Symmetry and Reduced Symmetry in Model Checking *

A. Prasad Sistla[1] Patrice Godefroid[2]

[1] University of Illinois at Chicago, Department of Electrical Engineering and Computer Science, Chicago, IL 60607, USA
[2] Bell Laboratories, Lucent Technologies, Naperville, IL 50566, USA

Abstract. Symmetry reduction methods exploit symmetry in a system in order to efficiently verify its temporal properties. Two problems may prevent the use of symmetry reduction in practice: (1) the property to be checked may distinguish symmetric states and hence not be preserved by the symmetry, and (2) the system may exhibit little or no symmetry. In this paper, we present a general framework that addresses both of these problems. We introduce "Guarded Annotated Quotient Structures" for compactly representing the state space of systems even when those are asymmetric. We then present algorithms for checking any temporal property on such representations, including non-symmetric properties.

1 Introduction

In the last few years there has been much interest in symmetry-based reduction methods for model checking concurrent systems [10, 2, 4, 5, 8, 12]. These methods exploit automorphisms, of the global state graph of the system to be verified, induced by permutations on process indices and variables. Existing symmetry-reduction methods, for verification of a correctness property given by a temporal formula ϕ, can be broadly classified into two categories: the first class of methods [2, 4, 10, 12] consider only those automorphisms that preserve the atomic predicates appearing in ϕ, construct a *Quotient Structure* (QS) and check the formula ϕ on the QS using traditional model-checking algorithms; the second class of methods [5] consider all automorphisms, induced by process/variable permutations, and construct an *Annotated Quotient Structure* (AQS), and unwind it to verify the formula ϕ.

In this paper, we generalize symmetry-based reduction in several ways. First, the mathematical framework, used to formalize symmetry reduction, supports *any* automorphism on the system's state graph; for example, automorphisms induced by permutations on variable-value pairs can be considered in addition to those induced by permutations on process indices and variables. Thus, this framework allows for more automorphisms and hence greater reduction.

* Sistla's work is supported in part by the NSF grant CCR-9988884 and was partly done while visiting Bell Laboratories.

Second, we introduce the notion of *Guarded Annotated Quotient Structure* (GQS) to represent, in a very compact way, the state graph of systems with little or even no symmetry. In a nutshell, a GQS is an AQS whose edges are also associated with a guard representing the condition under which the corresponding original program transition is executable. Given a program P and its reachability graph G, by adding edges to G (via a transformation of P), we obtain another graph H that has more symmetry than G, and hence can be represented more compactly. A GQS for G can be viewed as an AQS for H whose edges are labeled with guards in such a way that the original edges of G can be recovered from the representation of H. To verify a temporal formula ϕ, the GQS is unwound as needed, by tracking the values of the atomic predicates in ϕ and the guards of the GQS, so that only edges in G are considered. The GQS of G can be much smaller than its QS because it is defined from a larger set of automorphisms: a GQS is derived by considering all the automorphisms of H, which exhibits more symmetry than G, including those automorphisms that do not preserve the atomic predicates in ϕ. We show that unwinding GQS on-demand, in order to verify a property ϕ, can be done without ever generating a structure larger than QS.

Third, we present two new techniques for further optimizing the model-checking procedure using GQSs. These techniques minimize the amount of unwinding necessary to check a formula ϕ and may yield an exponential improvement in performance. The first technique, called *formula decomposition*, consists of decomposing ϕ into groups of top-level sub-formulas so that atomic predicates with in a group are correlated; the satisfaction of ϕ can then be checked by checking each group of sub-formulas separately, which in turn can be done by successively unwinding the GQS with respect to only the predicates appearing in each group separately; therefore, unwinding GQS with respect to all the atomic predicates appearing in ϕ simultaneously can be avoided. The second technique, called *sub-formula tracking*, consists of identifying a maximal set of "independent" sub-formulas of ϕ and unwinding the GQS by tracking these sub-formulas only. These two complementary techniques can be applied recursively.

The paper is organized as follows. Section 2 introduces the background information and notation. Section 3 introduces GQS and the model-checking method employing it. Section 4 presents the techniques based on formula decomposition and sub-formula tracking. Section 5 presents preliminary experimental results. Section 6 contains concluding remarks and related work. Proofs of theorems are omitted due to space limitations.

2 Background

A Kripke structure K is a tuple $(S, E, \mathcal{P}, L)$ where S is a set of elements, called states, $E \subseteq S \times S$ is a set of edges, $\mathcal{P}$ is a set of atomic propositions and $L : S \rightarrow 2^{\mathcal{P}}$ is a function that associates a subset of $\mathcal{P}$ with each state in S. CTL^* is a logic for specifying temporal properties of concurrent programs (e.g., see [3]). It includes the temporal operators U (until), X (nexttime) and the existential path quantifier E. Two types of CTL^* formulas are defined inductively: path formulas and state formulas. Every atomic proposition is a state formula as well as a path formula. If p and q are state formulas (resp., path formulas) then $p \wedge q$ and $\neg p$ are also state formulas (resp., path formulas). If p and q are path formulas then $p\mathsf{U}q$, $\mathsf{X}p$ are path formulas and $\mathsf{E}(p)$ is a state formula. Every state formula is also a path formula. We use the abbreviation $\mathsf{EF}(p)$ for $\mathsf{E}(True\mathsf{U}p)$ and $\mathsf{AG}(p)$ for $\neg(\mathsf{EF}\neg p)$. A CTL^* formula is a state formula. CTL is the fragment of CTL^* where all path formulas are of the form $p\mathsf{U}q$ or of the form $\mathsf{X}p$ where p, q are state formulas. CTL^* formulas are interpreted over Kripke structures (e.g, see [3] for a detailed presentation of the semantics of CTL^*).

