Symmetry Reductions in Model Checking *

E. M. Clarke† E. A. Emerson‡ S. Jha† A.P. Sistla§

Abstract

The use of symmetry to alleviate state-explosion problems during model-checking has become a important research topic. This paper investigates several problems which are important to techniques exploiting symmetry. The most important of these problems is the *orbit problem*. We prove that the orbit problem is equivalent to an important problem in computational group theory which is at least as hard as the graph isomorphism but not known to be *NP*-complete. This paper also shows classes of commonly occurring groups for which the orbit problem is easy. Some methods of deriving symmetry for a shared variable model of concurrent programs are also investigated. Experimental results providing evidence of reduction in state space by using symmetry are also provided.

1 Introduction

Temporal Logic Model Checking is a technique for determining whether a temporal logic formula is valid in a finite state system $M = (S, R, L)$, where S is the state space, R is the state transition relation, and L is a function that labels states with sets of atomic propositions [4]. Such a structure is usually called a *Kripke structure* and may have an enormous number of states because of the *state explosion* problem. A system with n boolean state variables can have a state space which is exponential in n. An efficient Model Checking procedure tries to reduce the number of states that are actually explored [2, 4]. Recently, techniques which exploit the inherent symmetry of the system while performing model checking have become quite popular [3, 5, 10].

Basically, the idea of exploiting symmetry is the following: given a Kripke Structure $M = (S, R, L)$, a symmetry group G is a group acting on the state set S that *preserves* the transition relation R, G partitions the state set S into equivalence classes called *orbits*. A quotient model M_G is constructed that contains one or more representative from each orbit. The state space S_G of the quotient model will, in general, be much smaller than the original state space S. This makes it possible to verify much larger structures.

*This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597 and in part by the National Science Foundation under Grant no. CCR-8722633 and in part by the Semiconductor Research Corporation under Contract 92-DJ-294.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA

‡Department of Computer Science, University of Texas, Austin, TX

§Department of Electrical Engg and Computer Science, University of Illinois, Chicago, IL

This paper investigates several problems associated with exploiting symmetry. For example, given a group G and two states s and s', the *orbit problem* asks whether s and s' are in the same orbit. Determining whether two states are in the same orbit is at the core of any model checking procedure exploiting symmetry. We prove that the orbit problem is equivalent to an important problem in computational group theory. We explore ways of deriving symmetries of shared variable concurrent programs. Since the orbit problem in its full generality is quite hard, this paper also shows that the orbit problem is easy for certain commonly occurring groups and provides alternative techniques which do not require solving the orbit problem. We have also built a model-checker called `SYMM` which allows the user to specify the symmetries of the model and then uses the symmetries to perform efficient model-checking. Few of these problems were considered in [3, 5]. The results provided in this paper represent significant advances over those presented in [3, 5]. Although the paper only presents results for asynchronous composition of processes, all the results do extend to synchronous composition of processes but are not presented here because of lack of space. Complete details will be provided in the full version of the paper.

Our paper is organized as follows: In Section 2 we introduce a shared variable model. Section 3 gives a technique to derive symmetries of shared variable programs. Section 4 investigates the complexity of the orbit problem. Section 5 investigates some special classes of the orbit problem. Experimental results on an arbiter example are shown in section 6. Section 7 concludes with some interesting future directions. Appendix A contains the background on temporal logic, symmetry groups and group theory. Readers who are not familiar with these concepts are encouraged to read this appendix. Some selected proofs are provided in Appendix B.

2 A Shared Variable Model of Computation

A *shared variable program* is defined with the state sets and the transition relation as follows:

- $S = Loc^I \times D^V$ is the finite set of *states*, with Loc a finite set of individual process *locations*, I the set of process indices, and V is a finite set of shared *variables* over a finite *data domain* D.

- $R \subseteq S \times S$ which represents the transitions of the system.

For convenience, each state $s = (s', s'') \in S$ can be written in the form $(\ell_1, \ldots, \ell_n, v = d, \ldots, v' = d')$ indicating that processes $1, \ldots, n$ are in locations $\ell_1, \ldots, \ell_n$, respectively and the shared variables $v, \ldots, v'$ are assigned data values $d, \ldots, d'$, respectively.

Next, we define a labeling function for a shared variable program. The set of *terms* are expressions of the form l_i ($i \in I$) and $v = d$ ($v \in V$ and $d \in D$). The set of atomic propositions AP are constructed from the set of terms by the logical connectives $\wedge$ and $\neg$. Given an atomic proposition $p \in AP$ and a state $s \in S$, the satisfaction relation $s \models p$ is defined in the following way:

$s \models l_i$ iff the i-th process in the state s is in location l_i.
$s \models (v = d)$ iff the shared variable v has the value d in the state s.
$s \models f \wedge g$ iff $s \models f$ and $s \models g$.

$s \models \neg f$ iff $s \not\models f$.

Given a shared variable program, we can construct a corresponding Kripke Structure $M = (S, R, L)$ (S and R were defined before) by constructing the following labeling function $L : S \to 2^{AP}$: $p \in L(s) \Leftrightarrow s \models p$.

In practice, for ordinary model checking, M is the Kripke Structure corresponding to a finite state *concurrent program* $\mathcal{P}$ of the form $\|_{i=1}^{n} K_i$ consisting of processes $K_1, \ldots, K_n$ running in parallel. Each K_i may be viewed as a finite state transition graph with node set *Loc*. An arc from node ℓ to node ℓ' may be labeled by a guarded command $B \to A$. The guard B is an atomic proposition that can inspect shared variables and local states of "accessible" processes. A is a set of *simultaneous assignments* to shared variables $v := d \parallel \cdots \parallel v' := d'$. When process K_i is in local state ℓ and the guard B evaluates to *true* in the current global state, the program $\mathcal{P}$ can nondeterministically choose to advance by firing this transition of K_i which changes the local state of K_i to be ℓ' and the shared variables in V according to A. Thus the arc from ℓ to ℓ' in K_i represents a *local transition* of K_i denoted by $\ell : B \to A : \ell'$.

