Symmetry Reduction Techniques for Explicit-State Model Checking

Alastair F. Donaldson* and Alice Miller

Department of Computing Science
University of Glasgow, 17 Lilybank Gardens,
Glasgow, Scotland, G12 8QQ.
{ally,alice}@dcs.gla.ac.uk

Abstract. We present a survey of our recent work on symmetry reduction techniques for explicit-state model checking, addressing the problems of automatic symmetry detection before search, and efficient exploitation of symmetry during search. We describe TopSPIN, an implementation of our techniques for the SPIN model checker, and discuss a user study assessing the feasibility of our approach.

1 Introduction

Model checking concurrent systems comprised of replicated components can potentially be made easier by exploiting symmetry induced by the replication. If such component symmetry can be identified before search then the model checking algorithm can be modified to consider a single state from each equivalence class of symmetric states. This results in reduced space requirements for verification by model checking. Symmetry reduction can potentially speed up model checking if an efficient procedure is available to determine whether or not a given state is equivalent to a previously reached state.

We give an overview of techniques for automatic symmetry detection and efficient exploitation of symmetries in explicit-state model checking. Symmetry detection is via analysis of the static channel diagram of a message passing concurrent system specification [4]. Computational group theoretic techniques are then used to analyse the structure of the resulting symmetry group so that a symmetry reduction strategy for efficient model checking can be chosen [6]. We describe our implementation, TopSPIN [5], a symmetry reduction package for the SPIN model checker [14]. TopSPIN requires that a Promela specification satisfies certain restrictions, and we discuss a user study which assesses the feasibility of these restrictions in practice [7].

2 Symmetry in Model Checking

Model checking a concurrent system involves first specifying the system in some high level language (e.g. Promela [14]). This high level specification can then

* Supported by the Carnegie Trust for the Universities of Scotland.
be expanded to a Kripke structure $\mathcal{M} = (S, s_0, R, L)$, where $S$ is a finite set of states with initial state $s_0$, $R \subseteq S \times S$ a transition relation, and $L : S \rightarrow 2^{AP}$ a labelling function. The set $AP$ of atomic propositions refer to the values of local and global variables, and contents of buffered channels.

A bijection $\alpha : S \rightarrow S$ which satisfies, for all $(s, t) \in R$, $(\alpha(s), \alpha(t)) \in R$, is an automorphism or symmetry of $\mathcal{M}$. The set of all symmetries form a group $\text{Aut}(\mathcal{M})$ under composition of mappings. If a subgroup $G$ of $\text{Aut}(\mathcal{M})$ is known in advance then model checking can be performed over a quotient Kripke structure, $\mathcal{M}_G$, typically smaller than the original. For $s \in S$, $[s]_G = \{\alpha(s) : \alpha \in G\}$ is the orbit of $s$ under $G$. It is standard to construct the quotient Kripke structure with respect to $G$ by taking $S_G$ (the set of quotient states) to be $\{\text{min}[s]_G : s \in S\}$. Thus, given a state $s$, it is essential to be able to compute $\text{min}[s]_G$ efficiently. This is known as the constructive orbit problem, and is NP-hard [2]. For a survey of symmetry reduction techniques in model checking see [12].

3 Automatic Symmetry Detection via Static Channel Diagram Analysis

Structural symmetries of a model $\mathcal{M}$ are typically inferred by extracting a communication graph from the associated high level specification. Practical examples of communication graphs include the coloured hypergraph of a shared variable concurrent program [2], and the static channel diagram (SCD) of a message passing specification [4], which we use within our framework.

The SCD of a specification is a bipartite, directed, coloured graph. The vertices are process identifiers and channel names, and are coloured according to their types in the specification. An edge from process $i$ to channel $c$ exists if there is a send statement $c!\text{msg}$ in the body of process $i$. Likewise, an edge exists from $c$ to $i$ if the body of $i$ includes a receive statement $c?\text{msg}$. Since the SCD associated with a specification is typically small, its automorphisms can be computed efficiently using a package such as saucy [3]. These automorphisms can then be used to infer automorphisms of the Kripke structure associated with the specification [4]. The TopSPIN symmetry reduction package (see Sect. 5) detects symmetries of Promela specifications via SCD analysis. For this method of symmetry reduction to be applied, a Promela specification must satisfy certain restrictions. In Sect. 6 we describe a user study to evaluate the feasibility of these restrictions.

4 Efficient Exploitation of Symmetry

Recall that efficient symmetry reduction requires a solution to the constructive orbit problem – computing $\text{min}[s]_G$ for a given state $s$ – which is NP-hard. We now outline various strategies which we use to solve this problem efficiently based on analysis of the structure of the group $G$. These are described in detail in [6].

