ctrl(s)$. We have formalised this approach using the ideas of partitions and stabilisers, and generalised the method to apply to arbitrary subgroups of S_n. An approximate strategy for fully symmetric groups – the *sorted* strategy – is also proposed, and experimental results using the SymmSpin symmetry reduction package for Spin illustrate the same trade-off between memory and verification time as observed in Section 7.1. Another approach to exploiting symmetry in the Spin model checker handles reference variables by adding keywords to the Promela language [6].

A recent approach to symmetry breaking in constraint programming requires a solution to a problem related to COP [13]. During search, symmetry breaking is performed by determining whether the partial assignment of variables at a given node is lexicographically least in its orbit under a symmetry group G. If not, search backtracks. The approach relies on a variant of an algorithm for finding the smallest image of a set under a permutation group [14]. This problem can be shown to be polynomial time equivalent to COP.

9 Conclusions and Future Work

We have presented an approach to extending symmetry reduction strategies which provide memory optimal reduction under a simple model of computation, so that memory optimality is maintained under a realistic model of computation where components hold references to one another. This approach formalises and generalises the *segmented* strategy employed by the SymmSpin model checker, and applies to arbitrary symmetry groups for which a polynomial time COP strategy can be found. We have implemented our techniques within the TopSpin symmetry reduction package for Spin, using Gap to perform group theoretic computations. Experimental results using two Promela examples show that the approach results in a significant speedup over symmetry reduction by enumeration, and illustrate the trade off between memory and verification time associated with the choice of an exact or approximate symmetry reduction strategy.

The main bottleneck of the approach is the enumeration of the partition stabiliser after a polynomial time COP strategy has been applied. Future work includes implementing a canonicalisation algorithm presented in [14] within TopSpin to alleviate this problem.

Acknowledgements Thanks to David Manlove for useful discussion relating to the relationship between COP and COPR, and to the EPSRC funded Symmetry in Search network (SymNet), for providing a forum for discussions which aided this work.

References

[1] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):65–80, 2002.

[2] G. Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *LNCS*. Springer-Verlag, 1991.

[3] M. Caler and A. Miller. Generalising feature interactions in email. In *FIW'03*, pages 187–204. IOS Press, 2003.

[4] 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.

[5] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[6] F. Derepas and P. Gastin. Model checking systems of replicated processes with Spin. In *SPIN'01*, LNCS 2057, pages 235–251. Springer-Verlag, 2001.

[7] 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.

[8] A.F. Donaldson and A. Miller. Exact and approximate strategies for symmetry reduction in model checking. In *FM'06*, LNCS 4085, pages 541–556. Springer, 2006.

[9] E.A. Emerson and T. Wahl. Dynamic symmetry reduction. In *TACAS'05*, LNCS 3440, pages 382–396. Springer, 2005.

[10] The Gap Group. *GAP–Groups Algorithms and Programming, Version 4.2.* Aachen, St. Andrews, 1999. http://www-gap.dcs.st-and.ac.uk/~gap.

[11] G. J. Holzmann. *The SPIN model checker: primer and reference manual.* Addison Wesley, 2003.

[12] C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2): 41–75, 1996.

[13] C. Jefferson, T. Kelsey, S. Linton and K. Petrie. GAPLex: combining static and dynamic symmetry breaking. CP Pod Technical Report, CPPod-15-2006, 2006.

[14] S. Linton. Finding the smallest image of a set. In *ISSAC'04*, pages 229–234. ACM Press 2004.

[15] G.C. Rota. The number of partitions of a set. *Amer. Math Monthly*, 71: 498–504, 1964.

[16] A. Seress. *Permutation Group Algorithms.* Cambridge University Press, 2003.