The Kripke structure $M = (S, R, L)$ corresponding to $\mathcal{P}$ is thus defined using the obvious formal operational semantics. First, the set of (all possible) states S is determined from $\mathcal{P}$ because it provides us with the set of local (i.e., individual process) locations *Loc*, process indices I, variables V, and data domain D. For states $s, t \in S$, we define $s \to t \in R$ iff $\exists i \in I$ such that the process K_i can cause s to move to t, denoted by $s \to_i t$ iff $\exists i \in I$ $\exists$ local transition $\tau_i = \ell_i : B_i \to A_i : m_i$ of K_i which *drives* $s = (s', s'')$ to $t = (t', t'')$; i-th component of s' equals ℓ_i, the i-th component of t' equals m_i, all other components of s' equal the corresponding component of t', predicate $B_i(s) = true$, and $t'' = A_i(s'')$. $A_i(s'')$ is constructed from s'' by replacing the values of the shared variables according to the simultaneous assignment statement A_i. The labeling function L is defined as before. Notice that we use asynchronous composition in the definition given here. Analogous definition can be given for synchronous composition. All the results given in the paper hold for synchronous composition, but are not stated because of lack of space.

3 Deriving Symmetry

This section deals with deriving symmetry for shared variable programs introduced in the previous section. Intuitively, if one has a graph G whose nodes corresponds to processes and the processes communicate over the edges of G, an automorphism of the graph G should manifest itself into a symmetry of the underlying structure. Succinctly speaking, *structural symmetry introduces symmetry in the model* [3]. This section proves that for certain cases one can derive the symmetry of the model from the topology of the system. Given a concurrent program $\mathcal{P} = \|_{i=1}^{n} K_i$, we build a hypergraph $HG(\mathcal{P})$. Under certain restrictions, we prove that each automorphism of $HG(\mathcal{P})$ is also a symmetry of the underlying Kripke Structure M. A restricted version of the theorem appeared in [5]. In [5] the authors assumed that all processes are isomorphic and the variables are only shared between two processes. For instance, consider an arbiter which maintains a global *master* variable (indicating who has the resource) could be handled by the framework presented in this section, but the theorem given in [5] does not apply in this context.

Let $\mathcal{P} = \|_{i=1}^{n} K_i$ be a concurrent program. In this section the index set is $I = [n]$. Each shared variable v is subscripted by the set of indices of the processes which access that shared variable. For example, if x is accessed by processes 1, 4, and 5, we write x as $x_{\{1,4,5\}}$. Notice that each shared variable is uniquely determined by its name and subscript, but we allow shared variables to have the same name as long as their subscripts are different. For example, $x_{\{1,2\}}$ and $x_{\{3,4\}}$ are allowed. A permutation $\pi \in S_n$ acts on the variables in a natural manner, i.e., $\pi(x_w) = x_{\pi(w)}$. A permutation π acting on $[n]$ is called *consistent* if and only if for every shared variable x_w, $x_{\pi(w)}$ is a variable as well. This means that we only allow permutations which map shared variables to shared variables.

We define how a consistent permutation π acts on states, atomic propositions, and processes. Let π be a consistent permutation.

- Given a state $s = (l_1, \cdots, l_n, v_{w_1} = d_1, \cdots, v_{w_k} = d_k)$, the state $\pi(s)$ is defined as follows: i-th process is in location $l_{\pi(i)}$ in the state $\pi(s)$, and the shared variable $v_{\pi(w)}$ in the state $\pi(s)$ has the same value as the variable v_w in the state s.

- Let $p \in AP$ be an atomic proposition. $\pi(p)$ is recursively defined as follows: $\pi(f \wedge g) = \pi(f) \wedge \pi(g)$, $\pi(\neg f) = \neg\pi(f)$, $\pi(l_i) = l_{\pi(i)}$, and $\pi(v_w = d) = (v_{\pi(w)} = d)$.

- Given a simultaneous assignment $A = (v_{w_1} = d_1 \| \cdots \| v_{w_k} = d_k)$, define $\pi(A)$ as the following simultaneous assignment: $v_{\pi(w_1)} = d_1 \| \cdots \| v_{\pi(w_k)} = d_k$

- Given a process K_i, the process $\pi(K_i)$ is constructed in the following manner: $l : B \to A : l'$ is a transition in K_i iff $l : \pi(B) \to \pi(A) : l'$ is a transition in $\pi(K_i)$.

Definition 3.1 A *colored hypergraph* with n vertices and k colors is a 3-tuple $H = ([n], E, L)$ such that $E \subseteq 2^{[n]}$ is the *edge set*, and $L : [n] \to [k]$ is the *coloring function* which colors each node with one of the k colors. A permutation π acting on $[n]$ is called an *automorphism* of the hypergraph H iff the following two conditions hold:
For all $1 \leq i \leq n$, $L(i) = L(\pi(i))$.
$w \in E$ iff $\pi(w) \in E$.
The group of automorphisms of the hypergraph H is denoted by $Aut(H)$.

Definition 3.2 Given a concurrent program $\mathcal{P} = \|_{i=1}^{n} K_i$, define the *corresponding* colored hypergraph $HG(\mathcal{P}) = ([n], E, L)$ in the following manner:

- $w \in E$ iff there exists a shared variable with subscript w.

- Assume that we have a equivalence relation $\cong$ on the processes $K_1, \cdots, K_n$. Partition the processes $K_1, \cdots, K_n$ into equivalence classes induced by the relation $\cong$. Let $c_1, \cdots, c_k$ be the k equivalence classes. The coloring function L is defined as follows: $L(i) = r$ iff the process K_i is in the equivalence class c_r.

The definition of the equivalence relation $\cong$ will depend on the particular example, but here are two choices:
$K_i \cong K_j$ if and only if there exists a consistent permutation π such that $\pi(K_i) = K_j$.
K_i and K_j are the instances of the same `MODULE` definition [14].

Our goal is to relate the automorphism group of the Kripke Structure M (corresponding to the program $\mathcal{P} = \|_{i=1}^n K_i$) to the automorphism group of the hypergraph $HG(\mathcal{P})$. We want to make sure that every automorphism of $HG(\mathcal{P})$ which maps vertex i to j also maps process K_i to K_j, i.e., respects the structure of the program $\mathcal{P}$. To achieve this we introduce the some definitions. Let $\Gamma(K_i)$ be the set of indices j such that there exists a shared variable x_w such that $\{i, j\} \subseteq w$. Intuitively, if $j \in \Gamma(K_i)$, then K_i and K_j share some variables. $\Gamma(K_i)$ is called the *neighborhood* of K_i. We require that $i \in \Gamma(K_i)$, i.e., a process is in its own neighborhood. Process K_i *respects* its neighborhood if and only if given any consistent permutation π such that $\pi(i) = j$ and for all $r \in \Gamma(K_i)$, $K_r \cong K_{\pi(r)}$, then $\pi(K_i) = K_j$. Intuitively, the various conditions can be interpreted as follows:

- We only consider permutations π which map similar processes (or nodes in the corresponding hypergraph) to similar processes, or for all $r \in \Gamma(K_i)$, $K_r \cong K_{\pi(r)}$.