Let $K = (S, R, \mathcal{P}, L)$ and $K' = (S', R', \mathcal{P}, L')$ be two Kripke structures with the same set of atomic propositions. A bisimulation between K and K' is a binary relation $U \subseteq S \times S'$ such that, for every $(s, s') \in U$, the following conditions are all satisfied: (1) $L(s) = L'(s')$; (2) for every t such that $(s, t) \in R$, there exists $t' \in S'$ such that $(t, t') \in U$ and $(s', t') \in R'$; and (3) for every t' such that $(s', t') \in R'$, there exists $t \in S$ such that $(t, t') \in U$ and $(s, t) \in R$. We say that a state $s \in S$ is bisimilar to a state $s' \in S'$, if there exists a bisimulation U between K and K' such that $(s, s') \in U$. It is well-known that bisimilar states satisfy the same CTL^* formulas.

We define a predicate over a set S as a subset of S. Let ϕ be a bijection on S, i.e., a one-to-one mapping from S to S. Let C be a predicate over S. Let $f(C)$ denote the set $\{f(x) : x \in C\}$. Let f^{-1} denote the inverse of the bijection ϕ. If f, g are two bijections then we let fg denote their composition in that order; note that in this case, fg is also a bijection. Throughout the paper we use the following identity relating the inverse and composition operators: $(fg)^{-1} = g^{-1}f^{-1}$.

Let $G = (S, E)$ be the reachability graph of a concurrent program where S denotes a set of nodes/states and $E \subseteq S \times S$. An automorphism of G is a bijection on S such that, for all $s, t \in S$, $(s, t) \in E$ iff $(f(s), f(t)) \in E$. We say that an automorphism respects a predicate C over S if $f(C) = C$. The set of all automorphisms of a graph forms a group $Aut(G)$. Given a set $P_1, ..., P_k$ of predicates over S, the set of automorphisms of G that respect $P_1, ..., P_k$ form a subgroup of $Aut(G)$.

Let $\mathcal{G}$ be a group of automorphisms of G. We say that states $s, t \in S$ are equivalent, denoted by $s \equiv_{\mathcal{G}} t$, if there exists some $f \in \mathcal{G}$ such that $t = f(s)$. As observed in [2, 4, 10], $\equiv_{\mathcal{G}}$ is an equivalence relation. A *quotient structure* of G with respect to $\mathcal{G}$ is a graph $(\bar{S}, \bar{E})$ where $\bar{S}$ contains exactly one node in each equivalence class of $\equiv_{\mathcal{G}}$ and $(\bar{s}, \bar{t}) \in \bar{E}$ iff there exists some t such that $t \equiv_{\mathcal{G}} \bar{t}$ and $(\bar{s}, t) \in E$. Each state $\bar{s} \in \bar{S}$ represents all states in S that belong to its equivalence class. Different quotient structures can be defined by choosing different representatives for each equivalence class. However, all these structures are isomorphic. We denote by $rep(s, \mathcal{G})$ the representative element of the equivalence class to which s belongs. In what follows, $QS(G, \mathcal{G})$ denotes the quotient structure obtained by choosing a unique representative for each equivalence class.

A predicate P on the edges of G is a subset of $S \times S$. We say that an edge (s,t) in E, satisfies P if $(s,t) \in P$. Let $True$ denote the set $S \times S$. For an edge predicate P and automorphism ϕ on states, let $f(P) = \{(f(s), f(t)) : (s,t) \in P\}$. Given a group $\mathcal{G}$ of automorphisms on G, we can extend the equivalence relation $\equiv_{\mathcal{G}}$ from states in S to edges in E as follows: two edges $e = (s,t)$ and $e' = (s',t')$ are equivalent (written as $e \equiv_{\mathcal{G}} e'$) if there exists some $g \in \mathcal{G}$ such that $s' = g(s)$ and $t' = g(t)$. It is easy to see that $\equiv_{\mathcal{G}}$ on E is an equivalence relation [9].

3 Model Checking using Guarded Annotated Quotient Structures

In this section, we introduce Guarded Annotated Quotient Structures (GQS) as extensions of Annotated Quotient Structures considered in [4, 5]. These structures can be defined with respect to arbitrary automorphisms and can compactly represent the state space of systems that contain little symmetry. For example, consider a resource allocation system composed of a resource controller and three identical user processes, named a, b and c. When multiple user processes request the resource at the same time, the controller process allocates it to one of the requesting users according to the following priority scheme: user a is given highest priority while users b and c have the same lower priority. This system exhibits some symmetry since users b and c are "interchangeable". Now consider a similar system but where the three user processes are given equal priority. This system exhibits more symmetry since all three users are now "interchangeable". Thus, the system without priorities has more symmetry than the system with priorities. A guarded annotated quotient structure allows us to verify systems with reduced symmetry (e.g., a system with priorities) by treating these as if they had more symmetry (e.g., a system without priorities) and without compromising the accuracy of the verification results. For instance, in the state graph G, of the above resource allocation system with priorities, a state s where all three users have requested the resource has only one outgoing edge (granting the resource to user a). By adding two other edges from s (granting the resource to the two other user processes), the state graph H of the system without priorities can be defined. Since H exhibits more symmetry than G, it can be verified more efficiently. Thus, by viewing G as H extended with guards so that G can be re-generated if needed, model checking can be done more efficiently.