**Enumeration** If $G$ is a relatively small group ($|G| < 100$ say) then for a state $s$, $\text{min}[s]_G$ can be computed by enumerating the elements of $G$, and returning
\( \min\{ \alpha(s) : \alpha \in G \} \). This approach is implemented in TopSPIN with two optimisations, applied simultaneously. As the operation of applying a transposition to a state is less expensive than that of applying an arbitrary permutation, a group element \( \alpha \) is expressed as a product of transpositions and \( \alpha(s) \) is computed by applying these transpositions to \( s \) in order. TopSPIN uses a stabiliser chain to enumerate the elements of \( G \). This means that each \( \alpha \in G \) need not be applied to a state \( s \) from scratch; partial images of \( s \) under coset representatives induced by the stabiliser chain may be re-used.

**Minimising Sets** Using terminology from [10], a group \( H \) is said to be **nice** if there is a small set \( X \subseteq H \) such that \( t = \min[s]_H \) iff \( \alpha(t) \geq t \forall \alpha \in X \). If \( H \) is nice with respect to a subset \( X \) then we call \( X \) a **minimising set** for \( H \). Given a minimising set \( X \) for \( G \), the element \( \min[s]_G \) can be computed by applying elements of \( X \) to \( s \) until a fixpoint is reached. TopSPIN uses this symmetry reduction strategy in cases where \( G \) is isomorphic to a fully symmetric group \( S_m \), for some \( m \leq n \), which simultaneously permutes several disjoint subsets of \( \{1, 2, \ldots, n\} \). (Such groups occur commonly in practice, e.g. a set of processes may have associated channels, so that any permutation of the processes must also permute the associated channels.) In this case a minimising set quadratic in \( n \) can be efficiently computed, even though \(|G| = n! \) [6].

**Disjoint and Wreath Products** If \( G \) can be decomposed as a disjoint or wreath product of subgroups then \( \min[s]_G \) can be computed by considering these subgroups individually [2]. We use computational group theoretic algorithms to automate the analysis of \( G \). If \( G \) can be expressed as a product of subgroups then each subgroup is analysed. A **composite** symmetry reduction strategy is obtained via this recursive process.

**Local Search** If none of the above strategies are applicable then, since enumeration is very expensive, it may be infeasible to compute \( \min[s]_G \). In this case an **approximate** symmetry reduction strategy is employed, based on hillclimbing local search using the group generators. This does not guarantee unique representatives, but is sound as at least one state per equivalence class will be stored. Though not as space-efficient as enumeration, this strategy can be considerably faster.

## 5 TopSPIN— a Symmetry Reduction Package for SPIN

In order to check properties of a Promela specification, SPIN first converts it 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 [1], 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.
First, the SCD of the Promela specification is extracted by the SymmExtractor tool [4]. The group of symmetries of the SCD, Aut(SCD), is computed using saucy [3], 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 \( G \) (as described in Sect. 4) 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.

Currently, TopSpin is limited to the verification of safety properties. We have reported encouraging experimental results which show that our symmetry reduction techniques can lead to significant savings in memory and verification time [5, 6].

6 User Evaluation via a Student Assessed Exercise

The automatic symmetry detection component of TopSpin requires that certain restrictions are satisfied by a Promela specification. These include: the use of an init process in which all processes are instantiated symmetrically within an atomic statement; restrictions on the use of channel variables; and a constrained set of allowable operations on variables which take as their values process identifiers. These restrictions aim to make automatic symmetry detection tractable and straightforward to implement, without unduly modifying the way that Promela programs are specified.

To evaluate the success of the restrictions in meeting this aim, we studied a set of example solutions to an assessed exercise from a model checking course at the University of Glasgow [7]. Following the Glasgow Ethics Code [13], we obtained permission from 17 students to analyse their exercise solutions, each of which consisted of three versions of a Promela specification for a two user telephone system. Intuitively, such a specification should exhibit one non-trivial symmetry which switches the local states of the users (and their associated channels) throughout all global states. The example specifications were small enough that symmetries of their associated models could be computed explicitly using the Spin-to-Grape tool [8]. For each specification, we compared these symmetries with those detected more efficiently by TopSpin, noted violations of the above restrictions, and documented ways in which the specification and/or TopSpin could be modified to avoid such violations.

The outcomes of the study were: a short set of modelling guidelines as part of the TopSpin documentation (to avoid the destruction of potential symmetries by asymmetric modelling, and to encourage adherence to certain restrictions); and a to-do list of extensions to the symmetry detection component of TopSpin which will allow some restrictions to be relaxed. As well as potentially improving
the usability of our symmetry reduction techniques, the study provides a novel case study in evaluating a formal methods tool with users.

7 Conclusions

We have described, at a high level, techniques for automatic symmetry detection and exploitation in explicit-state model checking, described the TopSPIN symmetry reduction package for the SPIN model checker, and discussed an evaluation of this tool with users. Future work includes making the changes directed by the evaluation; and extending TopSPIN to exploit virtual symmetry [9] – a precise notion of “almost” symmetry for model checking.

References