- The main condition is that if $\pi(i) = j$, then $\pi(K_i) = K_j$. This condition allows us to translate a automorphism of the hypergraph $HG(\mathcal{P})$ into the symmetry of the Kripke Structure for $\mathcal{P}$.

If the processes are entirely symmetric (given any permutation π, $\pi(K_i) = K_{\pi(i)}$), then K_i respects their neighborhood. To see this observe that given any permutation π such that $\pi(i) = j$, we have $f(K_i) = K_{\pi(i)} = K_j$. An algorithm for determining whether process respects its neighborhood exists, but is not provided in this paper because of lack of space.

Theorem 3.1 Let $HG(\mathcal{P})$ be the hypergraph corresponding to the program $\mathcal{P} = \|_{i=1}^n K_i$. Moreover, assume that for all $1 \leq i \leq n$, K_i respects its neighborhood. Let M be the Kripke Structure corresponding to $\mathcal{P}$. Given these conditions, $Aut(HG(\mathcal{P})) \leq Aut(M)$.

4 Complexity of the Orbit Problem

In this section we assume that the state space of our system is given by assignments to n boolean state variables $x_1, \cdots, x_n$. Therefore, the state space is isomorphic to B^n (where $B = \{0, 1\}$). We assume that the symmetry group $G \leq S_n$ acts on B^n in the natural way: a permutation σ maps a vector $(z_1, \cdots, z_n)$ to $(z_{\sigma(1)}, \cdots, z_{\sigma(n)})$. The orbit problem is at the core of any method exploiting symmetry [3, 5, 12, 10]. The orbit problem asks whether two states s and s' (which in this case are two 0-1 vectors of size n) are in the same orbit, i.e., there exists a permutation $\sigma \in G$ such that $s' = \sigma(s)$. In [3] it was proved that the graph isomorphism problem can be reduced to the orbit problem. Therefore, the orbit problem is atleast as hard as the graph isomorphism problem. Here we show that the orbit problem is equivalent to the problem of finding a set stabilizer of a set Y in a coset (we call this problem SSC). Since the graph isomorphism problem can be reduced to SSC [7], this result subsumes the result which appeared in [3]. Moreover, SSC (and hence the orbit problem) is equivalent to several important problems in computational group theory, which are harder than graph isomorphism, but not known to be NP-complete. Proofs of most the theorems are based on techniques introduced in [13].

The Orbit Problem (OP): Given two 0-1 vectors x and y of size n and a group $G \leq S_n$, does there exist a permutation $\sigma \in G$ which maps x to y, i.e., $y = \sigma(x)$.

Set Stabilizer in a coset (SSC): Given a set $Y \subseteq [n]$, let $G \leq S_n$ be a group and $\gamma \in S_n$ be a permutation. The problem is to find whether there exists $\sigma \in G\gamma$ which *stabilizes* the set Y, i.e., $\sigma(Y) = Y$.

Constructive Set Stabilizer in a coset ($CSSC$): Given a set $Y \subseteq [n]$, let $G \leq S_n$ be a group and $\gamma \in S_n$ be a permutation. The problem is to find whether there exists $\sigma \in G\gamma$ which stabilizes the set Y, i.e., $\sigma(Y) = Y$ and if yes, to exhibit such a σ.

Lemma 4.1 The problems SSC and $CSSC$ are polynomially equivalent.

Theorem 4.1 The problems OP and SSC are polynomially equivalent.

In general, the SSC problem is hard because the *graph isomorphism* problem can be reduced to it [7]. Conditions under which SSC can be solved in polynomial time are discussed in [7, 13]. In [7] it is proved that the *Coset Intersection Emptiness* problem stated below is polynomially equivalent to several important problems in computational group theory.

Definition 4.1 Coset Intersection Emptiness (CIE) Given the groups $A, B \leq S_n$ by generating sets and given a permutation $\pi \in S_n$, test whether $A\pi \cap B$ is empty.

Theorem 4.2 The set stabilizer in a coset problem (SSC) is polynomially equivalent to **CIE**.

Therefore, using the previous theorems one can deduce that OP is polynomially equivalent to **CIE** and hence several problems in computational group theory.

4.1 The Constructive Orbit Problem

Modeling states by boolean variables, in some cases, is too cumbersome and detailed. For example, consider the shared variable program introduced in Section 2. Let $\mathcal{P} = \|_{i=1}^{n} K_i$ be a concurrent program which does not have shared variables. Let the size of the set of locations Loc be k. In this case, a typical state in $\mathcal{P}$ is given by a vector of size n whose elements are integers between 1 and k, i.e., the space $[k]^n$. Permuting the processes K_i amounts to permuting the corresponding integers in that state. A symmetry group $G \leq S_n$ acts on the space $[k]^n$ in the following way: a permutation $\sigma \in G$ maps $(x_1, \cdots, x_n)$ to $(x_{\sigma(1)}, \cdots, x_{\sigma(n)})$.

Given a symmetry group G, one frequently needs a *representative function* $\xi : S \rightarrow S$ [3, 5, 10] (S is the state space of the system) which has the following properties:

- s and $\xi(s)$ are in the same orbit.
- If s and s' are in the same orbit, then $\xi(s) = \xi(s')$.

Such a representative function is used during state exploration in [3, 5, 10]. The need to find such a representative function motivates the following problem.

Definition 4.2 The Constructive Orbit Problem (COP): Given a group acting on $[n]$ and vector $x = (x_1, \cdots, x_n)$ find the lexicographically least element (or lex-least element for short) in the orbit of x (the group G permutes the indices of x)

Notice that if one can solve COP in polynomial time, one can construct the representative function ξ. Given a state x, $\xi(x)$ is simply the lex-least element in its orbit. In [1] it is proved that the problem is NP-hard. The paper also shows that for certain special groups COP can be solved in polynomial time. Actually, for our purposes it is enough to find a canonical element from each orbit.