Formally, let $H = (S, F)$ be a graph such that $F \supseteq E$ and $Aut(G) \subseteq Aut(H)$, i.e., H is obtained by adding edges to $G = (S, E)$ such that every automorphism of G is also an automorphism of H.[1] Let $\mathcal{H}, \mathcal{G}$ be groups of automorphisms of H and G, respectively, such that $\mathcal{H} \supseteq \mathcal{G}$. As indicated earlier, $\equiv_{\mathcal{H}}$ defines equivalence relations on the nodes and edges of H. For any edge $e \in F$, let $Class(e, \mathcal{H})$ denote the set of edges in the equivalence class of e defined by $\equiv_{\mathcal{H}}$. Let $\mathcal{Q} = \{Q_1, ..., Q_l\}$ be a set of predicates on S such that each automorphism in

[1] Our results can easily be extended to allow the addition of nodes as well as edges. Note that adding edges/nodes to a graph may sometimes reduce symmetry.

$\mathcal{G}$ also respects all the predicates in $\mathcal{Q}$. Let $QS(G, \mathcal{G}) = (\bar{U}, \bar{E})$ be the quotient structure of G with respect to $\mathcal{G}$ as defined earlier.

A *Guarded Annotated Quotient Structure* of $H = (S, F)$ with respect to $\mathcal{H}$, denoted by $GQS(H, \mathcal{H})$, is a triple $(\bar{V}, \bar{F}, C)$ where $\bar{V} \subseteq S$ is a set of states that contains one representative for each equivalence class of states defined by $\equiv_{\mathcal{H}}$ on S, $\bar{F} \subseteq \bar{V} \times \bar{V} \times \mathcal{H}$ is a set of labeled edges such that, for every $\bar{s} \in \bar{V}$ and $t \in S$ such that $(\bar{s}, t) \in F$, there exists an element $(\bar{s}, \bar{t}, f) \in \bar{F}$ such that $f(\bar{t}) = t$, and C is a function that associates a predicate $C(e)$ with each labeled edge $e \in \bar{F}$ such that (1) $C(e) \cap Class(e, \mathcal{H}) = E \cap Class(e, \mathcal{H})$ (i.e., $C(e)$ denotes all edges in $Class(e, \mathcal{H})$ that are edges in the original graph G) and (2), for all $g \in \mathcal{G}$, $g(C(e)) = C(e)$ (i.e., g respects the edge predicate C).

Given a labeled edge $e = (\bar{s}, \bar{t}, f) \in \bar{F}$, $f \in \mathcal{H}$ is called the label of e and denotes an automorphism that can be used to obtain the corresponding original edge in F; the edge predicate $C(e)$ can in turn be used to determine whether this edge is also an edge of G. Labels of edges in $\bar{F}$ and the edge predicate C are used to unwind $GQS(H, \mathcal{H})$ when necessary during model checking, as described later. Note that edge predicates C that satisfy the above conditions always exist: for instance, taking $C(e) = E$ always satisfies the definition. In practice, a compact representation of an edge predicate C satisfying the conditions above can be obtained directly from the description of the concurrent program. For example, in the case of the resource allocation system, the edge predicate $C(e)$ is defined as follows: if the labeled edges e denotes the allocation of the resource to a user, then $C(e)$ asserts that if there is a request from user a then a is allocated the resource; for all other labeled edges, $C(e)$ is the predicate $True$. Similarly, the automorphisms labeling edges in $\bar{F}$ can also have succinct implicit representations. For example, any automorphism induced by permutations of n process indices as considered in [4, 5, 8] can be represented by an array of n variables ranging over n. Tools like SMC [14] and Murphi [10] includes optimized algorithms for representing and manipulating such sets of permutations.

Given a set $\mathcal{Q}$ of predicates over S that are all respected by the automorphisms in $\mathcal{G}$, we define three Kripke structures $K\_Stru(G, \mathcal{Q})$, $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ and $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ derived from $G = (S, F)$, $QS(G, \mathcal{G}) = (\bar{U}, \bar{E})$ and $GQS(H, \mathcal{H}) = (\bar{V}, \bar{F}, C)$, respectively. We show that *these three Kripke structures are pairwise bisimilar, and hence can all be used for CTL^* model checking.* Since $\mathcal{G}$ is a subgroup of $\mathcal{H}$, each equivalence class of $\equiv_{\mathcal{H}}$ is a union of smaller equivalences classes defined by $\equiv_{\mathcal{G}}$. Thus, the number of equivalence classes of $\equiv_{\mathcal{H}}$ is smaller than those of $\equiv_{\mathcal{G}}$, and $GQS(H, \mathcal{H})$ contains (possibly exponentially) fewer nodes than $QS(G, \mathcal{G})$. $QS(G, \mathcal{G})$ itself can be much smaller than G.

