Combining Symmetry Reduction and Under-Approximation for Symbolic Model Checking

Sharon Barner and Orna Grumberg
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
{skeidar,orna}@cs.technion.ac.il

February 24, 2003

Abstract

This work presents a collection of methods that integrate *symmetry reduction* and *under-approximation* with *symbolic model checking* in order to reduce space and time. The main objective of these methods is *falsification*. However, under certain conditions, they can provide *verification* as well.

We first present algorithms that use symmetry reduction to perform on-the-fly model checking for temporal safety properties. These algorithms avoid building the orbit relation and choose representatives on-the-fly while computing the reachable states. We then extend these algorithms to check liveness properties as well. In addition, we introduce an iterative on-the-fly algorithm that builds subsets of the orbit relation rather than the full relation.

Our methods are fully automatic. The user should supply some basic information about the symmetry in the verified system. However, the methods are *robust* and work correctly even if the information supplied by the user is incorrect. Moreover, the methods return correct

results even when the computation of the symmetry reduction has not been completed due to memory or time explosion.

We implemented our methods within the IBM model checker RuleBase and compared their performance to that of RuleBase. In most cases, our algorithms outperformed RuleBase in both time and space.

1 Introduction

This work presents a collection of model-checking methods that integrate *symmetry reduction* and *under-approximation* with *symbolic model checking* in order to reduce space and time. The main objective of these methods is *falsification*, that is, proving that a given system does not satisfy its specification. However, under certain conditions they can also be used for *verification*, i.e., proving that the system does satisfy its specification.

Our methods are fully automatic. The user should supply some basic information about the symmetry in the verified system. However, the methods are *robust* and work correctly even if the information supplied by the user is incorrect. Moreover, the methods return correct results even when the computation of the symmetry reduction has not been completed due to memory or time explosion.

Temporal logic model checking [9] is a technique that accepts a finite state model of a system and a temporal logic specification and determines whether the system satisfies the specification. The main problem of model checking is its high memory requirements. *Symbolic model checking* [20] can be used to overcome this problem. Symbolic model checking, based on BDDs [5], can handle larger systems, but is still limited in its capacity. Thus, additional work is needed in order to make model checking feasible for larger systems.

Symmetry reduction can also be employed to reduce the memory and time requirements of symbolic model checking. This is the method we exploit in this work. Symmetry reduction is based on the observation that many systems consist of several similar components. Switching the roles of such components does not change system behavior. Thus, system states can be partitioned into equivalence classes called *orbits*, and the system can be verified by examining only representative states from each orbit.

Two main problems arise, however, when integrating symmetry reduction with symbolic model checking. One is building the orbit relation and the other is choosing a representative for each orbit. [18] proves that the BDD for the orbit relation is exponential in the number of BDD variables and suggests choosing more than one representative for each orbit in order to obtain a smaller BDD. However, this method does not solve the problem of choosing the representatives. The choice of representatives is significant because it strongly influences the size of the BDDs representing the symmetry-reduced model. [14] suggests choosing generic representatives. This approach involves

compiling the symmetric program to a reduced model over the generic states. Such a compilation can only be applied to programs written with a special syntax in which symmetry is defined inside the program. [17] introduces a DFS algorithm for explicit model checking. This algorithm chooses as a representative for an orbit the first state discovered in that orbit, thus avoiding the need to choose the representative in advance. Unfortunately, it is not applicable to symbolic model checking because performing DFS is very inefficient with BDDs.

We suggest a new approach that avoids building the orbit relation and chooses representatives on-the-fly while computing the reachable states. Unlike [17], our algorithm uses BDD criteria to guide the choice of the representatives. Reachability is performed using *under-approximation* which, at each step, explores only a subset of the reachable states. Some of the unexplored states are symmetric to the explored ones, and the symmetry information is exploited to ensure that those states will never be explored. Thus, easier symbolic forward steps are obtained.

We first apply this approach to verifying properties of the form $AG(p)$[1], where p is a boolean formula. If we find a "bad" state that does not satisfy p, we conclude that the checked system does not satisfy $AG(p)$. On the other hand, if no "bad" state is found, we cannot conclude that the system satisfies $AG(p)$. This is because reachability with under-approximation does not necessarily explore every reachable state. We next present a special version of our algorithm in which the under-approximation is guided by *hints* [3]. Under certain conditions this algorithm can also verify the system.

The algorithms described above are based on reachability and are often referred to as *on-the-fly* model checking algorithms. How to extend on-the-fly model checking for $AG(p)$ to verifying general *safety temporal properties* is well known. This is done by building an automaton describing the property and running it together with the system. We specified conditions on the automaton that also guarantee the correctness of the on-the-fly algorithm when the automaton runs together with the symmetry-reduced model. The suggested conditions hold for the tableau construction used for symbolic LTL model checking [7], when restricted to LTL safety properties. They also hold for the satellite used in symbolic model checking of RCTL formulas [2]. By running the automaton together with the reduced model, we save both space

[1] $AG(p)$ means that p holds along every path, in every state on the path.

and time when verifying such formulas.

On-the-fly model checking cannot handle liveness properties. In order to handle them, we developed two extensions that combine symmetry reduction with classical (not on-the-fly) symbolic model checking. One is easy to perform and is mainly suitable for falsification. The other is more expensive but can handle verification as well.

Another approach to the orbit relation problem is to build the orbit relation iteratively. We present an on-the-fly symbolic algorithm in which only a subset of the orbit relation is built in each iteration. The quotient model is constructed according to this subset and the property is checked on the quotient model. This algorithm may take more iterations to find a "bad" state than do on-the-fly algorithms with symmetry reduction. However, since it keeps the BDDs which represent the model small, it may terminate where the on-the-fly algorithms would explode.

Previous works expect the user to provide a symmetry group that is also an invariance group [18]. In many cases two formulas checked on the same model require different invariance groups because each formula breaks the symmetry of the model differently. Thus, the user needs to supply different invariance groups for different formulas. In other works [22, 10], the program is written in a special syntax by which the invariance group can be found. In these cases only formulas which do not break the symmetry of the model are allowed.

In contrast, we build the invariance group automatically, once the symmetry group has been given. Supplying the symmetry group usually requires only a high-level understanding of the system and therefore is easier than supplying the invariance group.

We implemented our methods within the enhanced model checking tool RuleBase [1], developed by the IBM Haifa Research Laboratories, and compared the performance of our methods with that of RuleBase. Experiments show that our methods performed significantly better, with respect to both time and space, in checking liveness properties. For temporal safety properties one method performed better with respect to time. However, its space requirements were worse for small examples and identical for larger ones. The other method performed significantly better, with respect to both time and space, in special cases.

The rest of the paper is organized as follows. Section 2 gives some basic definitions. Section 3 shows how to build the invariance group. Section 4

presents an algorithm for on-the-fly symbolic model checking with symmetry reduction and then introduces hints into this algorithm. Sections 6 and 7 handle temporal safety properties and liveness properties, respectively. Section 8 presents an iterative algorithm for on-the-fly symbolic model checking which builds a subset of the orbit relation in each iteration, and Section 9 presents our experimental results. We conclude in Section 10 with directions for future research.

2 Preliminaries

2.1 Temporal logic - CTL^*, $ACTL^*$

CTL^* is a powerful temporal logic. In this work we define CTL^* in negation normal form in which negations are applied only to atomic propositions. CTL^* is defined over a set of atomic propositions AP. There are two types of formulas in CTL^*: state formulas and path formulas, defined as follows:

- state formulas:
  - If $p \in AP$, p and $\neg p$ are state formulas.
  - If φ_1 and φ_2 are state formulas, then $\varphi_1 \vee \varphi_2$ and $\varphi_1 \wedge \varphi_2$ are state formulas.
  - If φ_1 is a path formula, then $E\varphi_1$ and $A\varphi_1$ are state formulas.
- path formulas:
  - If φ_1 is a state formula, then φ_1 is also a path formula.
  - If φ_1 and φ_2 are path formulas, then $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ $X\varphi_1$, $F\varphi_1$, $G\varphi_1$, $\varphi_1 U \varphi_2$, and $\varphi_1 R \varphi_2$ are path formulas.

CTL^* is the set of state formulas generated by these rules.

The semantics of CTL^* are defined with respect to a Kripke structure M=(S,S_0,R,L), where S is a finite set of states, $R \subseteq S \times S$ is a total transition relation, L is a labeling function which labels each state with the set of atomic propositions AP true in that state, and S_0 is the set of initial states. A path π in a Kripke structure M is an infinite sequence of states, $s_0, s_1, s_2, \ldots$, in which $\forall i \geq 0$ $(s_i, s_{i+1}) \in R$. π^i denotes the suffix of π starting at s_i. For a

state formula φ $M, s \models \varphi$ indicates that φ is true in state s, and for a path formula ψ $M, \pi \models \psi$ indicates that ψ is true along π.

Assuming that φ_1 and φ_2 are state formulas and ψ_1 and ψ_2 are path formulas, the relation $\models$ is defined as follows:

- $M, s \models p \Leftrightarrow p \in L(s)$.
- $M, s \models \neg p \Leftrightarrow M, s \not\models p$.
- $M, s \models \varphi_1 \vee \varphi_2 \Leftrightarrow M, s \models \varphi_1$ or $M, s \models \varphi_2$.
- $M, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M, s \models \varphi_1$ and $M, s \models \varphi_2$.
- $M, s \models E\psi_1 \Leftrightarrow$ there exists a path $\pi = s_0, s_1, \ldots$ in M such that $s_0 = s$ and $M, \pi \models \psi_1$.
- $M, s \models A\psi_1 \Leftrightarrow$ for every path $\pi = s_0, s_1, \ldots$ in M such that $s_0 = s$, $M, \pi \models \psi$.
- $M, \pi \models \varphi_1 \Leftrightarrow s$ is the first state of π and $M, s \models \varphi$.
- $M, \pi \models \psi_1 \vee \psi_2 \Leftrightarrow M, \pi \models \psi_1$ or $M, \pi \models \psi_2$.
- $M, \pi \models \psi_1 \wedge \psi_2 \Leftrightarrow M, \pi \models \psi_1$ and $M, \pi \models \psi_2$.
- $M, \pi \models X\psi_1 \Leftrightarrow M, \pi^1 \models \psi_1$.
- $M, \pi \models F\psi_1 \Leftrightarrow \exists k \geq 0\ M, \pi^K \models \psi_1$.
- $M, \pi \models G\psi_1 \Leftrightarrow \forall i \geq 0\ M, \pi^i \models \psi_1$.
- $M, \pi \models \psi_1 U \psi_2 \Leftrightarrow \exists k \geq 0\ M, \pi^k \models \psi_2$ and $\forall 0 \leq i < k\ M, \pi^i \models \psi_1$.
- $M, \pi \models \psi_1 V \psi_2 \Leftrightarrow \forall k \geq 0$ if $\forall i < k\ M, \pi^i \not\models \psi_1$ then $M, \pi^k \models \psi_2$.

A structure M satisfies formula φ ($M \models \varphi$) if every initial state $s \in S_0$ satisfies φ.

$ACTL^*$ is the sub-logic of CTL^* in which all formulas contain only universal path quantifiers.

A CTL^* formula is *boolean* if it contains only atomic propositions and boolean operators.

Definition 2.1 *A formula β is a* maximal boolean subformula *of a formula φ if β is a boolean subformula of φ and for all subformulas β' of φ, if β is a subformula of β' then β' is not boolean.*

2.2 BDDs

A Binary Decision Diagram (BDD) [5] is a data structure for representing boolean functions. BDDs are defined over boolean variables. They are often (but not always) concise in their memory requirement, and most boolean operations can be performed efficiently on BDD representations. In [20] it has been shown that BDDs can be very useful for representing Kripke structures and performing model checking symbolically. Given a Kripke structure M whose set of states is to be represented by BDDs, we represent each state s of M by a valuation of a vector of BDD variables $\bar{v}$. We also associate with each BDD variable v_i an additional variable v_i'. We can now represent the transition relation $R \subseteq S \times S$ by the BDD $R(\bar{v}, \bar{v}')$.

2.3 Model Checking

Model checking is a technique for verifying finite state systems. The state space of the system is exhaustively searched to determine if it satisfies a given specification [9]. The model under verification is usually described by a Kripke structure M and the specification by temporal logic formulas. The set of atomic propositions AP_φ of the formula φ under evaluation is a subset of the set of atomic propositions AP of the model M. Since the number of states grows exponentially with the number of state variables, it is very difficult to verify large models. This problem is called the *state explosion problem*. Symbolic model checking [6] uses BDDs [5] to implement model checking algorithms. It thus makes model checking applicable to larger designs, although this applicability is still limited by space requirements.

2.4 On-the-Fly Symbolic Model Checking

On-the-fly symbolic model checking [2] is a method which checks a given formula φ while computing the reachable states of the model. Most of the constructions for on-the-fly symbolic model checking build an automaton A_φ and a formula $\psi = \mathrm{AG(p)}$, where p is a boolean formula. The construction

guarantees that $M \times A_\varphi \models \psi$ if and only if $M \models \varphi$. Since $\psi = \mathrm{AG(p)}$, verifying ψ amounts to computing the reachable states while checking that each state satisfies p. If a state which falsifies p is found, then there is no need to complete the reachability computation. If all reachable states satisfy p, then $M \models \varphi$. One of the most useful operations in model checking and on-the-fly model checking in particular is *image computation*. Given a set of states S and a relation A, represented by the BDDs $S(\bar{v})$ and $A(\bar{v}, \bar{v}')$ respectively, the image computation finds the set of all states related by A to some state in S.

Definition 2.2 $Im_A(S(\bar{v})) = \exists \bar{v}(S(\bar{v}) \wedge A(\bar{v}, \bar{v}'))$.

2.5 Partial Search

While symbolic model checking can be very efficient, it can still suffer from explosion in the BDD size. One solution is to perform a partial search of the reachable state space while avoiding large BDDs [21]. Other methods perform a partial search guided by the user [3] or by the checked specification [23]. In all methods the set of reachable states discovered in each step is an under-approximation of the set of reachable states which would have been discovered in BFS. This property enables combining partial search with on-the-fly model checking.

2.6 Bisimulation and Simulation Relation

The goal of our work is to exploit symmetry for producing smaller models that are easier to model check. It is important that the smaller models will preserve properties of the original models. To formalize these ideas we define two relations over models: the bisimulation and the simulation relations.

Definition 2.3 *Let M and M' be two Kripke structures over the same set of atomic propositions AP. A relation $B \subseteq S \times S'$ is a* bisimulation relation *between M and M' for a set of boolean formulas BS over AP if for every initial state s_0 of M there is an initial state s'_0 of M' such that $(s_0, s'_0) \in B$, and for every initial state s'_0 of M' there is an initial state s_0 of M such that $(s_0, s'_0) \in B$. Moreover, if $(s, s') \in B$, then*

1. $\forall \beta \in BS\ [s \models \beta \Leftrightarrow s' \models \beta]$.