5 Working Around the Orbit Problem

Results of section 4 prove that the Orbit problem and the Constructive Orbit Problem are quite hard in its full generality. In this section we discuss two possible techniques which will help circumvent the complexity of the orbit problem.

1. We prove that for a large class of groups, which occur commonly in practice, the orbit problem can be easily solved. This means that if the symmetries are restricted to this class of groups, the orbit problem can be easily solved.

2. We also describe an approach that uses multiple representatives from each orbit rather than just one. This approach is partcularly useful for symbolic model checking [3].

The ensuing subsections outline these approaches.

5.1 Easy Groups

Notice that if a group $G \leq S_n$ has polynomial size, COP for G can be solved in polynomial time by exhaustive enumeration. For example, a rotation group acting on set of size $[n]$ has order n. Therefore, for the rotation group one can solve COP in linear time. The lemma given below states that if COP can be solved in polynomial time for two disjoint groups J and K, then COP can be solved in polynomial time for their direct product.

Lemma 5.1 Let G be a disjoint product of J and K. If COP for J and K can be solved in polynomial time, then COP for G can be solved in polynomial time.

The next lemma is similar to the previous one but refers to wreath products.

Lemma 5.2 Let $G = J \wr K$ be the wreath product of J and K. The group J, K, G act on the sets $[n], [m], [nm]$ respectively. If COP for J, K can be solved in polynomial time, then COP for G can be solved in polynomial time.

Lemma 5.3 Let S_n be the full symmetric group acting on the set $[n]$. The COP problem for S_n can be solved in polynomial time.

Proof: Given a vector $x = (x_1, \cdots, x_n)$, the lex-least element of x under the group S_n can be obtained by sorting the elements x_i. $\Box$

In practice, symmetries are given as a set of transpositions. For example, in a system which has the star topology, the two outer processes can be switched. The lemma given below states that if the group is only generated by transpositions, then COP for it can be solved in polynomial time.

Lemma 5.4 Let G be a permutation group acting on the set $[n]$. Assume that G is generated by a set of transpositions S. The *COP* problem for G can be solved in polynomial time.

This means that during model-checking if one restricts to the class of groups mentioned above, exploiting symmetry is relatively easy. Hence while exploiting symmetry one should try to work with these easy groups.

5.2 Multiple Representatives

Multiple Representatives were discussed in [3]. The account given here is much cleaner and general. Assume that we are given a Kripke Structure $M = (S, R, L)$ and a symmetry group G of M. Let $C \subseteq G$ be an arbitrary set of permutations which is *inverse closed*, i.e., $\pi \in C$ implies that $\pi^{-1} \in C$. We also assume that the identity permutation e is in C. Let $Rep \subseteq S$ be a set of *representatives* which satisfies the following requirements:

1. Every orbit of S under the action of G has a non-empty intersection with *Rep*.
2. Given an $s \in S$, there exists a $\sigma \in C$ such that $\sigma(s) \in Rep$.

Let $M_{Rep} = (Rep, R_{Rep}, L_{Rep})$, where R_{Rep} and L_{Rep} are defined as follows:

- We have that $(r_1, r_2) \in R_{Rep}$ iff there exists $s \in S$ and $\sigma \in C$ such that $(s, r_2) \in R$ and $\sigma(s) = r_1$.
- $L_{Rep}(r) = L(r)$.

The *representative relation* $\xi \subseteq S \times Rep$ is defined as follows: $(s, r) \in \xi$ iff there exists a $\sigma \in C$ such that $\sigma(s) = r$. Notice that $R_{Rep} = \eta^{-1} \circ R$.

Theorem 5.1 let $M = (S, R, L)$ be a Kripke structure, G be a *symmetry group* of M, and h be a CTL^* formula. If G is an invariance group for all the atomic propositions p occurring in h, then for all r such that $(s, r) \in \xi$

$$M, s \models h \Leftrightarrow M_{Rep}, r \models h$$

A natural question to ask is: how does one choose *Rep* given a set of polynomial size C? One natural choice of *Rep* is given below:

$$r \in Rep \quad \Leftrightarrow \quad \forall(\sigma \in C)(r \geq \sigma(r))$$

where $\leq$ represents the lexicographical ordering on the 0-1 vectors. It is easy to check that *Rep* has the required properties. Also notice that given an $s \in S$ and $r \in Rep$ one can decide the following questions in polynomial time?

1. Is s a representative, i.e., $s \in Rep$?
2. Is $(s, r) \in \xi$?

First we argue that the choice of C is very important on how much reduction is achieved. For example, let us consider the case when $C = G^{[1,k]}$. In this case we will only permute the first indices k indices of the states. Therefore if there exists an orbit $\Theta \subseteq S$ which does not involve these k indices, then $\Theta \subseteq Rep$, i.e., the whole orbit has to be represented. We need a way of choosing C such that we have uniform savings across all orbits of S in G. One possibility is to take a subgroup $H \leq G$ such that $[G : H]$ is polynomial in n. Let B be the full right traversal of H in G. Let C be the inverse closure of B. The subgroup H can be chosen in many ways and its choice will depend upon the structure of the system being verified. One choice is as follows: Fix a set $I = \{i_1, \cdots, i_k\} \subseteq [n]$ of size k. Let H be the pointwize stabilizer of I in G. The *pointwise stabilizer* of I in G is the following group:

$$\{\sigma \in G \mid \forall(i \in I)(\sigma(i) = i)\}$$

Notice that $[G : H] \leq n^k$. The subgroup H and its full traversal in G can be found in polynomial time [6].

6 Empirical Results

Using the multiple representatives theory presented in section 5.2 we have built a symbolic model checker called `SYMM`. The tool `SYMM` has a language based on the shared variable model of computation described earlier. It allows the user to give CTL specifications. The tool also provides the facility for the user to give symmetries of the system. We have verified several examples using our tool. Here we describe our results for an arbiter example. In this arbiter, each module has a priority number. While competing for the bus, modules with higher priority number are given preference. Among competing modules with the same priority number, the winner is decided non-deterministically. Winner is the module which wins the competing phase. The arbitration cycle (which results in a winner) has six phases. We give a brief overview of these phases. The interested reader is referred to the IEEE Futurebus standard for a more detailed account [9].