For each predicate Q_j $(1 \leq j \leq l)$ in $\mathcal{Q}$, we introduce an atomic proposition denoted q_j. Let $\mathcal{X} = \{q_i : 1 \leq i \leq l\}$. Let $K\_Stru(G, \mathcal{Q})$ denote the Kripke structure $(S, E, \mathcal{X}, L)$ where for any $s \in S$, $L(s) = \{q_j : s \in Q_j\}$. The Kripke structure $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ is given by $(\bar{U}, \bar{E}, \mathcal{X}, M)$ where $M(\bar{s}) = \{q_j : \bar{s} \in Q_j\}$. The following theorem has been proven in [4, 2, 10].

Theorem 1. *There exists a bisimulation between the structures $K\_Stru(G, \mathcal{Q})$ and $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ such that every state $s \in S$ is bisimilar to its representative in $\bar{U}$.*

Therefore, any CTL^* formula over atomic propositions in $\mathcal{X}$ is satisfied at a state s in $K\_Stru(G, \mathcal{Q})$ iff it is satisfied at its representative $rep(s, \mathcal{G})$ in $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$.

If the edge predicate C is implicitly represented by a collection of edge predicates $\Theta_1, ..., \Theta_r$, the Kripke structure $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ is obtained from $GQS(H, \mathcal{H})$ by partially unwinding it and by tracking the node predicates in $\mathcal{Q}$ (i.e., the predicates $Q_1, ..., Q_l$) and the edge predicates $\Theta_1, ..., \Theta_r$ during this unwinding process. In other words, the unwinding is performed with respect to the predicates $Q_1, ..., Q_l$ and $\Theta_1, ..., \Theta_r$, not with respect to the states of G, in order to limit the unwinding as much as possible. This partial unwinding can be viewed as a particular form of "predicate abstraction", and is a generalization of the unwinding process described in [4, 5]. Precisely, the Kripke structure $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ is the tuple $(W, T, \mathcal{X}, N)$ where W, T and N are defined as follows:

- For all $\bar{s} \in \bar{V}$, $(\bar{s}, Q_1, ..., Q_l,\ \Theta_1, ..., \Theta_r) \in W$.
- Let $u = (\bar{s}, X_1, ..., X_l, \Phi_1, ..., \Phi_r)$ be any node in W, $e = (\bar{s}, \bar{t}, f)$ be a labeled edge in $\bar{F}$ and j be an integer such that Θ_j is the edge predicate $C(e)$. Further, assume that the edge $(\bar{s}, f(\bar{t}))$ satisfies the predicate Φ_j. For all such u and e, the node $v = (\bar{t}, f^{-1}(X_1), ..., f^{-1}(X_l), f^{-1}(\Phi_1), ..., f^{-1}(\Phi_r))$ is in W and the edge (u, v) is in T.
- For all $u = (\bar{s}, X_1, ..., X_l, \Phi_1, ..., \Phi_r) \in W$, $N(u) = \{q_i : \bar{s} \in X_i\}$.

The following theorem states that $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ and $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ are bisimilar.

Theorem 2. *Given $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ and $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ as previously defined, let $Z \subseteq \bar{U} \times W$ be a binary relation defined such that $(s, u) \in Z$ iff there exists an automorphism $f \in \mathcal{H}$ such that $f(t) = s$ and $u = (t, f^{-1}(Q_1), ..., f^{-1}(Q_l), f^{-1}(\Theta_1), ..., f^{-1}(\Theta_r))$. Then, the following properties hold:*

1. *Z is a bisimulation between $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ and $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$.*
2. *For all $u \in W$, there exists a node $s \in \bar{U}$ such that $(s, u) \in Z$.*
3. *Two nodes $u = (t, X_1, ..., X_l, \Phi_1, ..., \Phi_r)$ and $u' = (t', Y_1, ..., Y_l, \Delta_1, ..., \Delta_r)$ of $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ are related to a single node s of $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ through Z iff $t = t'$ and there exists some h in $\mathcal{H}$ such that $h(t) = t$ and $X_i = h(Y_i)$ for all $i = 1, ..., l$, and $\Phi_j = h(\Delta_j)$ for all $j = 1, ..., r$.*

From the previous theorem, we see that multiple nodes in $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ can be related through Z to a single node in $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$. Hence, in principle, $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ can sometimes have more nodes than $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$. The following construction can be used to further reduce the number of nodes in $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ so that the reduced structure has no more nodes than $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$. First, observe that all the nodes in $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ that are related through Z to a single node s in $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ can be represented by a single node since they are all bisimilar to each other. The algorithm for generating $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ can be modified to apply this reduction to construct a smaller Kripke structure $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$. Nodes in $GQS\_Stru(H, \mathcal{H}, \mathcal{Q})$ that are related to a single node in $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ can be detected by evaluating the condition stated in Part 3 of Theorem 2. It can be shown that, if $\mathcal{G}$ is the maximal subgroup of $\mathcal{H}$ consisting of all automorphisms of G that respect $Q_1, ..., Q_l$, then $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$ has the same number of nodes as $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$ and Z defines an isomorphism between the two structures; otherwise, $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$ has fewer nodes than $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$.