2. $\forall s_1[(s,s_1) \in R \Rightarrow \exists s'_1[(s',s'_1) \in R' \wedge (s_1,s'_1) \in B]]$.

3. $\forall s'_1[(s',s'_1) \in R' \Rightarrow \exists s_1[(s,s_1) \in R \wedge (s_1,s'_1) \in B]]$.

M and M' are *bisimulation equivalent for BS* (denoted $M \equiv_{bis} M'$) if there is a bisimulation relation B for BS between M and M'. The following lemma is immediate from the result in [4].

Lemma 2.1 *For every CTL^* formula φ over BS and two Kripke structures M, M' over AP, if $M \equiv_{bis} M'$, then $M' \models \varphi \Leftrightarrow M \models \varphi$.*

Definition 2.4 *Let M and M' be two Kripke structures over the same set of atomic propositions AP. A relation $SIM \subseteq S \times S'$ is a* simulation relation *between M and M' for a set of boolean formulas BS over AP if for every initial state s_0 of M there is an initial state s'_0 of M' such that $(s_0,s'_0) \in SIM$. Moreover, if $(s,s') \in SIM$, then*

1. $\forall \beta \in BS\ [s \models \beta \Leftrightarrow s' \models \beta]$.

2. $\forall s_1[(s,s_1) \in R \Rightarrow \exists s'_1[(s',s'_1) \in R' \wedge (s_1,s'_1) \in SIM]]$.

M is smaller than M' for BS by the simulation relation (denoted $M \leq_{sim} M'$) if there is a simulation relation for BS $SIM \subseteq S \times S'$. The following lemma is immediate from the result in [16].

Lemma 2.2 *For every $ACTL^*$ formula φ over BS and two Kripke structures M, M' over AP, if $M \leq_{sim} M'$ then $M' \models \varphi \Rightarrow M \models \varphi$.*

Lemma 2.2 is also true for Kripke structures with finite paths. See appendix A for a full definition of CTL^* over Kripke structures without a total transition relation and the proof of Lemma 2.2 on these structures.

2.7 The Product Model and the Restricted Model

We now define two special Kripke structures that will be used later.

Definition 2.5 *Let M, M' be two Kripke structures defined over the sets of atomic propositions, AP and AP', respectively. The* product structure *of M and M' is a Kripke structure over $AP \cup AP'$, defined as follows. $M \times M' = (S_{M\times M'}, S^0_{M\times M'}, R_{M\times M'}, L_{M\times M'})$ where*

- $S_{M\times M'} = \{\ (s,s') \mid s \in S \wedge\ \ s' \in S' \wedge L(s) \cap AP' = L'(s') \cap AP\}$.
- $S^0_{M\times M'} = \{\ (s,s') \mid (s,s') \in S_{M\times M'}\ \wedge\ \ s \in S_0\ \ \wedge\ \ s' \in S'_0\ \}$.
- $\forall (s,s'),(t,t') \in S_{M\times M'}$
  $[((s,s'),(t,t')) \in R_{M\times M'} \Leftrightarrow (s,t) \in R \wedge (s',t') \in R']$.
- $\forall (s,s') \in S_{M\times M'}[L((s,s')) = L(s) \cup L'(s')]$.

Definition 2.6 *Let M be a Kripke structure and A be a subset of S. The* restricted model $M|_A = (S|_A, S_0|_A, R|_A, L|_A)$ *is defined as follows:*

- $S|_A = A$.
- $S_0|_A = S_0 \cap A$.
- $\forall s,s' \in S|_A\ [(s,s') \in R|_A \Leftrightarrow (s,s') \in R]$.
- $\forall s \in S|_A\ [L|_A(s) = L(s)]$.

Lemma 2.3 *For every Kripke structure M and $A \subseteq S$, $M|_A \leq_{sim} M$.*

Proof: The relation $B = \{(s,s)|s \in A\}$ is a simulation relation between M and M'. ∎

2.8 Symmetry

Definition 2.7 *A permutation on a set A, $\sigma : A \to A$ is a one-to-one and onto function.*

For a set $A' \subseteq A$, $\sigma(A') = \{a|a \in A \wedge \exists a' \in A'\ \sigma(a') = a\}$. In this paper we use permutations over the set of states of a Kripke structure. Given a CTL^* formula β and a structure M, $\sigma(\beta)$ refers to applying σ to the set of states in M that satisfy β.

Definition 2.8 *A permutation that maps $a_1 \to a_2, a_2 \to a_3, \ldots, a_{k-1} \to a_k, a_k \to a_1$ is called a* cycle *and is denoted by $(a_1 a_2 \ldots a_k)$.*

Any permutation can be written as a composition of disjoint cycles $\sigma = c_1 c_2 \ldots c_m$.

Definition 2.9 *A permutation group G is a set of permutations together with the composition operation such that:*

- *The identity permutation e is in G.*
- *For every permutation $\sigma \in G$, there is a permutation $\sigma' \in G$ such that $\sigma\sigma' = e$. Such a σ' is usually called σ^{-1}.*
- *For every $\sigma_1, \sigma_2 \in G$, $\sigma_1\sigma_2$ is also in G.*

Definition 2.10 *$\sigma_1, \sigma_2, \ldots, \sigma_k$ are* generators *of permutation group G (denoted $G = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$) if G is the closure of the set $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ under the composition operation.*

Definition 2.11 *A permutation group G is a* symmetry group *of a Kripke structure M if every permutation $\sigma \in G$ preserves the transition relation and the initial states. That is, $\forall s, s' \in S$ $[(s, s') \in R \Leftrightarrow (\sigma(s), \sigma(s')) \in R$ and $s \in S_0 \Leftrightarrow \sigma(s) \in S_0]$.*

To exemplify the notations above, we consider a Kripke structure M constructed by three identical processes, p_1, p_2, p_3. Each process i uses the set of variables $\bar{v}_i$ and each state is over the set of variables $\bar{v} = \bar{v}_1\bar{v}_2\bar{v}_3$. The permutation which exchanges the values of $\bar{v}_i$ and $\bar{v}_j$ (each variable with its identical variable in the other process) will be denoted (p_i, p_j). Since the processes are identical, exchanging the values of $\bar{v}_i$ and $\bar{v}_j$ for every i and j will preserve the transition relation and the initial states. Thus the group $G = \{e, (p_1, p_2), (p_2, p_3), (p_1, p_3), (p_1, p_3, p_2), (p_1, p_2, p_3)\}$ is a symmetry group of M and the permutations $\{(p_1, p_2), (p_2, p_3)\}$ are a set of generators of G.

Definition 2.12 *Let BS be a set of boolean formulas. A symmetry group G of a Kripke structure M is an* invariance group *w.r.t. BS if for every boolean formula $\beta \in BS$, every $\sigma \in G$ and every $s \in S$ $[M, s \models \beta \Leftrightarrow M, \sigma(s) \models \beta]$.*

Definition 2.13 *A boolean formula β* preserves the symmetry *of a permutation σ if for every $s \in S$, $[M, s \models \beta \Leftrightarrow M, \sigma(s) \models \beta]$.*

Definition 2.14 *A boolean formula β* breaks the symmetry *of a permutation σ if β does not preserve the symmetry of σ.*

When BS is the set of atomic propositions of formula φ, our definition of an invariance group is identical to the definition of an invariance group in [18]. However, we prefer the BS set of the maximal subformulas of φ (see definition 2.1). This is because an invariance group defined w.r.t. the maximal subformulas may contain more permutations than an invariance group defined w.r.t. the set of atomic propositions. Consider the formula $\varphi = AG(\neg(p_1\_in\_critical \wedge p_2\_in\_critical))$, which is evaluated on M defined in the previous example. The set of atomic propositions are $\{p_1\_in\_critical, p_2\_in\_critical\}$. The largest invariance group defined according to these atomic propositions is $G = \{e\}$. This is because the atomic propositions break the symmetry of the permutations which exchange p_1, p_2 and p_3. However, the formula has only one maximal boolean subformula $\neg(p_1\_in\_critical \wedge p_2\_in\_critical)$, and the largest invariance symmetry group defined according to it is $G = \{(p_1, p_2), e\}$. This is because the formula preserves the symmetry of the permutation which exchanges p_1 and p_2.

Given an invariance group G and a Kripke structure M, we can partition the states S into equivalence classes. The equivalence class of s is $[s] = \{s' | \exists \sigma \in G,\ \sigma(s) = s'\}$. Each [s] is called an *orbit* and the relation OR $= \{(s, s') | s, s' \in S \ and \ [s] = [s']\}$ is called the *orbit relation.*

For a Kripke structure $M = (S, S_0, R, L)$ and an invariance group G w.r.t. a set of boolean formulas BS, the *quotient structure* $M_G = (S_G, S_G^0, R_G, L_G)$ is defined as follows:

- $S_G = \{\ [s] \mid s \in S\ \}$.
- $S_G^0 = \{\ [s] \mid s \in S_0\ \}$.
- $R_G = \{\ ([s], [s']) \mid (s, s') \in R\ \}$.
- $L_G([s]) = \{\beta | \beta \in BS \wedge s \models \beta\}$.

In [12, 18] it has been proved that for every CTL^* formula ψ over BS, and every state $s \in S$, $M_G, [s] \models \psi \Leftrightarrow M, s \models \psi$.

The quotient structure is built by having a representative chosen from each orbit. Then a *representative function* $\xi : S \rightarrow Rep$, which maps each state s to its orbit representative, is defined. M_G is now defined by

- S_G = Rep.

- $S_G^0 = \{s | \exists s' \ \xi(s') = s \wedge s' \in S_0\}$.
- $R_G = \{ \ (s, s') \mid s, s' \in Rep \wedge \exists s'' \in S \ [R(s, s'') \wedge \xi(s'') = s']\}$.
- $L_G(s) = \{\beta | \beta \in BS \wedge s \models \beta\}$.

In many cases, however, it is easier to choose more than one representative for each orbit.

Definition 2.15 $\xi \subseteq Rep \times S$ *is a* representative relation *if for all* s, s' $(s, s') \in \xi \Leftrightarrow s \in Rep \wedge [s] = [s']$.

Definition 2.16 *The* quotient structure *for multiple representative* $M_m = (S_m, S_m^0, R_m, L_m)$ *is a Kripke structure in which:*

- $S_m = Rep$.
- $S_m^0 = \{ \ s \mid \exists s' \in S_0 \ (s, s') \in \xi\}$.
- $R_m = \xi^{-1} R \xi$.
- $L_m(s) = \{\beta | \beta \in BS \wedge s \models \beta\}$.

[18] shows that for every CTL^* formula ψ over BS and every state $s \in Rep$, $M_m, s \models \psi \Leftrightarrow M_G, [s] \models \psi$.

In our work we use a stronger property of the quotient model:

Lemma 2.4 *For every Kripke structure* M *and every invariance group* G *of* M *w.r.t* BS, $M_G \equiv_{bis} M$ *for* BS.

Proof: $B = \{([s], s) | s \in S\}$ is a bisimulation relation for BS between M_G and M. ∎

Lemma 2.5 *For every Kripke structure* M, *every invariance group* G *of* M *for* BS *and every set* $Rep \subseteq S$ *which contains at least one representative from each orbit,* $M_m \equiv_{bis} M_G$ *for* BS.

Proof: $B = \{(s, [s]) | s \in Rep\}$ is a bisimulation relation for BS between M_m and M_G. ∎

Lemmas 2.1, 2.4, and 2.5 imply that for every CTL^* formula ψ over boolean formulas in BS,

$$M \models \psi \Leftrightarrow M_G \models \psi \Leftrightarrow M_m \models \psi.$$

3 Building the Invariance Group

Previous algorithms showed how to perform model checking with symmetry when the invariance group is supplied by the user [18]. In many cases two formulas evaluated on the same model require different invariance groups because each formula breaks the symmetry of the model differently. For example, given two symmetric processes p_1 and p_2, the formula $AGAF(p_1\_in\_critical)$ breaks the symmetry between p_1 and p_2 while the formula $AG(\neg(p_1\_in\_critical \wedge p_2\_in\_critical))$ does not break the symmetry at all. This implies that the user may need to supply different invariance groups for different formulas. Supplying them may require close familiarity with system details. In other works [22, 10], the program is written in a special syntax which enables the invariance group to be found. In these cases only formulas which do not break the symmetry of the model are allowed (e.g., SCTL formulas in [22]).

We suggest a new method that, given a symmetry group G and a formula φ, automatically generates an invariance group G_{inv} w.r.t. the set of maximal boolean subformulas of a given formula φ. Our method is useful because providing a symmetry group often requires only a high-level understanding of the system and is therefore easier to supply than an invariance group.

Our method works as follows. Given a set of generators for a symmetry group G, an invariance group G_{inv} is defined by restricting the generators of G to those σ_i that satisfy $\sigma_i(\beta) = \beta$ for every maximal boolean subformula β. As a result, the orbits of G_{inv} obtained by its generators do not contain both states that satisfy β and states that do not satisfy β. This implies that G_{inv} is an invariance group. The following lemma illustrates the correctness of our approach.

Lemma 3.1 *Let $\sigma_1, \sigma_2, \ldots, \sigma_k$ be generators of a symmetry group G of a Kripke structure M and let MAX be the set of maximal boolean subformulas of a given formula φ. If $IG = \{\sigma_i | \forall \beta \in MAX,\ \sigma_i(\beta) = \beta\}$ is not empty, it generates an invariance group G_{inv} of M w.r.t. MAX.*

Proof: We first prove that if IG is not empty, it generates a permutation group G_{inv}. Any $\sigma \in G_{inv}$ can be written as a composition of disjoint cycles $\sigma = c_1 c_2 \ldots c_m$.

For all $\sigma \in IG$, $\sigma = c_1 c_2 \ldots c_m$ of length $l_1, l_2, \ldots, l_m$ respectively.

- Since IG is not empty, $\exists \sigma \in IG.\ \sigma^{l_1 l_2 \ldots l_m} = \mathrm{e}$. Since σ is a generator of G_{inv}, $\sigma^{l_1 l_2 \ldots l_m} \in G_{inv}$, which means that $e \in G_{inv}$.

- For all $\sigma' \in IG$, $\sigma'^{(l_1 l_2 \ldots l_m - 1)} = \sigma'^{-1}$. Since σ' is a generator of G_{inv}, $\sigma'^{(l_1 l_2 \ldots l_m - 1)} \in G_{inv}$, which means that $\sigma'^{-1} \in G_{inv}$. For all $\sigma \in G_{inv}$, $\sigma = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_l}$ where $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_l} \in IG$. Since $\sigma^{-1} = \sigma_{i_l}^{-1}\ldots\sigma_{i_2}^{-1}\sigma_{i_1}^{-1}$ and for all $\sigma' \in IG$, $\sigma'^{-1} \in G_{inv}$, we get that $\sigma^{-1} \in G_{inv}$.