Phase 0: In this phase modules decide to compete for the bus.
Phase 1: Noticing that a competition phase is about to begin, other modules might decide to compete.
Phase 2: In this phase a winner (called the *master elect*) is selected from the set of competing modules. Winner is selected according to the rules described earlier.
Phase 3: Other modules check that the master elect had the highest priority number among the competing modules. If this is not the case, modules assert an error. If an error has not occurred, this phase continues until the master of the bus relinquishes its control. In this phase a module with higher priority than the master elect might start a new arbitration cycle. This is called *deposing* the master elect.
Phase 4: In this phase the current master of the bus might inhibit transfer of control of the bus to the master elect. If the master relinquishes its control over the bus, the arbitration cycle moves to the last phase.
Phase 5: In this phase the master elect gets control of the bus. This phase is called the *transfer of tenure* phase.

There are three boolean variables in each module which ensures the proper sequencing of the phases in an arbitration cycle. The internal state of the module depends on the outcome of the arbitration cycle and is shown below.

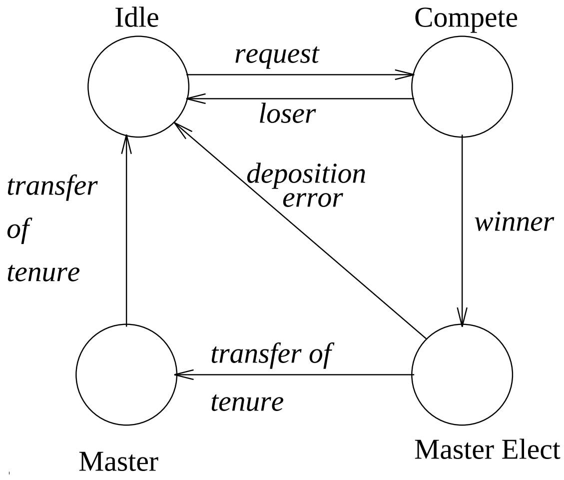

Figure 1: States of the Arbiter

Notice that two modules with the same priority number can be permuted without changing the behavior of the system. Formally, the permutation corresponding to exchanging two modules with the same priority number is a symmetry of the system. However, two modules with different priority numbers cannot be exchanged. In the table given below the first column shows the configuration of the system. Assume that we have m sets of n modules (denoted by $\{M_{1,1}, \cdots, M_{1,n}\}, \cdots, \{M_{m,1}, \cdots, M_{m,n}\}$). Moreover, the priority numbers of two modules $M_{j,l}$ and $M_{r,t}$ are the same iff $j = r$, or they belong to the same set. In this case the set of representatives *Rep* is the set of states where the first module from a particular set is always chosen master elect. The set C contains all the transpositions of the form $\sigma_{i,l} = ((i,1),(i,l))$ $(1 \leq i \leq m$ and $1 \leq l \leq n)$ and their products. The transposition $\sigma_{i,l}$ corresponds to exchanging modules $M_{i,1}$ and $M_{i,l}$. It is easy to see that C is inverse closed. Moreover, given an arbitrary state s there exists a representative state $r \in Rep$ and $\sigma \in C$ such that $\sigma(s) = r \in Rep$. Consider a state s where $M_{j,l}$ is the master elect. In this case $\sigma_{i,l}(s)$ is the state where $M_{j,1}$ is the master elect. Hence all conditions for applying the multiple representative theory are satisfied. The table of experimental results is shown below.

System Config	Time	BDD size	Time (Symm)	BDD size (Symm)
10m	163.41	369,705	37.04	27,671
12m	487.56	921,034	43.12	36,135
10m10m	1171.85	932,429	126.17	73,514
12m12m	-	-	198.195	93,094

The property that was checked was that only one module is picked master elect. The first column shows the system configuration. For example, 10m is the configuration with 10 modules having the same priority number. 10m10m denotes the configuration where the first 10 modules have a higher priority number than the last 10 modules (there are 20 modules in all). Columns 2 and 3 shows the time (in seconds) and the maximum BDD size encountered during checking the temporal property without using symmetry. The experiments were run on an Sun ULTRA SPARC. The last column shows the same statistic with using symmetry. The symbol - means that we were unable to check the property in the time allowed. As can be seen from the table we get considerable savings by using symmetry.

7 Conclusion

In this paper we investigated various problems associated with exploiting symmetry in model checking. We also provided ways of deriving symmetry for shared variable concurrent programs. An important research problem will be to take some existing hardware description languages and derive symmetry information statically from the system descriptions written in that language. Ideas presented in section 3 are applicable in this context. This paper also makes connection between exploiting symmetry in model checking and computational group theory. An important research direction will be to use some of the powerful techniques available in the computational group theory literature in the model checking domain. It will be also very useful to apply some of the symmetry ideas on industrial size examples.

References

[1] L. Babai and E. Luks. Canonical labeling of graphs. In *Proceedings of the 15th ACM STOC*, 1983.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. 98(2):142–170, June 1992.

[3] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst.*, 8(2):244–263, Apr. 1986.

[5] E. Emerson and A. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–130, 1996.

[6] M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutations groups. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, 1980.

[7] C. Hoffman. *Group Theoretic Algorithms and Graph Isomorphism*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1982.

[8] T. Hungerford. *Algebra*. Springer-Verlag, 1980.

[9] IEEE Computer Society. *IEEE Standard for Futurebus+—Logical Protocol Specification*, Mar. 1992. IEEE Standard 896.1–1991.

[10] C. Ip and D. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–76, 1996.

[11] N. Jacobson. *Basic Algebra*. W.H. Freeman and Company, 1985.

[12] K. Jensen. Condensed state spaces for symmetrical coloured petri nets. *Formal Methods in System Design*, 9(1/2):7–40, 1996.

[13] E. Luks. Permutation groups and polynomial-time computation. In *Workshop on Groups and Computation*, volume 11 of *Dimacs*. American Mathematical Society, Oct. 1991.

[14] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.

A Background

A.1 Temporal Logic

Let AP be a set of atomic propositions. A Kripke structure over AP is a triple $M = (S, R, L)$, where

- S is a finite set of *states*,
- $R \subseteq S \times S$ is a *transition relation*, which must be total (i.e., for every state s_1 there exists a state s_2 such that $(s_1, s_2) \in R$).
- $L : S \to 2^{AP}$ is a *labeling function* which associates with each state a set of atomic propositions that are true in the state.