In summary, the procedure for incrementally constructing the reachable part of $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$ from $GQS(G, \mathcal{H})$ is the following. We maintain a set $To\_explore$ of nodes that have yet to be treated. Initially, $To\_explore$ contains nodes of the form $(s_0, Q_1, ..., Q_l, \Theta_1, ..., \Theta_r)$ where s is the representative of an equivalence class containing an initial state. We iterate the following procedure until $To\_explore$ is empty. We remove a node $u = (t, X_1, ..., X_l, \Phi_1, ..., \Phi_r)$ from $To\_explore$. For each labeled edge $e = (t, t', f)$ in $GQS(G, \mathcal{H})$, we check if the edge $(t, f(t'))$ satisfies the edge predicate Φ_j, where j is the index such that Θ_j is the edge predicate $C(e)$. If this condition is satisfied we do as follows. We construct the node $v = (t', Y_1, ..., Y_l, \Delta_1, ..., \Delta_r)$ where $Y_i = f^{-1}(X_i)$ for $1 \leq i \leq l$ and $\Delta_j = f^{-1}(\Phi_j)$ for $1 \leq j \leq r$. Then, we check if there exists a node $w = (t', Z_1, ..., Z_l, \Psi_1, ..., \Psi_r)$ in the partially constructed $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$ and a $h \in \mathcal{H}$ such that $t' = h(t')$ and $Z_i = h(Y_i)$ for all $i = 1, ..., l$, and $\Psi_j = h(\Delta_j)$ for all $j = 1, ..., r$ (i.e., the condition of Part 3 of Theorem 2 is checked). If this condition is satisfied, we add an edge from u to w; otherwise, we add v as a new node, include it in $To\_explore$ and add an edge from u to v.

Consider a CTL^* formula ϕ defined over a set $prop(\phi)$ of atomic propositions that each corresponds to a predicate in $\mathcal{Q}$. Let $pred(\phi) \subseteq \mathcal{Q}$ denote the set of predicates corresponding to $prop(\phi)$. From Theorem 2, it is easy to see that the formula ϕ is satisfied at node s in $K\_Stru(G, \mathcal{Q})$ iff it is satisfied at the node $u = (rep(s, \mathcal{H}), f^{-1}(R_1), ..., f^{-1}(R_m), f^{-1}(\Theta_1), ..., f^{-1}(\Theta_r))$ in the structure $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ where f is the automorphism such that $s = f(rep(s, \mathcal{H}))$. Thus, model checking the CTL^* formula ϕ can be done on the Kripke structures $GQS\_Stru(H, \mathcal{H}, pred(\phi))$ or $Greduced\_Stru(H, \mathcal{H}, pred(\phi))$ obtained by unwinding $GQS(H, \mathcal{H})$ with respect to the set $pred(\phi)$ of predicates only. Let us call this the *direct approach.*

4 Formula Decomposition and Sub-formula Tracking

In this section, we discuss two complementary techniques that can improve the direct approach of the previous section.

4.1 Formula Decomposition

Any CTL^* state formula ϕ can be rewritten as a boolean combination of atomic propositions and existential sub-formulas of the form $\mathsf{E}\phi'$. Let $Eform(\phi)$ denote the set of existential sub-formulas of ϕ that are not sub-formulas of any other existential sub-formula of ϕ (i.e., they are the top-level existential sub-formulas of ϕ). Checking whether a state s satisfies a state formula ϕ can be done by checking whether s satisfies each sub-formula in $Eform(\phi)$ separately, and then combining the results.

For each $\phi' \in Eform(\phi)$, we can determine whether s satisfies ϕ' in the structure $K\_Stru(G, \mathcal{Q})$ by unwinding $GQS(H, \mathcal{H})$, with respect to the predicates in $pred(\phi')$ only, to obtain the Kripke structure $GQS\_Stru(H, \mathcal{H}, pred(\phi'))$ and by checking if the corresponding node satisfies ϕ' in this structure. Formulas in $Eform(\phi)$ that have the same set of atomic propositions can be grouped and their satisfaction can be checked at the same time using the same unwinding. Obviously, unwinding with respect to smaller sets of predicates can yield dramatic performance improvements.

Correlations between predicates can also be used to limit the number of unwindings necessary for model checking. Two predicates Q_i and Q_j in $\mathcal{Q}$ are *correlated* if, for all $f \in \mathcal{H}$, $f(Q_i) = Q_i$ iff $f(Q_j) = Q_j$. It is easy to see that the relation "correlated" is an equivalence relation. We say that two atomic propositions are correlated if their corresponding predicates are correlated. Correlations between predicates can sometimes be detected very easily. For instance, with the framework of [4, 5] where automorphisms induced by process permutations are considered, two predicates referring to variables of a same process are correlated: the predicates $x[1] = 5$ and $y[1] = 10$ are correlated if $x[1]$ and $y[1]$ refer to the local variables x and y of process 1, respectively.