- For all $\sigma_1, \sigma_2 \in G_{inv}$, $\sigma_1 = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_l}$ $\sigma_2 = \sigma_{j_1}\sigma_{j_2}\ldots\sigma_{j_{l'}}$, where $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_l}, \sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_{l'}} \in IG$. Since $\sigma_1\sigma_2 = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_l}\sigma_{j_1}\sigma_{j_2}\ldots\sigma_{j_{l'}}$, $\sigma_1\sigma_2 \in G_{inv}$.

We have shown that G_{inv} is a permutation group. Since its generators are a subset of the generators of the symmetry group G, G_{inv} is also a symmetry group. We still need to show that G_{inv} is an invariance group.

We first prove that for every $\sigma \in G_{inv}$ and every $\beta \in MAX$, $\sigma(\beta) = \beta$. G_{inv} is generated by IG, meaning that every $\sigma \in G_{inv}$ can be written as $\sigma = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_m}$, where $\sigma_{i_1}, \sigma_{i_2}, \ldots\sigma_{i_m} \in IG$. By the definition of IG we know that for every j, $\sigma_{i_j}(\beta) = \beta$. Thus $\sigma(\beta) = \beta$. Since $\sigma(\beta) = \beta$ for every $\sigma \in G_{inv}$, we have that for every $\beta \in MAX$, $\sigma \in G_{inv}$ and $s \in S$, $[M, s \models \beta \Leftrightarrow M, \sigma(s) \models \beta]$.

■

In our previous example, the set of generators of the symmetry group of a model with two processes is $\{\sigma\}$, where $\sigma = (p_1, p_2)$. For formula $\varphi = AGAF(p_1\_in\_critical)$, the only maximal boolean subformula is $p_1\_in\_critical$. Since $\sigma(p_1\_in\_critical) = p_2\_in\_critical$, which is not identical to $p_1\_in\_critical$, we conclude that $IG = \phi$. This implies that the largest invariance group w.r.t. the maximal boolean subformulas of φ is G = $\{e\}$. For the formula $\varphi = AG(\neg(p_1\_in\_critical \wedge p_2\_in\_critical))$, the only maximal boolean subformula is $\neg(p_1\_in\_critical \wedge p_2\_in\_critical)$. Since $\sigma(\neg(p_1\_in\_critical \wedge p_2\_in\_critical) = \neg(p_2\_in\_critical \wedge p_1\_in\_critical) = \neg(p_1\_in\_critical \wedge p_2\_in\_critical)$, IG = $\{\sigma\}$. Thus, in this case the original symmetry group is also an invariance group for φ.

Definition 3.1 *The* largest invariance group G_{inv} *with respect to a symmetry group G is an invariance group such that for every invariance group $G' \subseteq G$, $|G'| \leq |G_{inv}|$.*

For a given symmetry group G, the invariance group G_{inv} generated by IG may not be the largest invariance group that exists for M with respect to G. For example, if a symmetry group $G = \{e, (p_1, p_2), (p_2, p_3), (p_1, p_3), (p_1, p_3, p_2), (p_1, p_2, p_3)\}$ is given by the generators $\{(p_1, p_2), (p_2, p_3)\}$, and IG is calculated with respect to the formula, $\varphi = AG(\neg(p_1\_in\_critical \wedge p_3\_in\_critical))$. Since $IG = \{e\}$, we get that $G_{inv} = \{e\}$. However, the largest invariance group with respect to G is $\{(p_1, p_3), e\}$.

The largest invariance group with respect to a given symmetry group can be obtained by restricting the symmetry group G as follows: $G_{inv} = \{\ \sigma \in G \mid \forall \beta \in MAX,\ \sigma(\beta) = \beta\}$. However, the number of permutations in G may be exponential in the number of generators of G. Thus it is not practical to construct G_{inv} directly from G.
Below we explain how the construction of the invariance group can be implemented with BDDs:

- A permutation σ can be represented as a BDD by $(\bar{v} = \sigma(\bar{v}'))$. However, it is easier to represent it as a binary relation $\sigma(\bar{v}, \bar{v}')$. For example, consider a permutation $\hat{\sigma}$ defined over a set of processes $1 \ldots n$, where each process i uses the set of variables $\bar{v}_i$ and each state is over the variables $\bar{v} = \bar{v}_1 \bar{v}_2 \ldots \bar{v}_n$. If $\hat{\sigma}$ exchanges the role of processes i and j we denote it by $\hat{\sigma}(i) = j$. The permutation σ over the state variables $\bar{v}$ is induced from $\hat{\sigma}$. The BDD $\sigma(\bar{v}, \bar{v}')$ that represents σ will be defined by $\sigma(\bar{v}, \bar{v}') = \wedge_i(\bar{v}_i = \bar{v}'_{\hat{\sigma}(i)})$.

- For a boolean formula β, let $\beta(\bar{v})$ be the BDD representing the set of all states satisfying β. Then the BDD for $\sigma(\beta)$ is defined by $\sigma(\beta(\bar{v})) = \exists \bar{v}(\sigma(\bar{v}, \bar{v}') \wedge \beta(\bar{v}))$.

- We check whether $\sigma(\beta) = \beta$. With a BDD representation, this amounts to checking that the BDDs $\beta(\bar{v})$ and $\sigma(\beta(\bar{v}))$ are actually the same BDD after we unprime the BDD variables in $\sigma(\beta(\bar{v}))$.

3.1 Extension to Fairness

When the model under verification consists of fairness constraints, the sets of fair paths in the quotient model should be identical to the set of fair paths in the original model. The set of fair paths can be made identical by requiring

that the generators of the invariance group satisfy $\sigma(\beta) = \beta$ for the formulas in the fairness constraints as well as in the checked formula.

A Kripke structure M with fairness constraints is a five-tuple $M = (S, S_0, R, L, F)$ where S, S_0, R and L are defined as in section 2.1, and $F = \{f_1, \ldots, f_m\}$, where for all i, f_i is a boolean formula over a set of atomic propositions $AP_F \subseteq AP$. π is a fair path if and only if for every f_i, $inf(\pi) \cap f_i \neq \phi$ [2] where $inf(\pi) = \{s \mid s = s_j \text{ for infinitely many } j\}$. The semantics of CTL^* with respect to a Kripke structure with fairness constraints , denoted $\models_F$, is similar to the semantics of CTL^* defined in section 2.1, except for the following changes:

- $M, s \models_F p \Leftrightarrow$ there exists a fair path starting from s and $p \in L(s)$.
- $M, s \models_F E\psi_1 \Leftrightarrow$ there exists a fair path π starting from s such that $M, \pi \models_F \psi_1$.
- $M, s \models_F A\psi_1 \Leftrightarrow$ for every fair path π starting from s, $M, \pi \models_F \psi_1$.

A quotient model for a Kripke structure with fairness constraints is a five-tuple $M_G = (S_G, S_G^0, R_G, L_G, F_G)$, where S_G, S_G^0, R_G, L_G are defined as in section 2.8, and $F_G = F$.

Definition 3.2 *Let BS be a set of boolean formulas.* A fair bisimulation relation B *for BS over M and M' with F and F' respectively is a bisimulation relation for BS (Definition 2.3) which satisfies, in addition, the following requirement. For every s, s' such that $B(s, s')$ it holds that:*

1. *For every fair path $\pi = s_0 s_1 \ldots$ from $s = s_0$ in M there is a fair path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ in M' such that for all $i \geq 0, B(s_i, s'_i)$.*
2. *For every fair path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ in M' there is a fair path $\pi = s_0 s_1 \ldots$ from $s = s_0$ in M such that for all $i \geq 0, B(s_i, s'_i)$.*

The following lemma is immediate from the result in [8].

[2] Here again, by abuse of notation, f_i refers to the set of states that satisfy the formula f_i.

Lemma 3.2 *Let B be a bisimulation relation for BS and let s, s' be two states such that $B(s, s')$. Then for every path $\pi = s_0 s_1 \ldots$ from $s = s_0$ there is a path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ such that for all $i \geq 0, B(s_i, s'_i)$, and for every path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ there is a path $\pi = s_0 s_1 \ldots$ from $s = s_0$ such that for all $i \geq 0, B(s_i, s'_i)$.*

The following lemma presents a sufficient but not necessary condition which guarantees that B is a fair bisimulation for BS. This condition is useful since it can easily be applied within our symmetry reduction.

Lemma 3.3 *For a given bisimulation relation B for BS over two Kripke structures in which $F = F'$ and two states s and s' such that $B(s, s')$, assume that the following holds: for every $f \in F$*

$$s \models f \Leftrightarrow s' \models f.$$

Then B is a fair bisimulation relation for BS.

Proof: We show that the additional requirement in definition 3.2 is satisfied.

1. For every s, s' such that $B(s, s')$ and for every fair path $\pi = s_0 s_1 \ldots$ from $s = s_0$ in M there is, according to Lemma 3.2, a path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ in M' such that for all $i \geq 0, B(s_i, s'_i)$. According to the definition of a fair path, for every f_i, $inf(\pi) \cap f_i \neq \phi$. Since for every $f \in F\, s \models f \Leftrightarrow\ s' \models f$ we get that for every f_i, $inf(\pi') \cap f_i \neq \phi$. Thus π' is also a fair path.

2. Similar to 1.

■

The following lemma is immediate from the result in [8].

Lemma 3.4 *For every CTL^* formula φ over BS and two Kripke structures M, M' over AP, if M and M' are fair bisimulation equivalent for BS, then $M \models_F \varphi \Leftrightarrow M' \models_{F'} \varphi$.*

Definition 3.3 *Let $\sigma_1, \sigma_2, \ldots, \sigma_k$ be generators of a symmetry group G of a Kripke structure $M = (S, S_0, R, L, F)$ and let MAX be the set of maximal boolean subformulas of a given formula φ. The* fair invariance group *of M w.r.t. $MAX \cup F$ is the group generated by $IG = \{\sigma_i | \forall \beta \in MAX \cup F,\ \sigma_i(\beta) = \beta\}$.*

Lemma 3.5 *Let G_{inv} be the fair invariance group generated by $IG = \{\sigma_i | \forall \beta \in MAX,\ \sigma_i(\beta) = \beta\}$. If IG is not empty, then M and $M_{G_{inv}}$ are fair bisimulation equivalent for BS.*

Proof: According to Lemma 2.4, $B = \{([s], s) | s \in S\}$ is a bisimulation relation for $MAX \cup F$ between $M_{G_{inv}}$ and M. We show that B also satisfies the following additional condition: for every $f \in F$

$$s \models f \Leftrightarrow [s] \models f.$$

According to the definition of the quotient model, $F_{G_{inv}} = F$ and $L_{G_{inv}}([s]) = L(s)$, which implies that $\forall f_i \in F\ [s] \models f_i \Leftrightarrow s \models f_i$. From Lemma 3.3 we get that M and $M_{G_{inv}}$ are fair bisimulation equivalent.

■

The consequence of Lemma 3.5 is that we can use the group generated by IG $= \{\sigma_i | \forall \beta \in MAX \cup F,\ \sigma_i(\beta) = \beta\}$ for symmetry reduction of structures with fairness constraints.

In [13] it is shown that even when the fairness constraints are symmetric to each other the symmetry in the model is not preserved. Thus no compression is possible under fairness when using "purely group-theoretic" methods. Our algorithm does not contradict [13]. Assume that the fairness constraints are symmetric to each other but each single fairness constraint breaks the symmetry in the model. In this case, the fairness constraints will break the symmetry of all generators which do not preserve the symmetry in the model with respect to a single fairness constraint. The symmetry is broken by removing these generators from the fair invariance group. In other words, when using our algorithm in this case, the symmetry in the model is not preserved.

4 Symmetry with On-the-fly Representatives

The algorithm presented in this section is aimed at avoiding the two main problems of symmetry reduction: building the orbit relation and choosing a representative for each orbit.

Let $M = (S, S_0, R, L)$ be a Kripke structure and $\sigma_1, \ldots, \sigma_k$ be a set of generators of a symmetry group G of M. Also let $\varphi = AG(p)$, where p is a boolean formula. The symbolic algorithm **Symmetry_MC**, presented

in Figure 2, applies on-the-fly model checking for M and φ, using under-approximation and symmetry reduction.

The algorithm works in iterations. Starting with the set of initial states, a subset `under` of the current set of states is chosen at each iteration. The successors of `under` are computed. However, every state which is symmetric to (i.e., in the same orbit with) a previously reached state is removed. The states that are first found for each orbit are taken to be the orbit representatives. Note that an orbit may have more than one representative if several of its states are reached when the orbit is encountered for the first time. At any step, the set of representatives is checked to see if it includes a state that violates p. If such a state is found (line 9), then the computation is stopped and a counterexample is produced. We then conclude that $M \not\models AG(p)$. The use of under-approximation ensures that if **Symmetry_MC** terminates without violating p, this does not indicate that $M \models AG(p)$.

Figure 1 shows a possible execution of **Symmetry_MC**. State s is discovered first by the under-approximated search and entered to `reach_rep` as a representative. When $\sigma(s)$ is discovered, it is deleted because $\sigma(s) \in \sigma$**_step**(`reach_rep`).

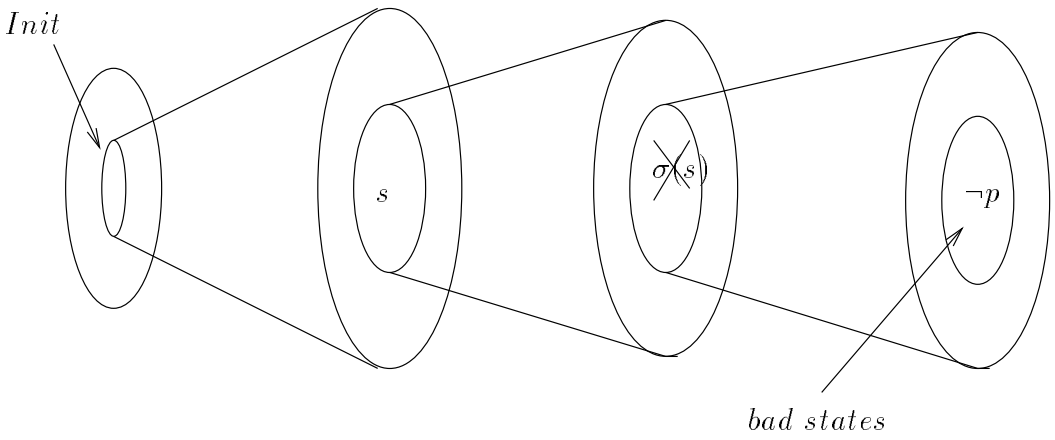

Figure 1: A possible execution of **Symmetry_MC**

A useful optimization can be performed by deleting from memory the BDD for the set `full_reach`, which is often quite large, immediately after it has been used (after line 7). This may prevent memory explosion. In addition, the forward step in line 5 usually requires large memory utilization

as well. By removing the BDD for `full_reach` before computing the forward step, we decrease the memory usage in each iteration.