There are two types of formulas in $CTL^\star$: *state formulas* (which are true in a specific state) and *path formulas* (which are true along a specific path). The state operators in $CTL^\star$ are: **A** ("for all computation paths"), **E**("for some computation paths"). The path operators in $CTL^\star$ are: **G**("always"), **F**("sometimes"), **U**("until"), and **V** ("unless"). Let AP be a set of atomic propositions. A state formula is either:

- p, if $p \in AP$;
- if f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas;
- if f is a path formula, then $\mathbf{E}(f)$ is a state formula.

A path formula is either:

- a state formula;
- if f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}\, f$, $f\, \mathbf{U}\, g$, and $f\, \mathbf{V}\, g$ are path formulas;

$CTL^\star$ is the set of state formulas generated by the above rules.

We define the semantics of $CTL^\star$ with respect to a Kripke structure $M = (S, R, L)$. A *path* in M is an infinite sequence of states $\pi = s_0, s_1, ...$ such that, for every $i \geq 0, (s_i, s_{i+1}) \in R$. π^i denotes the *suffix* of π starting at s_i. The i-th state on the path π is denoted by $\pi[i]$. We use the standard notation to indicate that a state formula f holds in a structure. $M, s \models f$ means that f holds at the state s in the structure M. Similarly, $M, \pi \models f$ means that the path formula f is true along the path π. Assume that f_1 and f_2 are state formulas and g_1 and g_2 are path formulas, then the relation $\models$ is defined inductively as follows:

1. $s \models p \Leftrightarrow p \in L(s)$

2. $s \models \neg f_1 \Leftrightarrow s \not\models f_1$

3. $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1 \, \text{or} \, s \models f_2$

4. $s \models \mathbf{E}(g_1) \Leftrightarrow$ there exists a path π starting with s such that $\pi \models g_1$

5. $\pi \models f_1 \Leftrightarrow s$ is the first state of π and $s \models f_1$

6. $\pi \models \neg g_1 \Leftrightarrow \pi \not\models g_1$

7. $\pi \models g_1 \vee g_2 \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$

8. $\pi \models \mathbf{X}\, g_1 \Leftrightarrow \pi^1 \models g_1$

9. $\pi \models g_1 \,\mathbf{U}\, g_2 \Leftrightarrow$ there exists $k \geq 0$ such that $\pi^k \models g_2$ and for all $0 \leq j < k, \pi^j \models g_1$.

10. $\pi \models g_1 \,\mathbf{V}\, g_2 \Leftrightarrow$ for every $k \geq 0$ if $\pi^j \not\models g_1$ for all $0 \leq j < k$, then $\pi^k \models g_2$.

CTL is the subset of $CTL^\star$ in which the path formulas are restricted to be:

- if f and g are state formulas, then $\mathbf{X}\, f$, $f \,\mathbf{U}\, g$, and $f \,\mathbf{V}\, g$ are path formulas.

The basic modalities of CTL are $\mathbf{EF}\, f$, $\mathbf{EG}\, f$, and $\mathbf{E}(f \mathbf{U} g)$, where f and g are again CTL formulas. The operators $\mathbf{AG}\, f$, $\mathbf{AF}\, f$ and $\mathbf{A}(f \mathbf{U} g)$ can be expressed in terms of the basic modalities described above.

A.2 Basic Group Theory

A *group* G is a set S endowed with an associative binary operation "." with the following properties:

1. There exists an identity element $1 \in S$ such that for all $s \in S$, $s.1 = s$.

2. Each element $s \in S$ has an unique inverse s^{-1} such that $s.s^{-1} = 1$.

The number of elements in S is called the *order* of the group. The symbol $[n]$ denotes the set $\{1, 2, \cdots, n\}$. S_n is the group of all permutations of the set $[n]$. A *cycle* of length k (denoted by $(i_0, \cdots, i_{k-1})$) is a permutation σ such that $\sigma(i_j) = i_{j+1 \pmod k}$. A *rotation group* (or *cyclic group* C_n) acting on the set $[n]$ is a permutation group generated by the cycle $(1, 2, \cdots, n)$. $Sym(X)$ denotes the group of all permutations on the set X. A group G is said to *act* on the set S if there exists a map $(g, x) \rightarrow gx$ from $G \times S$ into S satisfying

1. $1(x) = x$ for all $x \in S$.

2. $(g_1 g_2)(x) = g_1(g_2(x))$.

Notice that we write the image of (g, x) under the map as $g(x)$. Let $\langle g_1, \ldots, g_k \rangle$ be the smallest permutation group containing all the permutations $g_1, \ldots, g_k$. If $G = \langle g_1, \ldots, g_k \rangle$, then we say that the group G is *generated* by the set $\{g_1, \ldots, g_k\}$ and g_i $(1 \leq i \leq k)$ are called the *generators*. If H is a subgroup of G, we denote it by $H \leq G$.

We define certain operations on groups which will be used in the paper. Two groups G and H acting on a set S are called *disjoint* iff for all $\sigma \in G$ and all $\psi \in H$, $\sigma(i) \neq i$ implies that $\psi(i) = i$, and $\psi(i) \neq i$ implies that $\sigma(i) = i$. Intuitively, G and H act on different parts of S. *Product* of G and H (denoted by $G \cdot H$) is the group generated the set

$$\{\psi.\sigma \mid \psi \in G \ \wedge \ \sigma \in H\}$$

Disjoint product is product of two disjoint groups.