If two predicates Q_i and Q_j are correlated, the following property can be proven: if C is a subset of $\mathcal{Q}$ containing Q_i and $C' = C \cup \{Q_j\}$, then the Kripke structures obtained by unwinding with respect to either C or C' will be isomorphic. The above property allows us to combine unwindings corresponding to different formulas in $Eform(\phi)$ whose atomic propositions are correlated. First, we define an equivalence relation among formulas in $Eform(\phi)$: two formulas x and y in $Eform(\phi)$ are equivalent if every atomic proposition in x is correlated to some atomic proposition in y, and vice versa. This equivalence relation partitions $Eform(\phi)$ into disjoint groups $G_1, ..., G_w$. Let $pred(G_i) = \{\cup pred(\phi') : \phi' \in G_i\}$. Now for each group G_i, we can unwind $GQS(H, \mathcal{H})$ with respect to $pred(G_i)$ and check whether each formula in G_i is satisfied at $rep(s, \mathcal{H})$.

The number of unwindings can be further reduced by ordering the groups $G_1, ..., G_w$ as follows. We say that G_i *is above* G_j if every predicate in $pred(G_j)$ is correlated to some predicate in $pred(G_i)$. The relation "above" is a partial order. We call G_i a *top-group* if there is no group above it. Observe that, if G_i is above G_j, we can combine their unwindings. Hence, if $H_1, .., H_v$ denote the top-groups defined by the groups $G_1, ..., G_w$ ($v \leq w$), we can unwind $GQS(H, \mathcal{H})$ with respect to the predicates in $pred(H_i)$ for each group H_i separately, and check the satisfaction in state s of each formula in H_i and in all the groups G_i "below" it using this unwinding.

Note that using the formula decomposition technique can sometimes be less efficient than the direct approach of the previous section. This can be the case when there is a lot of overlap between the sets $pred(H_i)$ of predicates corresponding to the groups H_i obtained after partitioning $Eform(\phi)$.

4.2 Sub-formula Tracking

A CTL^* formula sometimes exhibits itself some internal symmetry. Exploiting formula symmetry was already proposed in [4]. Here, we generalize these ideas by presenting a unified unwinding process where decomposition and symmetry in a formula can be both exploited simultaneously.

Let ϕ be a CTL^* formula. Consider two state sub-formulas ϕ' and ϕ'' of ϕ. We say that ϕ' *dominates* ϕ'' in ϕ if ϕ'' is a sub-formula of ϕ' and every occurrence of ϕ'' in ϕ is inside an occurrence of ϕ'. We say that ϕ' and ϕ'' are *independent* in ϕ if neither of them dominates the other in ϕ. Thus, formulas that are not sub-formulas of each other are independent. Note that even if a formula is a sub-formula of another formula, it is possible for them to be independent: for instance, in the formula q given by $\mathsf{E}(\mathsf{EG}q_1 \,\mathsf{U}\, \mathsf{E}(q_1 \mathsf{U} q_2))$, the state sub-formulas q_1 and $\mathsf{E}(q_1 \mathsf{U} q_2)$ are independent since there is an occurrence of q_1 which does not appear in the context of $\mathsf{E}(q_1 \mathsf{U} q_2)$. Let $Sform(\phi)$ be the set of all sub-formulas of ϕ that are state formulas. Let $\mathcal{R}$ be a subset of $Sform(\phi)$. We say that $\mathcal{R}$ is a *maximal independent set* if it is a maximal subset of $Sform(\phi)$ such that the state formulas in $\mathcal{R}$ are all pairwise independent. There can be many such maximal independent subsets of $Sform(\phi)$. For instance, the set of all atomic propositions appearing in ϕ is obviously a maximal independent set. For the formula q given above, the set consisting of $\mathsf{EG}q_1$ and $\mathsf{E}(q_1 \mathsf{U} q_2)$ is a maximal independent set.

In what follows, we are interested in exploiting "good" maximal independent sets, i.e., sets $\mathcal{R}$ whose elements are symmetric or partially symmetric. A formula q is *symmetric* if, for every automorphism f in $\mathcal{G}$, $f(q) = q$; it is *partially symmetric* when this property holds for almost all f in $\mathcal{G}$. In general, detecting whether a sub-formula is symmetric is computationally hard. However, when syntactically symmetric constructs (similar to those in $ICTL^*$ [5]) are used, it is then easy to determine whether a sub-formula is symmetric. For instance, when only process permutations are used as automorphisms (as in [4, 5]), the sub-formula $\wedge_{i \in I} h(i)$ is symmetric when I is the set of all process indices and $h(i)$ is a formula that only refers to the local variables of process i; the same sub-formula is partially symmetric when I contains most process indices.

Let $\mathcal{R} = \{R_1, ..., R_m\}$ be a (preferably good) maximal independent set of sub-formulas of ϕ. We also view each element R_i of $\mathcal{R}$ as a predicate, i.e., as the set of states that satisfy the CTL^* formula R_i. Consider the Kripke structure $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ obtained by unwinding $GQS(H, \mathcal{H})$ with respect to $\mathcal{R}$. In a similar way, we can define $Greduced\_Stru(H, \mathcal{H}, \mathcal{R})$ following the procedure of Section 3.

Let ψ denote the formula obtained from ϕ by replacing every occurrence of the sub-formula R_i by a fresh atomic proposition r_i, for all $i = 1, ..., m$. The following theorem relates the satisfaction of ϕ and ψ.