Symmetry_MC(M, φ, $\sigma_1, \ldots \sigma_k$)

1 Calculate the generators of the invariance group of M
IG = $\{\sigma_i |$ for all maximal boolean subformulas β of φ: $\sigma_i(\beta) = \beta\}$
2 `reach_rep` = S_0, i=0
3 while $S_i \neq \emptyset$
4 choose `under` $\subseteq S_i$ (`under` is an under-approximation of S_i)
5 $S_{i+1} = Im_R$(`under`)
6 `full_reach` = σ**_Step**(`reach_rep`, $\sigma_1, \ldots \sigma_k$)
7 $S_{i+1} = S_{i+1}$ / `full_reach`
8 `reach_rep` = `reach_rep` $\cup S_{i+1}$
9 if $S_{i+1} \wedge \neg p \neq \emptyset$
10 generate a counterexample and break.
11 i = i+1.

Figure 2: The algorithm **Symmetry_MC** performs on-the-fly model checking of φ on M, using symmetry reduction.

The set of symmetric states that should be removed are computed using the procedure σ**_Step** (Figure 3) instead of using the orbit relation. σ**_Step** applies Im_{σ_i}(`sym_states`)[3] in order to obtain the states which are related by σ_i to states in `sym_states`. It repeatedly computes Im_{σ_i} for $i = 1, \ldots, k$, until a fixed point is reached. For a set of states A and a set of generators $IG = \{ \sigma_1, \ldots, \sigma_k \}$, σ**_Step** returns the set of all states belonging to the orbits of states in A according to $G = \langle IG \rangle$. The use of σ**_Step** for removing symmetric states is demonstrated in Figure 4.

We use σ**_Step** to exploit symmetry information without building the full orbit relation. There are several reasons to expect that σ**_Step** will result in a BDD which is smaller than that of the orbit relation. First, it represents a set of states and not a relation. Thus, it depends on one set of BDD variables, while the orbit relation depends on two sets. Second, it is applied only to

[3] σ_i can be viewed as the binary relation $\sigma(\bar{v}, \bar{v}')$.

```
   σ_Step(A, σ_1, σ_2, ..., σ_k)
1      sym_states = A;
2      old_sym_states = ∅
3      while old_sym_states ≠ sym_states
4           old_sym_states = sym_states
5           for i = 1...k
6              new_sym_states = Im_σi(sym_states)
7              sym_states = sym_states ∪ new_sym_states
8      return sym_states
```

Figure 3: The algorithm σ**_Step** calculates the states belonging to the orbits of states in A.

reachable states, which are usually represented by smaller BDDs. Indeed, our experiments successfully applied σ**_Step** to designs for which building the orbit relation was impossible.

Computationally, σ**_Step** is quite heavy. We avoided this problem in most of our experiments by stopping the computation of σ**_Step** before it reached a fixed point. As will be explained in the next section, this does not effect the correctness of **Symmetry_MC**. In general, there is a tradeoff between the amount of computation in σ**_Step** and the symmetry reduction obtained by **Symmetry_MC**.

If, when **Symmetry_MC** terminates, `reach_rep` contains at least one state from each reachable orbit, then M_m, defined according to `reach_rep` ($S_m =$ `reach_rep`), is bisimilar to M (see Section 2.8). Thus, if $M_m \models AG(p)$, then $M \models AG(p)$ as well. Note that $M_m \models AG(p)$ can be checked on-the-fly.

4.1 Robustness of Symmetry_MC

We now discuss the robustness of the algorithm **Symmetry_MC** for falsification in the presence of an incomplete σ**_Step** and an incorrect set of generators. Consider first the case in which the computation of procedure σ**_Step** is stopped before a fixed point is reached. σ**_Step** then returns only a subset of the states in the orbits of states in A. In this case, fewer states are

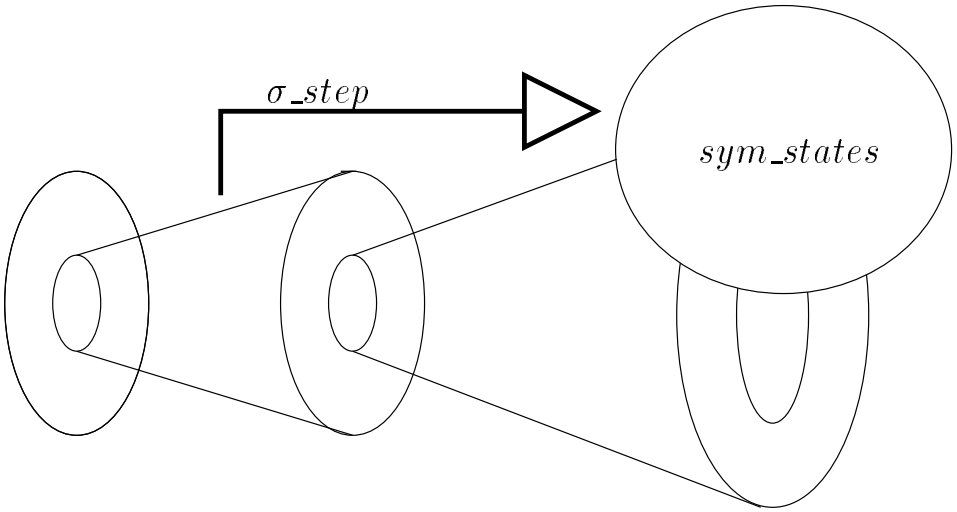

Figure 4: Removing symmetric states using σ**_Step**

removed from S_{i+1} and, as a result, `reach_rep` contains more states. Thus, we might have more representatives for each orbit.

Consider now the case in which the algorithm is given an incorrect set of generators. If a "bad" generator (a permutation which associates states that are not symmetric) is given, then σ**_Step** returns states which are not symmetric to any state in `reach_rep`. These states are removed from S_{i+1}, and it is possible that no representatives of their orbits will be added to `reach_rep`. Thus, `reach_rep` represents an under-approximation of the reachable orbits. However, `reach_rep` does not contain a representative of an unreachable orbit. Thus, if there is a state $s \in$ `reach_rep` which does not satisfy p, this state is reachable in the original model, and the counterexample generated by **Symmetry_MC** actually exists in the original model.

If a "good" generator (a permutation which associates pairs of symmetric states) is missing, then σ**_Step** returns fewer states and, as a result, there is more than one representative for each orbit. However, as in the previous case, `reach_rep` contains only reachable states, and therefore **Symmetry_MC** generates only real counterexamples. The following lemma summarizes the discussion above.

Lemma 4.1 *Given any set of generators, the algorithm* **Symmetry_MC** *is sound for falsification.*

Proof: We need to prove that if **symmetry_MC** generates a counterexample (line 10), then there is a reachable state s in M which does not satisfy p. Since **symmetry_MC** enters line 10 only if $S_{i+1} \wedge \neg p \neq \phi$, we only have to show that for every i and for every $s \in S_i$, s is reachable along transitions in M. The proof is by induction:

- i=0, all states in S_0 are reachable.
- Assume all states in S_i are reachable. $S_{i+1} = Im_R(\texttt{under})$ / σ**_step** (`reach_rep`) where `under` $\subseteq S_i$. Since all states in S_i are reachable, all states in `under` are reachable. By the definition of Im_R, $Im_R(\texttt{under})$ is also a set of reachable states. In addition, $S_{i+1} \subseteq Im_R(\texttt{under})$, and thus all states in S_{i+1} are reachable.

■

Several BDD optimizations common in the computation of forward steps in symbolic model checking may be useful in the implementation of procedure σ**_Step**:

- To simplify each σ_i according to `sym_states` before computing Im_{σ_i} (`sym_states`). Simplifying the BDD representing σ_i will result in a smaller BDD. The first possibility is that this BDD might contain pairs of states which are not symmetric to each other and in which the first state is not in `sym_states`. The second possibility is that this BDD might not contain pairs of states which are symmetric to each other and in which the first state is not in `sym_states`. In both cases the result of $Im_{\sigma_i}(\texttt{sym\_states})$ is the same as without simplification.
- To apply the partitioned transition relation with early quantification [11] in the computation of Im_{σ_i}. This is applicable because a permutation is often given as a conjunction of simpler expressions $((\sigma_i(\bar{v}, \bar{v}') = \wedge_j(\bar{v}_j = \bar{v}'_{\sigma(j)}))$.

5 Symmetry Reduction Combined with Hints

In this section we present a special case of the algorithm **Symmetry_MC** in which the under-approximation is guided by a sequence of hints given by the user [3].

Let M = (S,S_0,R,L) be a Kripke structure, $\sigma_1, \ldots, \sigma_k$ be a set of generators of a symmetry group G on M, and $h_1, \ldots, h_l$ be a sequence of hints where for all i, $h_i \subseteq S$ and $h_l = S$. Also, let $\varphi = AG(p)$ be a formula where p is a boolean formula. The algorithm **Hints_Sym**, presented in Figure 5, applies on-the-fly model checking for M and φ using hints and symmetry reduction. In each iteration we choose `under` to be the conjunction of a hint and the result of the previous image computation. First we calculate `under` according to h_1. When a fixed point is reached for hint h_i (i.e., there is no state which is both in the result of the previous image computation and in h_i), we switch to calculating `under` according to h_{i+1}. Since $h_l = S$, the image computations in the last fixed point computation are no longer restricted. As a result, when the last fixed point is reached `reach_rep` contains at least one representative from each orbit.

If $\sigma_1, \ldots, \sigma_k$ contain no "bad" generator, then our hints guarantee that when $S_i = \emptyset$, `reach_rep` contains at least one state from each orbit. Since $h_l = S$ when $hint = h_l$, `under` is equal to S_i in each iteration. Thus, when $S_i = \emptyset$, all reachable orbits are searched. In this case, the algorithm **Hints_Sym** is suitable for verification as well as for falsification. In many cases, the nonexistence of bad generators can be easily determined by the program syntax, for example when using the syntax presented in [22] or the one presented in [10]. In other cases it is expensive but possible to check whether all generators are good. (This can be done by checking whether $s \equiv_{bis} \sigma(s)$ for every state s and every generator σ.)

A useful optimization is to compute the set `full_reach` according to `reach_rep` only once for each hint (on line 12 instead of line 6 in algorithm **Hints_Sym**), and to use it in order to remove states in all steps in which this hint is used. This optimization saves computation time but may use more space since `full_reach` is in memory when Im_R is computed. Again there is a tradeoff here between time and space.

We chose to combine hints with **Symmetry_MC** since hints may be very useful in on-the-fly model checking of models with symmetry. For example, if for each process there is a signal en_i, which is active only when processor i is active, running the algorithm with the following hints will search the state space gradually.

- $h_1 = \{s | s \models \forall_{2 \leq i \leq n}(en_i = false)\}$.

Hints_Sym(M, φ, $\sigma_1, \ldots, \sigma_k$, $h_1, h_2, \ldots, h_l$)

1 Calculate the generators of the invariance group of M
IG = $\{\sigma_i |$ for all maximal boolean subformula β of φ: $\sigma_i(\beta) = \beta\}$
2 `reach_rep` = S_0, i = 0, `hint` = h_1, j = 2
3 while $S_i \neq \emptyset$
4 `under` = $S_i \cap$ `hint`
5 $S_{i+1} = Im_R$(`under`)
6 `full_reach` = σ**_Step**(`reach_rep`, $\sigma_1, \ldots \sigma_k$)
7 $S_{i+1} = S_{i+1}$/`full_reach`
8 `reach_rep` = `reach_rep` $\cup S_{i+1}$
9 if $S_{i+1} \wedge \neg p \neq \emptyset$
10 generate counterexample and break
11 if $S_{i+1} = \emptyset \wedge j \leq l$
12 `hint` = h_j
13 j = j+1
14 S_{i+1} = `reach_rep`
15 i = i+1
16 φ is TRUE

Figure 5: The algorithm **Hints_Sym** applies on-the-fly model checking of φ on M, using hints and symmetry reduction.

- $h_2 = \{s | s \models \forall_{3 \leq i \leq n}(en_i = false)\}$.
- ...
- $h_{n-1} = \{s | s \models (en_n = false)\}$.
- $h_n = S$.

When combining hints with symmetry reduction in algorithm **Hints_Sym**, the first reachable states of the k processes will be discovered when using $hint_k$ and will be kept as a representative in `reach_rep`. All other reachable states of k processors will be discovered by σ**_step** (line 6) and removed from the model (line 7) in the following iterations. For example, for k=2 all

combinations of processes 1 and 2 will be discovered when using $hint_2$ and entered into `reach_rep`. When using $hint_3$, all combinations of processes 1 and 3, and all combinations of processes 2 and 3, will be discovered and immediately removed since they are symmetric to states in `reach_rep`

If a bug occurs when 3 processes are active, it will be discovered while using $hint_3$ when only processes 1,2 and 3 are active. Since other combinations of three processes will be discovered only later, these hints enable us to find the bug before the BDD representing the reachable state space becomes too big.

6 Extension for Temporal Safety Properties

In this section we extend algorithm **Symmetry_MC** for temporal safety properties. There are several known algorithms which use a construction A_φ for the evaluated formula φ and the product model $M \times A_\varphi$ in order to model check more complex properties. We now show that it is possible to combine symmetry reduction with these algorithms.

Lemma 6.1 *Let $M \times A_\varphi$ be the product model of M and A_φ. Then for every invariance group G_{inv} of M w.r.t. the maximal boolean subformulas of φ, and for every $\sigma \in G_{inv}$, $(s,t) \in S_{M\times A_\varphi} \Leftrightarrow (\sigma(s),t) \in S_{M\times A_\varphi}$*

Proof:

According to the definition of an invariance group, for every invariance group G_{inv} and every $\sigma \in G_{inv}$, $L(s) = L(\sigma(s))$. According to the definition of a product model, (s, t) is a state in the product model if and only if $L(s) \cap AP_{A_\varphi} = L_{A_\varphi}(t) \cap AP\}$. It follows that for every invariance group G_{inv} of M w.r.t. the maximal boolean subformulas of φ, and for every $\sigma \in G_{inv}$,$(s,t) \in S_{M\times A_\varphi} \Leftrightarrow (\sigma(s),t) \in S_{M\times A_\varphi}$.

■

In this section a permutation is a permutation on each variable. The permutation is applied on each variable value $\sigma(s) = \sigma_1(a_1), \sigma_2(a_2), \ldots, \sigma_k(a_k)$ where $s = (a_1, a_2, \ldots, a_k)$. If the permutation is applied to a bigger set of variables, then the permutation on the additional variables is the identity permutation.

We first specify the requirements necessary for using construction A_φ with symmetry reduction.