Given a group G acting on $[n]$ and a set $Y \subseteq [n]$, the *stabilizer of* Y in G (denoted by G_Y) is the group given below.

$$\{\sigma \in G \mid \sigma(Y) = Y\}$$

Definition A.1 First, we give an informal definition of **wreath product** of two groups G and H denoted by $G \wr H$. Let $G \leq S_n$ and $H \leq S_m$. Take m disjoint copies of a set X of size n. The elements of $G \wr H$ work on $X_1 \cup \cdots \cup X_m$ in the following manner. First permute the elements of X_i according to a permutation in the group G and then permute the sets of X_i according to a permutation in H. Therefore, a permutation in $G \wr H$ can be written as $(g_1, \cdots, g_m, h)$ were $g_i \in G$ and $h \in H$. Formally, consider the set $[nm]$. The i-th block of integers is $\{(i-1)n+1, \cdots, (i-1)n+n\}$. A permutation $(g_1, \cdots, g_m, h)$ acts on the set $[nm]$ in the following way:

- First, apply the permutations g_i in the following manner:

$$g_i((i-1)n+j) \quad = \quad (i-1)n + g_i(j)$$

- Then apply the permutation h in the following way:

$$h((i-1)n+j) \quad = \quad (h(i)-1)n + j$$

Definition A.2 Let G be a group. Let $H \leq G$. Given two elements $a, b \in G$, we say that $a \equiv_r b$ if and only if $ab^{-1} \in H$. In a similar manner, $a \equiv_l b$ if and only if $a^{-1}b \in H$. It is easy to see that $\equiv_r$ and $\equiv_l$ are equivalence relations on the elements of G. The equivalence classes induced by $\equiv_r$ are called *right cosets* of G under H. Similarly, the equivalence classes induced by $\equiv_l$ are called *left cosets* of G under H.

Notice that Ha denotes the right coset containing the element a. aH denotes the left coset containing the element a. $[G : H]$ denotes the number of right cosets (which is equal to the number of left cosets [8]) of G under H. Given G and $H \leq G$, the *right traversal* of H in G is a set of $k = [G : H]$ elements $\{a_1, \cdots, a_k\}$ such that $G = \cup_{i=1}^{k} Ha_i$. Intuitively the right traversal has a representative from each right coset. *Left traversal* is defined in a similar manner using left cosets. A subgroup N of H is called a *normal subgroup* if and only if for all $a \in G$, $aNa^{-1} = N$. Given a normal subgroup $N \leq G$, the right and left cosets match, i.e., for all $a \in G$, $aN = Na$. Moreover, the set of all right or left cosets, denoted by $G \backslash N$, is a group of order $[G : N]$ under the binary operation $(aN)(bN) = (abN)$(see [8]). A group G is called *simple* if and only if G has no proper normal subgroups. For an excellent treatment of group theoretic concepts see [8, 11].

A.3 Symmetry Group

Let G be a permutation group acting on S, i.e., bijective mappings acting on the state space S of the Kripke structure M. A permutation $\sigma \in G$ is said to be a *symmetry* of M if and only if it preserves the transition relation R. More formally, σ must satisfy the following condition:

$$(\forall s_1 \in S)(\forall s_2 \in S)\,((s_1, s_2) \in R \Rightarrow (\sigma s_1, \sigma s_2) \in R)$$

The symbol $Aut(M)$ represents the group of all symmetries of a Kripke Structure M. G is a *symmetry group* for the Kripke structure M if and only if every permutation $\sigma \in G$ is a *symmetry* of M. Notice that our definition of a symmetry group does not refer to the labeling function L. Furthermore, since every $\sigma \in G$ has an inverse, which is also a symmetry, it can be easily proved that a permutation $\sigma \in G$ is a symmetry for a Kripke structure if and only if σ satisfies the following condition:

$$(\forall s_1 \in S)(\forall s_2 \in S)\left((s_1, s_2) \in R \Leftrightarrow (\sigma s_1, \sigma s_2) \in R\right)$$

It is easy to see that if every generator of the group G is a symmetry of M, then the group G is a symmetry group for M.

A.4 Quotient Models

Let G be a group acting on the set S and let s be an element of S. The *orbit* of s is the set $\theta(s) = \{t \mid (\exists\sigma \in G)(\sigma s = t)\}$. From each orbit $\theta(s)$ we pick a representative which we call $rep(\theta(s))$ with the restriction that $rep(\theta(s)) \in \theta(s)$. Intuitively, the quotient model is obtained by collapsing all the states in one orbit to a single representative state. The formal definition of the quotient model is given below.

Definition A.3 Let $M = (S, R, L)$ be a Kripke Structure and let G be a symmetry group acting on S. We define the *quotient model* $M_G = (S_G, R_G, L_G)$ in the following manner:

- The state set is $S_G = \{\theta(s) \mid s \in S\}$, the set of orbits of the states in S;
- The transition relation R_G is given by

$$R_G = \{(\theta(s_1), \theta(s_2)) | (s_1, s_2) \in R\}; \tag{1}$$

- The labeling function L_G is given by $L_G(\theta(s)) = L(rep(\theta(s)))$.

Next, we define what it means for a symmetry group G of a Kripke structure M to be an *invariance group* for an atomic proposition p. Intuitively, G is an invariance group for an atomic proposition p if and only if the set of states labeled by p is closed under the application of all the permutations of G. More formally, a symmetry group G of a Kripke structure $M = (S, R, L)$ is an invariance group for an atomic proposition p if and only if the following condition holds:

$$(\forall\sigma \in G)(\forall s \in S)(p \in L(s) \Leftrightarrow p \in L(\sigma s))$$

The theorem given below is proved in [3, 5]. The theorem states that if the temporal specification f has only invariant propositions, then f can be safely checked in the quotient model.

Theorem A.1 Let $M = (S, R, L)$ be a Kripke structure, G be a *symmetry group* of M, and h be a $CTL^\star$ formula. If G is an invariance group for all the atomic propositions p occurring in h, then

$$M, s \models h \Leftrightarrow M_G, \theta(s) \models h \tag{2}$$

where M_G is the quotient structure corresponding to M.

B Some Proofs

Proof of Lemma 4.1: It is obvious that SSC is polynomially reducible to $CSSC$. To show the other direction, let $G, \gamma, Y \subseteq [n]$ be an instance of $CSSC$. Let G^i be the subgroup of G which fixes $\{1, 2, \cdots, i\}$. We have a chain of subgroups $I = G^{n-1} \leq \cdots \leq G^1 \leq G^0 = G$. This chain of subgroups and the right traversal of G^i/G^{i+1} can be found in polynomial time [6]. We perform the following steps:

1. First, determine using SSC whether there exists a $\sigma \in G\gamma$ such that $\sigma(Y) = Y$. If the answer is **no**, stop, otherwise perform the remaining steps.

2. Let $\{\sigma_1, \cdots, \sigma_k\}$ be the right traversal of G^0/G^1. By definition, here exists a $\sigma \in G\gamma$ such that $\sigma(Y) = Y$ iff there exists $j \leq k$ such that $\sigma \in G^1\sigma_j\gamma$ and $\sigma(Y) = Y$.