Theorem 3. *Let s be a state in S and f be an automorphism in $\mathcal{H}$ such that $s = f(rep(s, \mathcal{H}))$. Then, the formula ϕ is satisfied at state s in the structure $K\_Stru(G, \mathcal{Q})$ iff ψ is satisfied at the node $u = (rep(s, \mathcal{H}), f^{-1}(R_1), ..., f^{-1}(R_m), f^{-1}(\Theta_1), ..., f^{-1}(\Theta_r))$ in the structure $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ iff ψ is satisfied at node u in the structure $Greduced\_Stru(H, \mathcal{H}, \mathcal{R})$.*

Thus, the previous theorem makes it possible to check a formula ϕ "hierarchically", by recursively checking sub-formulas R_i and then combining the results via the unwinding of $GQS(H, \mathcal{H})$ with respect to $\mathcal{R}$ only.

We now discuss the construction of the structures $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ and $Greduced\_Stru(H, \mathcal{H}, \mathcal{R})$. The states of both of these structures are of the form $(\bar{s}, X_1, ..., X_m, \Phi_1, ..., \Phi_r)$, where each X_i is a CTL^* state formula obtained by applying some automorphism to R_i during the unwinding process. Remember that, during the construction process, we need to be able to check whether a newly generated node $v = (\bar{t}, Y_1, ..., Y_m, \Delta_1, ..., \Delta_r)$ is the same as some previously generated node $u = (\bar{s}, X_1, ..., X_m, \Phi_1, ..., \Phi_r)$, i.e., whether $\bar{s} = \bar{t}$, $Y_i = X_i$ for all $i = 1, ..., m$, and $\Delta_j = \Phi_j$ for all $j = 1, ..., r$. Checking whether $\bar{s} = \bar{t}$ and $\Delta_j = \Phi_j$ for all $j = 1, ..., r$ can usually be done efficiently as previously discussed. However, checking whether $Y_i = X_i$ can be hard since each of these can now be any CTL^* state formulas, and checking equivalence of such formulas is computationally hard in general. Note that, if the CTL^* formula ϕ uses syntactically symmetric constructs such as those in $ICTL^*$ [4], then this check can always be done efficiently.

Another important aspect in the construction of $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ is the generation of $N(\bar{s})$ for each state $\bar{s}$. For a node $u = (\bar{s}, X_1, ..., X_m, \Phi_1, ..., \Phi_r)$, $r_i \in N(u)$ iff $\bar{s} \in X_i$. Since X_i can now be any CTL^* state formula, this means that $\bar{s} \in X_i$ iff $\bar{s}$ satisfies the formula X_i in the Kripke structure $K\_Stru(G, \mathcal{Q})$. Since X_i is obtained by applying a sequence of automorphisms in $\mathcal{H}$ to the state sub-formula R_i of ϕ, we know that $X_i = f(R_i)$ for some $f \in \mathcal{H}$. This automorphism f can be made available at the time of generation of u by maintaining automorphisms with states in the set $To\_explore$ used in the algorithm for generating $Greduced\_Stru(H, \mathcal{H}, \mathcal{Q})$ given in Section 3. Thus, checking whether $\bar{s} \in X_i$ reduces to checking whether $\bar{s}$ satisfies the sub-formula $f(R_i)$ in $K\_Stru(G, \mathcal{Q})$, which itself holds iff $f^{-1}(\bar{s})$ satisfies R_i in $K\_Stru(G, \mathcal{Q})$. The latter can be checked by recursively applying the above procedure to R_i instead of ϕ.

We thus obtain a complete recursive procedure which constructs different structures corresponding to the different sub-formulas R_i of ϕ. Note that the formula decomposition technique of Section 4.1 can be used to decompose sub-formulas R_i. Thus, formula decomposition and sub-formula tracking are complementary and can be both applied recursively. It is to be noted that if no good maximal independent set $\mathcal{R}$ can be found then the procedure of Subsection 4.1 should be applied directly.

Example

We illustrate the method by a brief example. Assume that we are using automorphisms induced by process permutations, as in [4, 5]. Consider a concurrent system of n processes. Consider the problem of model-checking with respect to the formula ϕ given by $\mathsf{E}(q_1 \mathsf{U} \wedge_{i \in I} \mathsf{E}h(i))$ where $h(i)$ is a path formula with no further path quantifiers and it only refers to the local propositions of process i, I is the set of all process indices excepting process 1, q_1 is the local proposition of process 1. Let ϕ' denote the sub-formula $\wedge_{i \in I} \mathsf{E}h(i)$. This is a partially symmetric sub-formula. We take $\mathcal{R}$ to be the set $\{q_1, \phi'\}$, since it is a "good" maximal independent set.

We construct $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$. Let M be the total number of nodes in $GQS(H, \mathcal{H})$. M can be exponentially smaller than the number of nodes in the full reachability graph, i.e., the number of nodes in $K\_Stru(G, \mathcal{Q})$. It is not difficult to show that the number of nodes in $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$ is at most nM. During the construction of $GQS\_Stru(H, \mathcal{H}, \mathcal{R})$, we need to determine which of its nodes satisfy the sub-formula ϕ'. To determine this, we invoke the procedure of subsection 4.1 only once. During this procedure, for each $i \in I$, we determine the nodes that satisfy the sub-formula $\mathsf{E}h(i)$. This is done by unwinding $GQS(H, \mathcal{H})$. The resulting structure is also of size at most nM. Thus the over all complexity of this procedure is $O(n^2 M)$.