Definition 6.1 *Given a logic $\mathcal{L}$ and a construction that associates with each $\varphi \in \mathcal{L}$ a structure A_φ, the construction A_φ is safe for symmetry reduction w.r.t $\mathcal{L}$ if there exists $\psi \in CTL^*$ such that the following conditions are satisfied:*

1. $\forall \varphi \in \mathcal{L}\ (M \models \varphi \Leftrightarrow M \times A_\varphi \models \psi)$.
2. *For every invariance group G_{inv} of M w.r.t. the maximal boolean subformulas of φ, every $\sigma \in G_{inv}$, and every $(s,t) \in S_{M \times A_\varphi}$, we have that $\sigma((s,t)) = (\sigma(s),t)$* [4].
3. *For every $\beta \in AP_\psi$ and every $(s,t),(s',t) \in S_{M \times A_\varphi}$,*

$$(s,t) \models \beta \Leftrightarrow (s',t) \models \beta.$$

The second condition requires that σ be defined only on s and that t be left unchanged. The third condition requires that the truth of all β in ψ depends only on t.

Lemma 6.2 *For every construction A_φ which is safe for symmetry reduction w.r.t $\mathcal{L}$, if G is an invariance group of structure M w.r.t. the maximal boolean subformulas of formula $\varphi \in \mathcal{L}$, then G is an invariance group of structure $M \times A_\varphi$ w.r.t the maximal boolean subformulas of ψ.*

Proof:

First we show that for every $\beta \in AP_\psi$, every $\sigma \in G$, and every $(s,t) \in M \times A_\varphi$, $(M \times A_\varphi,(s,t) \models \beta \Leftrightarrow M \times A_\varphi, \sigma(s,t) \models \beta)$.
By condition 3 of safe construction, $\forall \beta \in AP_\psi\ \forall \sigma \in G\ \forall (s,t) \in M \times A_\varphi$ $(s,t) \models \beta \Leftrightarrow (\sigma(s),t) \models \beta$. By condition 2 we have that $(s,t) \models \beta \Leftrightarrow \sigma(s,t) \models \beta$, as required ($(\sigma(s),t) \in S_{M \times A_\varphi}$, by lemma 6.1).

Next we show that $\forall \sigma \in G\ \forall (s,t)(s',t') \in S_{M \times A_\varphi}\ ((s,t),(s',t')) \in R_{M \times A_\varphi} \Leftrightarrow (\sigma(s,t),\sigma(s',t')) \in R_{M \times A_\varphi})$.

For every $\sigma \in G$ and for every $(s,t),(s',t') \in S_{M \times A_\varphi}$
$((s,t),(s',t')) \in R_{M \times A_\varphi} \Rightarrow$ (by the definition of product structure)
$(s,s') \in R \wedge (t,t') \in R_{A_\varphi} \Rightarrow$ (G is an invariance group of M)

[4] Note that $(\sigma(s),t) \in S_{M \times A_\varphi}$, according to lemma 6.1.

$(\sigma(s),\sigma(s')) \in R \wedge (t,t') \in R_{A_\varphi} \Rightarrow ((\sigma(s),t)$ and $(\sigma(s'),t'))$ are in $S_{M\times A_\varphi}$ by Lemma 6.1 and by the definition of product structure)
$((\sigma(s),t),(\sigma(s'),t')) \in R_{M\times A_\varphi} \Rightarrow$ (condition 2 of safe construction)
$(\sigma(s,t),\sigma(s',t')) \in R_{M\times A_\varphi}$.

For every $\sigma \in G$ and for every $(s,t),(s',t')$ such that $\sigma(s,t),\sigma(s',t') \in S_{M\times A_\varphi}$
$(\sigma(s,t),\sigma(s',t')) \in R_{M\times A_\varphi} \Rightarrow$ (condition 2 of safe construction)
$((\sigma(s),t),(\sigma(s'),t')) \in R_{M\times A_\varphi} \Rightarrow$ (by the definition of product structure)
$(\sigma(s),\sigma(s')) \in R \wedge (t,t') \in R_{A_\varphi} \Rightarrow (G$ is an invariance group of M)
$(s,s') \in R \wedge (t,t') \in R_{A_\varphi} \Rightarrow ((s,t)$ and (s',t') are in $S_{M\times A_\varphi}$ by Lemma 6.1 and by the definition of product structure)
$((s,t),(s',t')) \in R_{M\times A_\varphi}$.

■

Corollary 6.3 *For every construction A_φ which is safe for symmetry reduction w.r.t $\mathcal{L}$, for every $\varphi \in \mathcal{L}$ and the $\psi \in CTL^*$ which exists according to condition 1 of definition 6.1, the quotient structure $(M \times A_\varphi)_G$, built for $M \times A_\varphi$ and an invariance group G of M, satisfies $(M \times A_\varphi)_G \models \psi \Leftrightarrow M \models \varphi$.*

Proof: By condition 1 of safe construction there is ψ such that: $M \models \varphi \Leftrightarrow M \times A_\varphi \models \psi$. Since $M \times A_\varphi \equiv_{bis} (M \times A_\varphi)_G$ (by lemma 6.2), $M \times A_\varphi \models \psi \Leftrightarrow (M \times A_\varphi)_G \models \psi$. It follows that $(M \times A_\varphi)_G \models \psi \Leftrightarrow M \models \varphi$. ■

Note that using the safe construction enables us to find a set of generators of the invariance group of M according to the maximal boolean subformulas of φ and then to evaluate formula ψ on $M \times A_\varphi$ with a symmetry reduction that uses the same generators.

6.1 Safe Construction for LTL Model Checking

There are several A_φ constructions which are safe for symmetry reduction w.r.t logic $\mathcal{L}$. One example is the tableau construction in [7], when restricted to LTL safety properties.

LTL is a subset of CTL^*, consisting only of formulas of the form Af where f is a path formula and all its state sub-formulas are atomic propositions. LTL safety formulas are LTL formulas, restricted to the temporal operators X, V and G.

The tableau associated with an LTL formula Af is a structure $T_f = (S_f, S_f^0, R_f, L_f)$ over the set of atomic propositions of f, AP_f. $S_f = 2^{el(f)}$ where $el(f)$ is defined as follows:

- $el(p) = \{p\}$ if $p \in AP_f$.
- $el(\neg g) = el(g)$.
- $el(g \vee h) = el(g) \cup el(h)$.
- $el(Xg) = \{Xg\} \cup el(g)$.
- $el(gVh) = \{X(gVh)\} \cup el(g) \cup el(h)$.
- $el(Gg) = \{XGg\} \cup el(g)$.

L_f labels each state by the set of atomic propositions contained in the state. $R_f(s, s') = \bigwedge_{Xg \in el(f)}(s \in sat(Xg) \Leftrightarrow s' \in sat(g))$, where $sat(f)$ is defined as follows:

- $sat(g) = \{s | g \in s\}$ where $g \in el(f)$.
- $sat(\neg g) = \{s | s \notin sat(g)\}$.
- $sat(g \vee h) = sat(g) \cup sat(h)$.
- $sat(gVh) = sat(h) \cap (sat(g) \cup sat(X(gVh)))$.
- $sat(Gg) = sat(g) \cap sat(XGg)$.

The initial states of T_f, $S_f^0 = sat(f)$.

Note that since only LTL safety properties are considered, the tableau construction does not include fairness constraints.

Lemma 6.4 *[7] for all $Af \in LTL$, $M, s \not\models Af$ if and only if there is a state t in $T_{\neg f}$ such that $t \in sat(\neg f)$ and $(M \times T_{\neg f}), (s, t) \models EG(true)$.*

Lemma 6.5 *The tableau construction T_f (without fairness constraints) fulfills the requirements of Definition 6.1.*

Our proof is based on the construction in which the states of the tableau $T_{\neg f}$ and the states of M are defined over disjoint sets of state variables, and the states of the product model $M \times T_{\neg f}$ are pairs of states (s,t) where s is a state of M and t is a state in $T_{\neg f}$. However, other constructions, which differ from this one only in that the states of the product model $M \times T_{\neg f}$ are defined differently, are safe for symmetry reduction as well because they are equivalent to the first construction.

Proof:

We will show that all three requirements are fulfilled for $\mathcal{L}$ = LTL safety properties, $\varphi = Af$, and $A_\varphi = T_{\neg f}$.

1. According to lemma 6.4, for all $\varphi \in \mathcal{L}$, $(M \models \varphi \Leftrightarrow \forall s \in S_0$ there is no state t in $T_{\neg f}$ such that $(s,t) \in sat(\neg f)$ and $M \times T_{\neg f}, (s,t) \models EG(true))$. The set of initial states in $T_{\neg f}$ is sat$(\neg f)$. Thus by the definition of a product model we get that there is no state $t \in sat(\neg f)$ such that $M \times T_{\neg f}, (s,t) \models EG(true)$ if and only if $M \times T_{\neg f} \models \neg EG(true))$. Therefore $\exists \psi = \neg EG(true)$ such that $\forall \varphi \in \mathcal{L}$ $(M \models \varphi \Leftrightarrow M \times T_f \models \psi)$.

2. Since the state of T_f is defined over a set of state variables which is disjoint from the set of state variables of M, for every invariance group G_{inv} of M w.r.t. the maximal boolean subformulas of φ, no $\sigma \in G_{inv}$ changes the values of the variables which define the state of T_f. Therefore, for every $(s,t) \in S_{M \times T_f}, \sigma((s,t)) = (\sigma(s), t)$.

3. The only maximal boolean subformula of $\neg EG(true)$ is $\beta = true$. Every $(s,t), (s',t) \in S_{M \times T_{\neg f}}$ satisfies $((s,t) \models true \Leftrightarrow (s',t) \models true)$.

■

6.2 Safe Construction for RCTL Model Checking

Another safe construction is the satellite for RCTL formulas defined in [2]. When specifying a property of a model we usually describe what specification should hold in the model. Another way to specify a property is to describe what should never hold in the model. A regular expression can be used to describe the set of "bad" computations. For a regular expression RE, each

computation whose prefix RE describes is a "bad" computation. We denote a regular expression which describes "bad" computations by $\{RE\}(false)$.

Let A'_{RE} be a finite automaton which is built from the regular expression RE with an additional self-loop for each accepting state. Let A_{RE} be the Kripke structure that is generated from A'_{RE} by the standard construction which translates an automaton to a Kripke structure. Each state in A_{RE} which is generated from an accepting state in A'_{RE} is marked with a new atomic property, $match_{RE}$. Finally, M is checked against RE by evaluating $AG(\neg match_{RE})$ over the product model $M \times A_{RE}$.

$RCTL$ is the subset of CTL^* which can be translated to a regular expression. The automaton A_{RE} that is built for a formula $\varphi \in RCTL$ is called the satellite of φ. The full definition of $RCTL$ and an algorithm for translating a CTL formula in this subset to a regular expression specification can be found in [2].

Lemma 6.6 *The satellite construction T_f for RCTL formulas [2] fulfills the requirements of Definition 6.1.*

Proof: We will show that all three requirements are fulfilled for $\mathcal{L} =$ RCTL and for every $\varphi = \{RE\}(false)$, for which $A_\varphi = A_{RE}$.

1. In [2] it was proved that for all $\varphi \in RCTL$, $M \models \varphi \Leftrightarrow M \times A_\varphi \models AG(\neg match_{RE})$. Thus, $\exists \psi = AG(\neg match_{RE})$ such that $\forall \varphi \in \mathcal{L}$ $(M \models \varphi \Leftrightarrow M \times A_\varphi \models \psi)$.

2. According to [2], the state of A_φ is defined over a set of state variables which is disjoint from the set of state variables of M. Thus, for every invariance group G_{inv} of M w.r.t the maximal boolean subformulas of φ, no $\sigma \in G_{inv}$ changes the values of the variables which define the states of A_φ. Therefore, for every $(s,t) \in S_{M \times A_\varphi}, \sigma((s,t)) = (\sigma(s),t)$.

3. The only maximal boolean subformula of $AG(\neg match_R)$ is $\beta = \neg match_R$. The truth value of $match_R$ depends only on the variables which define the states of A_φ. Since these variables are disjoint from the variables which code M, every $(s,t),(s',t) \in S_{M \times A_\varphi}$ satisfies $((s,t) \models \beta \Leftrightarrow (s',t) \models \beta)$.

■

By combining a safe construction with symmetry reduction, we can apply symmetry reduction to a new set of algorithms. These include symbolic on-the fly model checking and symbolic LTL model checking algorithms, for which it was not applicable until now. We implemented our algorithms using the construction introduced in [2], which enabled us to check RCTL formulas on-the-fly while using symmetry reduction.

7 Extensions for Liveness Formulas

We now describe two possible extensions that combine classical (not on-the-fly) symbolic model checking with symmetry reduction. These extensions are useful for checking liveness properties, as well as other properties which cannot be checked on-the-fly.

7.1 Liveness Restricted to Representatives

The purpose of this extension is to falsify $ACTL^*$ formulas with respect to a structure M, while avoiding the construction of its quotient model M_G. The idea is to get a set of representatives, Rep, and to construct the restricted model $M|_{Rep}$ (see Definition 2.6). Since $M|_{Rep} \leq_{sim} M$, we have that for every $ACTL^*$ formula φ, if $M|_{Rep} \not\models \varphi$ then $M \not\models \varphi$. Thus, φ can be checked on the smaller model $M|_{Rep}$.

Note that in principle this idea works correctly with any set of representatives, even one that does not include a representative for each orbit. There are, however, advantages to choosing as Rep the set `reach_rep` obtained from the algorithm **Symmetry_MC**. First, since `reach_rep` includes only reachable states, its BDD is usually smaller than the BDD of an arbitrarily chosen set of states. Second, by construction, the states in `reach_rep` are connected by transitions, while an arbitrary set of representatives *Rep* might not be connected. Thus, $M|_{reach\_rep}$ often includes more behaviors than $M|_{Rep}$. As a result, it is more likely that a bad behavior, if one exists, will be found in $M|_{reach\_rep}$. Third, the states in `reach_rep` represent many other states in the system. Thus, if the system includes a bad behavior, it is more likely that `reach_rep` will reflect it.

In order to evaluate liveness properties, we propose the Algorithm **Live_Rep**. This algorithm

- Runs **Symmetry_MC** to obtain `reach_rep`.
- Performs classical symbolic model checking on $M|_{reach\_rep}$.

Unfortunately, $M|_{reach\_rep} \not\equiv_{bis} M$ even when there is a representative for each orbit. Consider Figure 6, which shows two symmetric cycles, C_1 and C_2, in M. The states in `reach_rep` are marked in black. One orbit is $\{s_1, t_1\}$, and s_1 is its representative. Another orbit is $\{s_2, t_2\}$, and t_2 is its representative. There are two edges, r_1 and r_2, between these two orbits. Since neither r_1 nor r_2 are in $M|_{reach\_rep}$, $M|_{reach\_rep} \not\equiv_{bis} M$. This is true even though there is a representative for each orbit in this example.