3. Make k calls to SSC (using the inputs $\{G^1, \sigma_1\gamma, Y\}$, $\cdots$, $\{G^1, \sigma_k\gamma, Y\}$) find a $j \leq k$ such that there exists a $\sigma \in G^1\sigma_j\gamma$ such that $\sigma(Y) = Y$.

4. Now use $G^1, \sigma_j\gamma, Y$ and repeat the above argument using G^1 instead of G and G^2 in place of G^1.

Iteratively, descending down the chain gives us the answer after $n - 1$ steps. Since the number of elements in the right traversal of G^i/G^{i+1} is atmost n, we make less than n^2 calls to SSC $\square$

Proof of Theorem 4.1: ($OP \Rightarrow SSC$:) Let x, y and G be given. Form a partition $\{C_0^x, C_1^x\}$ of the set $[n]$ in the following manner:

$$
\begin{aligned}
i \in C_0^x \quad &\Leftrightarrow \quad x_i = 0 \\
i \in C_1^x \quad &\Leftrightarrow \quad x_i = 1
\end{aligned}
$$

Define C_0^y and C_1^y in a similar manner using the vector y. Find a permutation γ such that $\gamma(C_0^x) = C_0^y$ and $\gamma(C_1^x) = C_1^y$. In this case, $\gamma(x) = y$. Suppose there exists $\sigma \in G$ such that $\sigma(x) = y$. Since $\gamma(x) = y$, this means that $\sigma^{-1}\gamma(x) = x$. Or equivalently, $\sigma^{-1}\gamma(C_0^x) = C_0^x$ (recall that C_0^x is the set of positions of 0s in x). Therefore, there exists a $\psi \in (G\gamma)$ which stabilizes C_0^x ($\psi(C_0^x) = C_0^x$) iff $\psi^{-1}\gamma(x) = y$ or equivalently $\psi(x) = y$. Hence, OP can be reduced to SSC.

($SSC \Rightarrow OP$:) Let $Y \subseteq [n]$, G, and $\gamma \in S_n$ be an instance of the problem SSC. Define a 0-1 vector x in the following manner:

$$
\begin{aligned}
i \in Y \quad &\Leftrightarrow \quad x_i = 0 \\
i \in Y^c \quad &\Leftrightarrow \quad x_i = 1
\end{aligned}
$$

Let $B_0 = \gamma(Y)$ and $B_1 = \gamma(Y^c)$. Define the 0-1 vector y such that $y_i = 0$ if $i \in B_0$ and $y_i = 1$ if $i \in B_1$. The group G in this instance of OP is just G. By the argument given in the previous part it is obvious that a permutation $\sigma \in G\gamma$ stabilizes the set Y iff $\sigma\gamma^{-1} \in G$ satisfies $y = \sigma\gamma^{-1}(x)$. $\square$

Proof of Theorem 4.2:
We will prove that SSC is polynomially equivalent to **CIE**. First we reduce to an instance of SSC to **CIE**. Let a set Y, a group G, and a permutation $\psi \in S_n$ be give. There exists a $\sigma \in G\psi$ which stabilizes the set Y iff $\sigma \in G\psi \cap Sym(Y)Sym([n] - Y)$. Notice that any permutation in the group $Sym(Y)Sym([n] - Y)$ stabilizes the set Y. Therefore, SSC is reducible to **CIE**.

Now we reduce **CIE** to SSC. This part is bit more complicated. Given groups $A \leq S_n$, $B \leq S_n$, and a permutation $\pi \in S_n$, we want to test whether $A\pi \cap B$ is non empty. Consider the set $[n] \times [n]$ and the group $D = \{(\alpha, \beta) \mid \alpha \in A \text{ and } \beta \in B\}$. The permutations (α, β) act on (i, j) in the following manner $(\alpha, \beta)(i, j) = (\alpha(i), \beta(j))$. Let $Z = \{(i, i) | i \in [n]\}$ and $Z_\pi = \{(\pi^{-1}(i), i) | i \in [n]\}$. It is easy to see that there exists $\gamma \in D$ such that $\gamma(Z) = Z_\pi$ iff there exists $\gamma_1 \in D$ such that $\gamma_1(Z_\pi) = Z$ (γ^{-1} can serve as γ_1) iff there exists a $\sigma \in A\pi \cap B$. Now consider the group $H = D \wr C_2$ (C_2 is the cyclic group acting on the set [2]), the only permutation in C_2 is $(1, 2)$). H acts on two copies of the set $[n] \times [n]$. Let Z_1 be the set corresponding to Z in the first copy and Z_2 be the set corresponding to Z_π in the second copy of $[n] \times [n]$. Let $\psi = (e, e, (1, 2)$ be the permutation in G (e is the identity permutation). Let $G \leq H$ be all permuations of the form (α_1, α_2, e) such that $\alpha_1, \alpha_2 \in D$. Basically, in G we are not allowed to switch the two copies of the set $[n] \times [n]$. Now it is easy to see that $\sigma \in G\psi$ has the form $(\alpha_1, \alpha_2, (1, 2))$. There exists a permutation $\sigma = (\alpha_1, \alpha_2, (1, 2))$ which setwize stabilizes $Z_1 \cup Z_2$ iff $\alpha_1(Z) = Z_\pi$ iff $A\pi \cap B$ is non-empty. □.

Proof of Lemma 5.4: Consider a graph $K = ([n], E)$ with n vertices. Let S be the set of transpositions generating the group G. The edge $\langle i, j \rangle \in E$ iff $(i, j) \in S$. Let $C_1, C_2, \cdots, C_m$ be the connected components of the graph K. We will prove that each connected component corresponds to a full-symmetric group acting on the vertices in that connected component. Without loss of generality let $\{1, \cdots, k\}$ be the vertices in the connected component C_1 (otherwise we can rename the vertices). We will prove that $(1, r) \in G$ for $2 \leq r \leq k$. Let $\langle 1, i_1 \rangle, \cdots, \langle 1, r \rangle$ be the path from vertex 1 to r. Composing all the transpositions along the path, we get that $(1, r) \in G$. Since $\{(12), (13), \cdots, (1k)\}$ generates the full symmetric group S_k, the connected component C_1 corresponds to S_k. Therefore, G is the disjoint direct product of m full symmetric groups (one group for each connected component). Now using lemma 5.3 and lemma 5.1 we get that COP for G can be solved in polynomial time. □