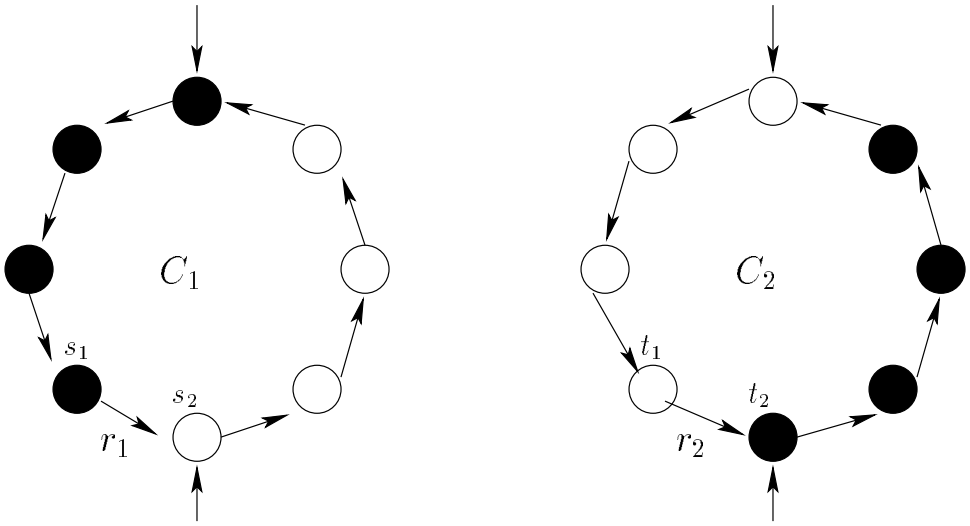

Figure 6: $M|_{reach\_rep} \not\equiv_{bis} M$.

7.2 Liveness with the Representative Relation

We now present another possibility for handling liveness properties. It is applicable only if no bad generators exist. This method is more expensive computationally, but is suitable for verification of liveness properties. As in

the previous section, we first compute `reach_rep` using the algorithm **Symmetry_MC**. Then we apply the procedure **Create_ξ**, presented below, to compute the representative relation $\xi \subseteq$ `reach_rep` $\times S$ (see definition 2.15). Next we construct M_m according to ξ. Finally, we run classical symbolic model checking on φ and M_m.

Lemma 7.1 *If S_m contains at least one representative for each orbit, then $M \equiv_{bis} M_m$. Otherwise, $M_m \leq_{sim} M$.*

Proof: Define the relation B = $\{ (s', s) \mid \exists \sigma \in G : \sigma(s') = s \wedge s' \in S_m \}$. If S_m contains at least one representative for each orbit, then B is a bisimulation relation for the set of maximal boolean subformulas of φ. Otherwise, B is a simulation relation for the set of maximal boolean subformulas of φ. ∎

If `reach_rep` is the result of the algorithm **Hints_Sym** and all generators are good, then `reach_rep` indeed contains at least one representative for each orbit, and M_m is bisimilar to M. Thus, M_m can be used for verifying full CTL^*.

7.2.1 Computing The Representative Relation ξ

Figure 7 presents the BDD-based procedure **Create_ξ**. This procedure builds the representative relation ξ for a given set of representatives, Rep, and a set of generators, $\sigma_1, \ldots, \sigma_k$, of an invariance group G of M w.r.t the maximal boolean subformulas of φ.

Suppose that each σ_i is represented by a BDD, $\sigma_i(\bar{v}, \bar{v}')$, and Rep is represented by a BDD, Rep($\bar{v}$). **Create_ξ** returns the BDD $\xi(\bar{v}, \bar{v}')$ for the representative relation $\xi \subseteq Rep \times S$.

Note that the computation on line 6 of the algorithm depends on three sets of BDD variables: $\bar{v}, \bar{v}', \bar{v}''$. Such a computation usually results in large intermediate BDDs that may cause a memory explosion. To avoid this, we implement line 6 using the operator **compose_odd**(A($\bar{v}, \bar{v}'$), B($\bar{v}'', \bar{v}'$)), which computes the BDD $\exists \bar{v}'($ A$(\bar{v}, \bar{v}') \wedge$ B$(\bar{v}'', \bar{v}'))$ using two sets of BDD variables rather than three. Thus, it is expected to result in smaller BDDs. The operator compose_odd has been defined in [19].

Next we show that for every set of generators $\{\sigma_1, \ldots \sigma_k\}$ and for every set of representatives *Rep*, **Create_ξ** calculates the representative relation ξ for the invariance group generated by $\{\sigma_1, \ldots \sigma_k\}$. According to Definition 2.15, $(s, s') \in \xi \Leftrightarrow s \in Rep \wedge [s] = [s']$. In **Create_$\xi$**, s, s' are represented by

Create_$\xi(\sigma_1, \ldots \sigma_k, \text{Rep})$

1 $\quad \xi(\bar{v}, \bar{v}') = \text{Rep}(\bar{v}) \wedge (\bar{v} = \bar{v}')$
2 $\quad$ `old_`$\xi(\bar{v}, \bar{v}') = \phi$
3 $\quad$ while `old_`$\xi(\bar{v}, \bar{v}') \neq \xi(\bar{v}, \bar{v}')$
4 $\quad\quad$ `old_`$\xi(\bar{v}, \bar{v}') = \xi(\bar{v}, \bar{v}')$
5 $\quad\quad$ for j = 1 ... k
6 $\quad\quad\quad$ `new`$(\bar{v}, \bar{v}') = \exists \bar{v}''(\xi(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$
7 $\quad\quad\quad$ $\xi(\bar{v}, \bar{v}') = \xi(\bar{v}, \bar{v}') \cup$ `new`$(\bar{v}, \bar{v}')$
8 $\quad$ return $\xi(\bar{v}, \bar{v}')$

Figure 7: The algorithm **Create_**ξ for computing $\xi \subseteq Rep \times S$

$\bar{v}, \bar{v}'$. Thus, we have to prove that $\xi(\bar{v}, \bar{v}')$ created by **Create_**ξ satisfies $\xi(\bar{v}, \bar{v}') \Leftrightarrow Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$.

Let $\xi_i(\bar{v}, \bar{v}')$ be $\xi(\bar{v}, \bar{v}')$ after i iterations of the "for" statement in algorithm **Create_**ξ.

Lemma 7.2 $\forall i \; [\xi_i(\bar{v}, \bar{v}') \Rightarrow (Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}'])]$.

Proof: Proof by induction on i:

- $\xi_0(\bar{v}, \bar{v}') = Rep(\bar{v}) \wedge \bar{v} = \bar{v}'$ (line 1).
  It follows that $\xi_0(\bar{v}, \bar{v}') \Rightarrow Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$.

- According to lines 6 and 7, $\xi_{i+1}(\bar{v}, \bar{v}') = \xi_i(\bar{v}, \bar{v}') \vee \; \exists \sigma_j \in \{\sigma_1, \ldots \sigma_k\}$ $\exists \bar{v}'' \; (\xi_i(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$. For all $\bar{v}, \bar{v}'$ which satisfy $\xi_{i+1}(\bar{v}, \bar{v}')$, one of the following cases is true:

  - $\bar{v}, \bar{v}'$ satisfy also $\xi_i(\bar{v}, \bar{v}')$. In this case $\bar{v}, \bar{v}'$ satisfy $Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$ since $\xi_i(\bar{v}, \bar{v}') \Rightarrow Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$.
  - $\bar{v}, \bar{v}'$ satisfy $\exists \sigma_j \in \{\sigma_1, \ldots \sigma_k\} \exists \bar{v}''(\xi_i(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$. In this case $\bar{v}$ satisfies $Rep(\bar{v})$ since $\xi_i(\bar{v}, \bar{v}'') \Rightarrow Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}'']$ and $\bar{v}, \bar{v}'$ satisfy $[\bar{v}] = [\bar{v}']$ since $\exists \bar{v}''(\xi_i(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$.

  In both cases $\bar{v}, \bar{v}'$ satisfy $Rep(\bar{v}) \wedge [\bar{v}] = \bar{[v']}$, which implies that $\xi_{i+1}(\bar{v}, \bar{v}')$ $\Rightarrow Rep(\bar{v}) \wedge [\bar{v}] = \bar{[v']}$.

■

Lemma 7.3 *($Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}'] \Rightarrow \xi(\bar{v}, \bar{v}')$, where $\xi(\bar{v}, \bar{v}')$ is calculated by* **Create_ξ**.

Proof: Assume there exist $\bar{v}, \bar{v}'$ which satisfy $Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$ but do not satisfy $\xi(\bar{v}, \bar{v}')$. Since $\bar{v}, \bar{v}'$ satisfy $Rep(\bar{v}) \wedge [\bar{v}] = [\bar{v}']$, there exists $\sigma_{j_1}, \sigma_{j_2}, ..., \sigma_{j_n} \in \{\sigma_1, \ldots \sigma_k\}$ such that $\sigma_{j_1}\sigma_{j_2}...\sigma_{j_n}(\bar{v}) = \bar{v}'$.
Let $\bar{v}''$ be $\sigma_{j_h}\sigma_{j_{h+1}}...\sigma_{j_n}(\bar{v})$ for the highest h which does not satisfy $\xi(\bar{v}, \bar{v}'')$ and let $\bar{v}^*$ be $\sigma_{j_{h+1}}...\sigma_{j_n}(\bar{v})$ for the same h.
According to these definitions, $\xi(\bar{v}, \bar{v}^*) \wedge \neg\xi(\bar{v}, \bar{v}'')$. Assume i is the first iteration of **Create_ξ** in which $\xi_i(\bar{v}, \bar{v}^*)$ is satisfied. Since $\sigma_{j_h} \in \{\sigma_1, \ldots \sigma_k\}$, $\bar{v}''$ satisfies $\exists\sigma_{j_h} \in \{\sigma_1, \ldots \sigma_k\}\ \exists\bar{v}^*(\xi_i(\bar{v}, \bar{v}^*) \wedge \sigma_{j_h}(\bar{v}^*, \bar{v}''))$ in iteration i. According to lines 6 and 7, $\bar{v}, \bar{v}''$ satisfy $\xi_{i+1}(\bar{v}, \bar{v}'')$, which implies that they also satisfy $\xi(\bar{v}, \bar{v}'')$, in contradiction to the definition of $\bar{v}''$. ■

Corollary 7.4 *The relation ξ computed by* **Create_$\xi(\sigma_1, \ldots \sigma_k, Rep)$** *is the representative relation for Rep and the invariance group generated by $\{\sigma_1, \ldots \sigma_k\}$.*

7.2.2 Robustness for Falsification

The relation ξ constructed by **Create_ξ** will be used in order to define a reduced model, as defined in 2.16. If the set Rep contains at least one representative from each orbit, then **Create_ξ** returns exactly the relation ξ used in the definition of R_m. Let this ξ be denoted by ξ_m.

Below we show that we can also get meaningful results in other cases. First assume that Rep does not include at least one representative from each orbit. In this case, ξ does not contain pairs of states from the unrepresented orbits, and $\xi \subseteq \xi_m$.

Now assume that time or space limitations have prevented the computation of ξ from being completed (meaning that the computation of **Create_ξ** is stopped before $old\_\xi = \xi$). In this case ξ might not associate each representative with all the states in its orbit. As a result, $\xi \subseteq \xi_m$.

For any set Rep and for any partially computed ξ, we can define the Kripke Structure $M_{Rep} = (S_{Rep}, S^0_{Rep}, R_{Rep}, L_{Rep})$ in which $S_{Rep} = Rep$, $S^0_{Rep} = \exists s'\xi(s, s') \wedge S_0(s')$, $R_{Rep} = \xi^{-1}R\xi$, $L_{Rep} = \mathrm{L}$. If $\xi = \xi_m$ then $R_{Rep} = R_m$, and there is a bisimulation relation between M and M_{Rep} for the set

of maximal boolean subformulas of φ (since $M_{Rep} \equiv_{bis} M_m \equiv_{bis} M$). This implies that M and M_{Rep} agree on every CTL^* formula. If $\xi \subseteq \xi_m$, then $R_{Rep} \subseteq R_m$, and there is a simulation relation between M_{Rep} and M for the set of maximal boolean subformulas of φ (since $M_{Rep} \leq_{sim} M_m \equiv_{bis} M$). Thus, for every $ACTL^*$ formula φ, if $M_{Rep} \not\models \varphi$ then $M \not\models \varphi$.

We conclude that the algorithm **Create_ξ** is robust for falsification in the presence of incomplete Rep or ξ.

8 Iterative Symmetry Reduction

In this section we discuss another way to exploit symmetry for falsification of safety properties. This time the method is based on a variant of the procedure **Create_ξ**, presented in the previous section. In section 4 we showed how to use **σ_Step** to avoid building the orbit relation. Another way to avoid building the full orbit relation is to build it iteratively. In each iteration, only a subset of the orbit relation, which is represented by a relatively small BDD, is built. This approach makes it possible to improve the algorithm presented in [18]. To this purpose we introduce the function **Create_ξ_limit** (Figure 8), which is identical to **Create_ξ** except for one change. **Create_ξ_limit** stops calculating the ξ relation when the BDD size is larger than `limit`. We use **Create_ξ_limit** iteratively on different sets of representatives in order to build different subsets of the orbit relation. We use these subsets of the orbit relation to search different subsets of the transitions of the quotient model in each iteration.

Lemma 8.1 *Given two sets of states A, B, if $A \subseteq B$ then* **Create_ξ**$(\sigma_1, \ldots \sigma_k, A) \subseteq$ **Create_ξ**$(\sigma_1, \ldots \sigma_k, B)$.

Proof: Let $\xi_l(\bar{v}, \bar{v}')$ be $\xi(\bar{v}, \bar{v}')$ after l executions of the "for" statement in algorithm **Create_ξ**. The proof is by induction on l:

- $\xi_{A_0}(\bar{v}, \bar{v}') = (A(\bar{v}) \wedge \bar{v} = \bar{v}')$ and $\xi_{B_0}(\bar{v}, \bar{v}') = (B(\bar{v}) \wedge \bar{v} = \bar{v}')$ (line 1). Since $A \subseteq B$, it follows that $\xi_{A_0}(\bar{v}, \bar{v}') \subseteq \xi_{B_0}(\bar{v}, \bar{v}')$.

- $\xi_{A_{i+1}}(\bar{v}, \bar{v}') = \xi_{A_i}(\bar{v}, \bar{v}') \vee \exists \sigma_j \in \{\sigma_1, \ldots \sigma_k\} \exists \bar{v}''(\xi_{A_i}(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$ and $\xi_{B_{i+1}}(\bar{v}, \bar{v}') = \xi_{B_i}(\bar{v}, \bar{v}') \vee \exists \sigma_j \in \{\sigma_1, \ldots \sigma_k\} \exists \bar{v}''(\xi_{B_i}(\bar{v}, \bar{v}'') \wedge \sigma_j(\bar{v}'', \bar{v}'))$ (lines 6,7). By the induction hypothesis, $\xi_{A_i}(\bar{v}, \bar{v}') \subseteq \xi_{B_i}(\bar{v}, \bar{v}')$. It follows that $\xi_{A_{i+1}}(\bar{v}, \bar{v}') \subseteq \xi_{B_{i+1}}(\bar{v}, \bar{v}')$.

Create_ξ_limit($\sigma_1, \ldots \sigma_k$, Rep, `limit`)

1 $\xi(\bar{v}, \bar{v}') = \mathrm{Rep}(\bar{v}) \wedge (\bar{v} = \bar{v}')$
2 $\texttt{old\_}\xi(\bar{v}, \bar{v}') = \phi$
3 while $\texttt{old\_}\xi(\bar{v}, \bar{v}') \neq \xi(\bar{v}, \bar{v}')$
and BDD_size($\xi(\bar{v}, \bar{v}')$) $\leq$ `limit`
4 $\texttt{old\_}\xi(\bar{v}, \bar{v}') = \xi(\bar{v}, \bar{v}')$
5 for j = 1 ... k
6 $\texttt{new}(\bar{v}, \bar{v}'') = \exists \bar{v}'(\xi(\bar{v}, \bar{v}') \wedge \sigma_j(\bar{v}'', \bar{v}'))$
7 $\xi(\bar{v}, \bar{v}') = \xi(\bar{v}, \bar{v}') \cup \texttt{new}(\bar{v}, \bar{v}')$
8 return $\xi(\bar{v}, \bar{v}')$

Figure 8: The algorithm **Create_ξ_limit** for computing $\xi \subseteq Rep \times S$ with limit on the BDD size

■

Let $M = (S, S_0, R, L)$ be a Kripke structure and $\{\sigma_1, \ldots, \sigma_k\}$ be a set of generators of a symmetry group G of M. In addition, let φ be a formula of the form $AG(p)$ where p is a boolean formula. The algorithm **Iterative_Sym**, presented in Figure 9, applies on-the-fly model checking for M and φ, using iterative symmetry reduction.

Iterative_Sym uses **Create_ξ_limit** to build, in each iteration, the relation ξ_i. If the size of the BDD which represents the representative relation ξ for Rep is larger than `limit`, **Create_ξ_limit** returns $\xi_i \subseteq \xi$. After calculating ξ_i, **Iterative_Sym** calculates a new model M'. M' is different from the quotient model M_m in two respects. First, it is built using ξ_i and not the full representative relation ξ. Since the size of the BDD which represents ξ_i is smaller in most cases than ξ, there is a high probability that the BDDs which represent M' are smaller than the BDDs which represent M_m. Second, $R' = \xi^{-1} R \cap (Rep \times Rep)$. Note that R' is restricted to states in Rep and not in Rep_i. If it were restricted to states in Rep_i, **Create_ξ_limit** might miss reachable representatives. To see how this might happen, consider three states, s_1, s_2 and s_3, as shown in Figure 10. s_1 and s_2 are related by R, and s_3 and s_2 are in the same orbit for which s_3 is the representative. Assume that $(s_3, s_2) \notin \xi_i$ because **Create_ξ_limit** did not find (s_3, s_2) in iteration i. As

Iterative_Sym(M, φ, $\sigma_1, \ldots \sigma_k$, Rep, limit)

1 Calculate the generators of the invariance group of M
IG = $\{\sigma_i |$ for all maximal boolean subformulas β of φ: $\sigma_i(\beta) = \beta\}$
2 `prev_reachable_states` = ϕ
3 `reachable_states` = S_0
4 `i` = 0
5 Rep_0 = Rep
6 while ($Rep_i \neq \phi \wedge$ `prev_reachable_states` $\neq$ reachable_states)
7 `old_reachable_states` = `reachable_states`
8 ξ_i = **Create_ξ_limit** ($\sigma_1, \ldots \sigma_k, Rep_i, limit$)
9 build M' according to ξ_i
10 (S' = Rep, $S'_0 = \xi_i^{-1} S_0$, $R' = \xi_i^{-1} R \cap (Rep \times Rep)$ and $L' = L$)
11 `reachable_states` = the reachable states of
$M' \cup$ `reachable_states`
12 if a "bad" state is found while calculating
the reachable states of M'
13 generate counterexample and break
14 `i` = `i` + 1
15 $Rep_i = Rep$ / `reachable_states`.
16 S_0 = `reachable_states`.

Figure 9: The algorithm **Iterative_Sym** evaluates φ using iterative symmetry reduction

a result, $(s_1, s_3) \notin R'$ and s_3 is not found to be a reachable representative in this iteration. In iteration $i+1$, s_1 is no longer in Rep_{i+1} since it was already in `reachable_states` (line 15) even though $(s_3, s_2) \in \xi_{i+1}$. If we restrict R' to Rep_{i+1}, we get that (s_1, s_3) is not in R'. Thus, s_3 is also not found to be reachable in iteration $i+1$.

If we restrict R' to Rep, then, since $s_3 \in Rep$, $(s_1, s_3) \in R'$ and s_3 is entered into `reachable_states` in iteration $i+1$. Once M' is constructed, a "bad" state found in M' will cause a counterexample to be generated. If a "bad" state is not found, the set `reachable_states` is a subset of the union of the reachable states of M_m and the initial states of M. We mark them as the initial states of the next iteration and build ξ_i again only for the

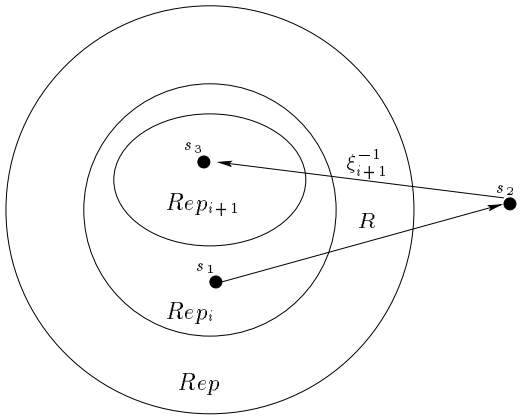

Figure 10: Two iterations of **Iterative_Sym**

representatives which are not known to be reachable states in M_m. In the next iteration we build ξ_{i+1} for a smaller set of representatives. Therefore we may build a larger portion of ξ for these representatives before we reach the limit in **Create_ξ_limit**. As a result, we might discover new states when computing the set of reachable states for M' in iteration $i+1$.

We now prove the correctness of the algorithm **Iterative_Sym** for falsification of safety properties. First we show that the transition relation R', which is built in each iteration of **Iterative_Sym**, is a subset of the transition relation R_m of the quotient model M_m, defined for *rep* (see definition 2.16).

Lemma 8.2 *Given a representative relation* ξ, $\xi^{-1}R \cap (Rep \times Rep) \subseteq \xi^{-1}R\xi$.

Proof: $(s,s') \in \xi^{-1}R \cap (Rep \times Rep) \Rightarrow$
$(s,s') \in \xi^{-1}R \wedge s \in Rep \Rightarrow$
$(s,s') \in \xi^{-1}R \wedge (s,s) \in \xi \Rightarrow (s,s') \in \xi^{-1}R\xi.$ ∎

Let ξ_m be the representative relation for the set of representatives *Rep* given as parameter to **Iterative_Sym** (definition 2.15).

Lemma 8.3 *For every iteration* i *of algorithm* **Iterative_Sym**, $\xi_i \subseteq \xi_m$.

Proof:

- According to line 15, $Rep_i \subseteq Rep$ for all i. By Lemma 8.1 we get that for all i,
  Create_ξ $(\sigma_1, \ldots \sigma_k,\ Rep_i) \subseteq$ **Create_ξ**$(\sigma_1, \ldots \sigma_k,\ Rep)$.

- **Create_ξ_limit**$(\sigma_1, \ldots \sigma_k,\ Rep_i,$ limit) is identical to **Create_ξ** except that it may be stopped after fewer iterations. In addition, once a pair of states is entered to $\xi(\bar{v}, \bar{v}')$, it is not removed in either algorithm. It follows that **Create_ξ_limit**$(\sigma_1, \ldots \sigma_k,\ Rep_i,$ limit$) \subseteq$ **Create_ξ**$(\sigma_1, \ldots \sigma_k,\ Rep_i)$.

- By corollary 7.4, **Create_ξ**$(\sigma_1, \ldots \sigma_k,\ Rep)$ produces the representative relation ξ_m for the set of representatives Rep.

It follows that for all iterations i of **Iterative_Sym**, $\xi_i =$ **Create_ξ_limit** $(\sigma_1, \ldots \sigma_k,\ Rep_i,$ limit$) \subseteq \xi_m$. ■

Lemma 8.4 *For every iteration i of algorithm* **Iterative_Sym**, $R' \subseteq R_m$.

Proof: According to Lemma 8.3, for every iteration i of algorithm **Iterative_Sym**, $\xi_i \subseteq \xi_m$. This implies that $R' = \xi_i^{-1} R \cap (Rep \times Rep) \subseteq \xi_m^{-1} R \cap (Rep \times Rep) \subseteq_{Lemma\ 8.2} \xi_m^{-1} R \xi_m = R_m$. ■

Next we show that if **Iterative_Sym** finds a "bad" state, this state is a "bad" reachable state of structure M_m.

Lemma 8.5 *For every iteration i of* **Iterative_Sym**, *every $s \in$* `reachable_states` *is either a reachable representative in M_m or an initial state of M.*

Proof: Let `reachable_states`$_i$ be the set `reachable_states` after iteration i. The proof is by induction on i:

- For every $s \in$ `reachable_states`$_0$, s is in S_0, and thus it is an initial state of M.

- For every $s \in$ `reachable_states`$_{i+1}$, one of the following cases is true:

  - s is in `reachable_states`$_i$. In this case, according to the induction hypothesis, s is either reachable in M_m or an initial state of M.

– There exists a state $s' \in S'_0$ from which s is reachable in M'. According to Lemma 8.3, $\xi_{i+1} \subseteq \xi_m$. It follows that all states in $\xi_{i+1}^{-1}(\texttt{reachable\_states}_i)$ are representatives of orbits in which there are states from $\texttt{reachable\_states}_i$. According to the induction hypothesis and because $S'_0 = \xi_{i+1}^{-1}(\texttt{reach able\_states}_i)$, s' is either a representative of an initial state of M or a representative of a reachable orbit in M_m. In both cases s' is reachable is M_m. By Lemma 8.4, $R' \subseteq R_m$. Thus, s is either reachable in M_m or an initial state of M as well.

■

The conclusion from the previous lemma is that if a "bad" state is found by **Iterative_Sym**, M does not satisfy φ.

9 Experimental Results

We implemented the algorithms **Hints_Sym**, **Live_Rep**, **Iterative_Sym**, **Create_ξ** and **Create_ξ_limit** in the IBM model checker RuleBase [1]. We ran it on a number of examples which contain symmetry. For each example we tuned our algorithms according to the evaluated formula, the difficulty level of computing the reachable states, and the difficulty level of building the transition relation. In most cases, our algorithms outperformed RuleBase with respect to both time and space. In the tables below, time is measured in seconds, memory (mem) in bytes, and the transition relation size (TR size) in the number of BDD nodes.

The Futurebus example: We ran the algorithm **Live_Rep** in order to check liveness properties on the Futurebus cache-coherence protocol with a single bus and a single cache line for each processor. We checked the property "along every path infinitely often some processor is in exclusive write." This property fails because our model does not include fairness constraints. The table in Figure 11 presents the results of evaluating the property for different numbers of processors. For comparison, we also ran the RuleBase classical symbolic model checking algorithm. Both algorithms applied dynamic BDD reordering. Dynamic BDD reordering is very important because the best BDD order for the classical algorithm is different from the best BDD order for our algorithm. In order to obtain a fair comparison between these algorithms,

we ran each one twice. In the first run, the algorithm reordered the BDD with no time limit in order to find a good BDD order. The initial order of the second run was the BDD order which was found by the first run.

The most difficult step in the Futurebus example is building the transition relation. When the transition relation was restricted to the representatives which were chosen on-the-fly, its size decreased and, as a result, the evaluation became easier. In this case we chose to complete the calculation of σ**_Step** in order to obtain the maximal reduction in the size of the transition relation. Figure 11 shows that both time and space were reduced dramatically using **Live_Rep**. We can also observe that the larger the number the processors was, the better the results were. This is to be expected, as the increase in the number of the reachable representatives is smaller than the increase in the number of reachable states.

#	#	classic algorithm			Live_Rep		
of processors	vars	time	mem	TR size	time	mem	TR size
5	45	132	43M	144069	101	41M	122769
6	54	607	118M	260625	265	56M	219572
7	63	2852	277M	418701	704	76M	379428
8	72	8470	589M	839055	3313	101M	457781
9	81	81,171	709M	1935394	4571	106M	819871
10	90	-	$>$ 1G	-	4909	120M	642083

Figure 11: **Hints_Sym** on Futurebus example

The Arbiter example: We ran algorithm **Hints_Sym** on an arbiter example with n processes. We checked the arbiter with regard to RCTL formulas which were translated to safe A_φ and ψ. For comparison, we ran RuleBase on-the-fly model checking and on-the-fly model checking with hints (without symmetry). All algorithms used dynamic BDD reordering and a partitioned transition relation [11]. In this case we calculated σ**_Step** only when we changed hints and stopped σ**_Step** before the fixed point was reached. The table in Figure 12 presents the results of the three algorithms on arbiter with 6, 8 and 10 processes. For each case we checked one property that passed and one that failed. We note that **Hints_Sym** reduced time but not necessarily space. This can be explained by the fact that σ**_Step** produced large intermediate BDDs but resulted in a significant reduction in S_i, thus reducing the computation time of the image steps.

# of processors	status	# vars	on-the-fly		on-the-fly + hints		Hints_Sym	
			time	mem	time	mem	time	mem
6	passed	65	53	40M	39	40M	42	40M
6	failed	65	213	52M	64	41M	51	87M
8	passed	84	581	64M	255	49M	179	87M
8	failed	84	745	71M	524	71M	292	83M
10	passed	105	1470	94M	598	67M	358	92M
10	failed	105	1106	93M	740	73M	520	91M

Figure 12: **Hints_Sym** compared to other on-the-fly algorithms

Comparing Create_ξ and Orbit_To_ξ: [18] presents an algorithm for computing ξ by building the orbit relation and then choosing the representatives. We refer to this algorithm as **Orbit_To_ξ** and compare it to **Create_ξ**. Both algorithms find the representative relation $\xi \subseteq Rep \times S$ for the set of representatives Rep chosen according to the lexicographic order. The results in Figure 13 show that **Create_ξ** gave better results in both time and space. We believe that this is because it saves less information while building ξ.

num of generators	num of vars	orbit_to_ξ		Create_ξ	
		time	mem	time	mem
3	16	0.26	26M	0.23	26M
4	20	30.4	33M	1.2	28M
5	24	1017	114M	18	42M
6	28	-	>1.5G	735	132M
7	32	-	>1.5G	29083	1.2G

Figure 13: **Create_ξ** compared to **Orbit_To_ξ**

Combinatorial Covering Suites: We ran **Iterative_Sym** on the Combinatorial Covering Suites problem. This problem is taken from the world of testing and coverage. Let $D_1, D_2, \ldots, D_k$ be the domains of k input variables. Each *test vector* is an assignment to all input variables, and a *test suite* with N tests is an array of N test vectors. A is a *2-wise covering suite* if for every two distinct input variables v_i, v_j and every $T \in D_i \times D_j$ there exists a test vector in which the assignment to v_i and v_j is equal to T. The *covering suite number* is the minimum integer N such that there exists a 2-wise covering suite with N test vectors. Combinatorial covering suites is based on the in-

tuition that when testing a program where each test is an assignment to all inputs, it is preferable to use the smallest test suite that still contains all combinations of inputs. The *covering suite number* is the minimal size of a test suite with such a property. We model the problem in RuleBase. We wrote a specification on the model which must fail if there exists a covering suite of size k. The counterexample to the specification was the k vectors of the covering suite. We observed that when the input variables are over the same domain, there is a symmetry between them. Changing the order of the input variables in the covering suite results in another covering suite. We ran **Iterative_Sym** and RuleBase on-the-fly model checking on a special case of the problem where all inputs are boolean. We compared these algorithms on different numbers of inputs. **Iterative_Sym** reduced both time and space. We gave **Iterative_Sym** the set of all the smallest representatives, ordered lexicographically, as the parameter *Rep*. We believe **Iterative_Sym** was successful in this example because sets of lexicographically ordered smallest representatives were compressed. However, our experience shows that in most cases the sets of lexicographic representatives are not compressed in their BDD size. In these cases we expect that **Hints_Sym** will give better results. The table in Figure 14 presents the results of **Iterative_Sym** on different numbers of inputs.

num of inputs	num of vars	Iterative_Sym		on-the-fly		on-the-fly + hints	
		time	mem	time	mem	time	mem
5	45	5.6	39M	318	86M	17.7	67M
6	66	32	41M	-	>1G	159605	1624M
7	91	36	41M	-	>1G	-	>1G
8	120	103	42M	-	>1G	-	>1G
9	153	175	50M	-	>1G	-	>1G
10	190	852	92M	-	>1G	-	>1G
11	231	4429	112M	-	>1G	-	>1G
12	276	26240	177M	-	>1G	-	>1G
13	325	66103	303M	-	>1G	-	>1G

Figure 14: **Iterative_Sym** compared to RuleBase on-the-fly and on-the-fly+hints

10 Conclusions and Directions for Future Research

The main contribution of this work is the introduction of an on-the-fly model checking algorithm that combines under-approximation with symmetry reduction, chooses the representatives according to BDD criteria, and avoids building the orbit relation. In addition, we introduce another iterative on-the-fly algorithm that builds subsets of the orbit relation rather than the full relation.

We show how to extend our algorithms both for temporal safety properties, based on a definition of safe for symmetry reduction constructions, and for liveness properties. We also show how to build the invariance group automatically from a given symmetry group.

Our work can be extended in several ways. First, rather than using hints to define the under-approximation (as is done in **Hints_Sym**), we can apply **Symmetry_MC** with other criteria for choosing the next set of states to explore. A useful possibility may be high-density, presented in [21].

In addition, our work can be combined with algorithms that use subsets of the reachable states. Often, if the algorithm is applied to some subset of states A, it is unnecessary to apply it to states which are symmetric to states in A. In these cases we may use σ**_Step** in order to ignore states which are symmetric to states in A without building the orbit relation. For example, σ**_Step** can be used in Prioritized Traversal [15]. There, the BDDs in the priority queue can be reduced by eliminating states which are in the orbits of states that were already explored.

Acknowledgments: We thank Cindy Eisner for many helpful discussions. We also thank Somesh Jha for his help with the examples.

References

[1] I. Beer, S. Ben-David, C. Eisner, and A. Landver. RuleBase: An industry-oriented formal verification tool. In *Design Automation Conference*, pages 655–660, June 1996.

[2] I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer-Aided Verification*, volume 1427 of *LNCS*, pages 184–194. Springer-Verlag, June 1998.

[3] R. Bloem, K. Ravi, and F. Somenzi. Symbolic guided search for CTL model checking. In *Design Automation Conference*, pages 29–34, June 2000.

[4] M. Browne, E. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Comput. Sci.*, 59:115–131, 1988.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE transactions on Computers*, C-35(8):677–691, 1986.

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

[7] E. Clarke, O. Grumberg, and H. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, 10(1), 1997.

[8] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

[9] 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, 1986.

[10] C.N. Ip and D.L. Dill. Better verification through symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Computer Hardware Description Languages and their Applications*, pages 87–100, Ottawa, Canada, 1993. Elsevier Science Publishers B.V., Amsterdam, Netherlands.

[11] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation. In David L. Dill, editor, *Proceedings of the Sixth International Conference on Computer-Aided Verification*, volume 818, pages 299–310. Springer-Verlag, June 1994.

[12] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer-Aided Verification*, volume 697 of *LNCS*. Springer-Verlag, June 1993.

[13] E. A. Emerson and A. P. Sistla. Utilizing symmetry when model-checking under fairness assumptions: An automata-theoretic approach. *ACM Transactions on Programming Languages and Systems*, 19(4):617–638, July 1997.

[14] E. A. Emerson and R. J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Conference on Correct Hardware Design and Verification Methods*, pages 142–156, 1999.

[15] R. Fraer, G. Kamhi, B. Ziv, M. Y. Vardi, and L. Fix. Prioritized traversal: Efficient reachability analysis for verification and falsification. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification*, volume 1855 of *LNCS*, pages 389–402. Springer-Verlag, July 2000.

[16] O. Grumberg and D. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[17] V. Gyuris and A. P. Sistla. On-the-fly model checking under fairness that exploits symmetry. *Formal Methods in System Design: An International Journal*, 15(3):217–238, November 1999.

[18] S. Jha. *Symmetry and Induction in Model Checking*. PhD thesis, CMU, 1996.

[19] S. Katz. Coverage of model checking. Master's thesis, Technion, Haifa, Israel, 2001.

[20] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

[21] K. Ravi and F. Somenzi. High-density reachability analysis. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 154–158, November 1995.

[22] A. P. Sistla, V. Gyuris, and E. A. Emerson. SMC: a symmetry-based model checker for verification of safety and liveness properties. *Software Engineering and Methodology*, 9(2):133–166, 2000.

[23] C. H. Yang and D. L. Dill. Validation with guided search of the state space. In *Design Automation Conference*, pages 599–604, June 1998.

A $ACTL^*$ Preservation Over a Kripke Structure with Finite Paths

Lemma 2.2 states that for every $ACTL^*$ formula φ over a set of boolean formulas BS over AP and two Kripke structures M, M' over AP, if $M \leq_{sim} M'$ then $M' \models \varphi \Rightarrow M \models \varphi$. This lemma is proved in [16] for every two Kripke structures with infinite paths. In this appendix we show that Lemma 2.2 is true for every two Kripke structures with finite paths as well.

First we define the semantics of $ACTL^*$ over a Kripke structure with finite paths.

An infinite path π in a Kripke structure M is an infinite sequence of states $s_0, s_1, s_2, \ldots$ in which $\forall i \geq 0$ $(s_i, s_{i+1}) \in R$. A finite path π in a Kripke structure M is a finite sequence of states $s_0, s_1, s_2, \ldots s_n$ in which $\forall 0 \leq i < n$ $(s_i, s_{i+1}) \in R$ and there is no state s' in M such that $(s_n, s') \in R$. As before, π^i denotes the suffix of π starting at s_i.

The size of a path π, $|\pi|$, is defined as follows:

- if π is an infinite path then $|\pi| = inf$, where $inf > n$ for all $n > 0$.
- if π is a finite path, $\pi = s_0, s_1, s_2, ..., s_n$ then $|\pi| = n$.

For a state formula φ, we write $M, s \models_{fin} \varphi$ to indicate that φ is true in state s and for a path formula ψ, we write $M, \pi \models_{fin} \psi$ to indicate that ψ is true along π where π might be finite.

The relation $\models_{fin}$ is defined as follows, assuming that φ_1 and φ_2 are state formulas and ψ_1 and ψ_2 are path formulas.

- $M, s \models_{fin} p \Leftrightarrow p \in L(s)$.
- $M, s \models_{fin} \neg p \Leftrightarrow M, s \not\models_{fin} p$.

- $M, s \models_{fin} \varphi_1 \vee \varphi_2 \Leftrightarrow M, s \models_{fin} \varphi_1$ or $M, s \models_{fin} \varphi_2$.
- $M, s \models_{fin} \varphi_1 \wedge \varphi_2 \Leftrightarrow M, s \models_{fin} \varphi_1$ and $M, s \models_{fin} \varphi_2$.
- $M, s \models_{fin} A\psi_1 \Leftrightarrow$ for every path $\pi = s_0, s_1, \ldots$ in M such that $s_0 = s$, $M, \pi \models_{fin} \psi_1$.
- $M, \pi \models_{fin} \varphi_1 \Leftrightarrow s$ is the first state of π and $M, s \models_{fin} \varphi_1$.
- $M, \pi \models_{fin} \psi_1 \vee \psi_2 \Leftrightarrow M, \pi \models_{fin} \psi_1$ or $M, \pi \models_{fin} \psi_2$.
- $M, \pi \models_{fin} \psi_1 \wedge \psi_2 \Leftrightarrow M, \pi \models_{fin} \psi_1$ and $M, \pi \models_{fin} \psi_2$.
- $M, \pi \models_{fin} X\psi_1 \Leftrightarrow |\pi| = 0$ or $M, \pi^1 \models_{fin} \psi_1$.
- $M, \pi \models_{fin} \psi_1 U \psi_2 \Leftrightarrow |\pi| = n \wedge \exists k\ [0 \leq k \leq n \wedge k \neq inf \wedge M, \pi^k \models_{fin} \psi_2$ and $\forall 0 \leq i < k:\ M, \pi^i \models_{fin} \psi_1]$ or $[n \neq inf \wedge \forall 0 \leq i \leq n\ M, \pi^i \models_{fin} \psi_1]$.
- $M, \pi \models_{fin} \psi_1 V \psi_2 \Leftrightarrow \forall 0 \leq k \leq n$ [if $\forall 0 \leq i < k\ M, \pi^i \not\models_{fin} \psi_1$ then $M, \pi^k \models_{fin} \psi_2$].

There exist different definitions of $\models_{fin}$ in the literature. These definitions guarantee that liveness properties actually hold before the path is ended. In this case the simulation relation should be defined differently. We use the regular simulation relation but change the definition of $\models_{fin}$.

Lemma A.1 *If $M, s \leq_{sim} M', s'$ then for every path $\pi = s_0, s_1, \ldots$ from $s = s_0$ in M there exists a path $\pi' = s'_0, s'_1, \ldots$ in M' such that $|\pi| \leq |\pi'|$ and for every $i \leq |\pi|$, $M, s_i \leq_{sim} M', s'_i$.*

Proof: Given any path $\pi = s_0, s_1, \ldots$ from $s = s_0$ in M we show how to build $\pi' = s'_0, s'_1, \ldots$ in M' such that $|\pi| \leq |\pi'|$ and for every $i \leq |\pi|$, $M, s_i \leq_{sim} M', s'_i$.

- $s'_0 = s'$.
- For every $0 \leq i < |\pi|$, assume that $s'_1, s'_2, \ldots, s'_i$ has already been defined such that $M, s_i \leq_{sim} M', s'_i$. Since $M, s_i \leq_{sim} M', s'_i$ and $(s_i, s_{i+1}) \in R$, there exists t such that $(s'_i, t) \in R$ and $M, s_{i+1} \leq_{sim} M', t$. We choose $s'_{i+1} = t$.

■

Lemma A.2 *If* $M, s \leq_{sim} M', s'$ *then for every*

$$\varphi \in ACTL^* \; M', s' \models_{fin} \varphi \Rightarrow M, s \models_{fin} \varphi.$$

Proof: The proof is by induction on φ. It is identical to the proof in [16] except for the following changes:

- $\varphi = A\varphi_1$.
  $M, s \models_{fin} \varphi$ if and only if for every path from s, $M, \pi \models_{fin} \varphi_1$. Let π be any path from s. According to Lemma A.1 there is a path π' from s' such that $|\pi| \leq |\pi'|$ and for every $i \leq |\pi|$, $M, s_i \leq_{sim} M', s'_i$. If $M', s' \models_{fin} \varphi$, then $M', \pi' \models_{fin} \varphi_1$ for every path π' from s'. By the induction hypothesis we get that for all π from s, $M, \pi \models_{fin} \varphi_1$. Thus $M, s \models_{fin} \varphi$.

- For every path formula ψ we show that for every π in M and π' in M' such that $|\pi| \leq |\pi'|$ and for all $i < |\pi|$, $M, s_i \leq_{sim} M', s'_i$,

  $$M', \pi' \models_{fin} \psi \Rightarrow M, \pi \models_{fin} \psi$$

  - $\psi = X\psi_1$.
    $M', \pi' \models_{fin} \psi$ implies that $|\pi'| = 0$ or $M', \pi'^1 \models_{fin} \psi_1$. If $|\pi| = 0$ then $M, \pi \models_{fin} \psi$. Otherwise, since $|\pi| \leq |\pi'|$, $|\pi^1| \leq |\pi'^1|$ and $|\pi'| > 0$. In addition, for all $i < |\pi^1|$, $M, s_{i+1} \leq_{sim} M', s'_{i+1}$. By the induction hypothesis we get that $M, \pi^1 \models_{fin} \psi_1$, which implies that $M, \pi \models_{fin} \psi$.
  - $\psi = \psi_1 U \psi_2$
    $M', \pi' \models_{fin} \psi$ where $|\pi'| = n$ implies that $\exists k[0 \leq k \leq n \wedge k \neq inf \wedge M', \pi'^k \models_{fin} \psi_2$ and $\forall 0 \leq i < k : \; M', \pi'^i \models_{fin} \psi_1]$ or $[n \neq inf \wedge \forall 0 \leq i \leq n \; M', \pi'^i \models_{fin} \psi_1]$. Assume there is $i \leq |\pi|$ such that $M, \pi^i \not\models_{fin} \psi_1$ and $\forall j \leq i$, $M, \pi^j \not\models_{fin} \psi_2$. Then by the induction hypothesis and the fact that $|\pi| \leq |\pi'|$, we get that there is $i \leq |\pi'|$ such that $M', \pi'^i \not\models_{fin} \psi_1$ and $\forall j \leq i \; M', \pi'^j \not\models \psi_2$, in contradiction to the assumption that $M', \pi' \models_{fin} \psi$.

■