However, if we use the direct approach and unwind $GQS(H, \mathcal{H})$ (or if we use $QS\_Stru(G, \mathcal{G}, pred(\phi))$) then we will get the full reachability graph. Thus we see that the above example is a case for which the method of this section is exponentially better than the direct approach; (an example program is the resource controller with n identical user processes). On other hand, one can give examples where the direct method is better than the method of this section. As observed, this occurs for cases when the formula has no symmetric (or partially symmetric) sub-formulas. It is to be noted that the formula ϕ, given above, is not an $ICTL^*$ formula and hence the methods of [4, 5] can't be applied.

5 Experimental Results

In this section, we report some preliminary experimental results evaluating the techniques proposed in this paper. Experiments were performed in conjunction with the SMC tool [14]. A first example is the simple resource allocation system described at the beginning of Section 3. We considered a variant of the system with priorities where user 1 is given higher priority than all other users. We checked the following property for various values of i: is it possible to reach a global state where one of the first i users is holding the resource and the resource is still available?

We used two approaches to check the above property. Both approaches give correct answer. The first approach employs the structure $QS\_Stru(G, \mathcal{G}, \mathcal{Q})$; here $\mathcal{G}$ is the set of automorphisms induced by process permutations that fix each of the first i processes and arbitrarily permute the other user processes. The second approach uses formula decomposition of Section 4.1. The decomposed sub-formulas are checked by unwinding $GQS(H, \mathcal{H})$ with respect to the atomic

Value of i	First Approach i.e., employing QS_Stru	Second approach,i.e., using formula decomposition
2	14/2676	14/1863
3	19/3260	16/1864
4	39/4270	18/1865
5	130/6505	20/1866
6	575/11404	22/1867

Table 1. Comparison of the two approaches.

predicates of the sub-formulas independently; here $\mathcal{H}$ is the of automorphisms, induced by process permuations, that arbitrarily permute all the user processes. Formula decomposition was performed manually and SMC was used to check the sub-cases.

Table 1 compares the run-time and memory usage of the two approaches, for the resource allocation system described above with a total number of 80 user processes. Each entry in the table has the form x/y where x is the run-time in seconds and y is the memory usage in Kbytes. Clearly, the second approach, i.e. the approach with formula decomposition, performs better than the first approach; the difference in their performances becomes more pronounced for larger values of i.

We also performed experiments using the Fire-wire protocol (with administrator module) considered in [14], using a configuration with three stations. We checked whether it is possible for either stations 1 or 2 not to receive an acknowledgment after a message is sent. Again, we compared the above two approaches. The first approach took 80 seconds and used 24 Mbytes of memory to complete the verification, while the second approach (i.e. the direct approach with formula decomposition) took 58 seconds and used 12.8 Mbytes of memory.

6 Conclusion and Related Work

We have presented new algorithmic techniques for exploiting symmetry in model checking. We have generalized symmetry reduction to a larger class of automorphisms, so that systems with little or no symmetry can be verified more efficiently using symmetry reduction. We also presented novel techniques based on formula decomposition and sub-formula tracking. Preliminary experimental results are encouraging. Full implementation, and further evaluation with respect to real world examples, needs to be carried out as part of future work.

As mentioned earlier, symmetry reduction in model checking has been extensively studied in [10, 2, 4, 5, 8, 12, 13, 6, 7]. The problem of verifying properties of systems with little or no symmetry was first considered in [6, 7]. The work presented in [7] considered also considers general automorphisms. There, only the verification of symmetric properties was discussed. In contrast, our algorithms can be used to verify any property specified in CTL^*, even if the property is not symmetric. [5] presents a verification method for $ICTL^*$ formulas. Our sub-formula tracking technique can also be used to efficiently verify properties specified in $ICTL^*$, in addition to being applicable to any CTL^* formula. Formula symmetry was explicitly considered in [4] where quotient structures are constructed with respect to automorphisms representing symmetries of the program as well as of the formula. Our sub-formula tracking technique indirectly uses formula symmetry dynamically as the GQS is unwound.

References

1. Aggarwal S., Kurshan R. P., Sabnani K. K.: *A Calculus for Protocol Specification and Validation.* in Protocol Specification, Testing and Verification III, H. Ruden, C. West (ed's), pp19–34, North-Holland, 1983.
2. Clarke, E. M., Filkorn, T., Jha, S.: *Exploiting Symmetry in Temporal Logic Model Checking.* CAV93, LNCS **697** Springer-Verlag, 1993.
3. Emerson, E. A.: *Temporal and modal logic.* In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science.* Elsevier/MIT Press, Amsterdam/Cambridge, 1990.
4. Emerson, E. A., Sistla, A. P.: *Symmetry and Model Checking.* CAV93, LNCS **697** Springer-Verlag, 1993; journal version appeared in Formal Methods in System Design, 9(1/2),1996, pp 105-130.
5. Emerson, E. A., Sistla, A. P.: *Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach.* CAV95, LNCS **939** Springer-Verlag, 1995.
6. Emerson E. A., Treffler R., *From Symmetry to Asymmetry: New techniques for Symmetry Reduction in Model-checking*, Proc. of CHARME 1999.
7. Emerson E. A., Havlicek J. W., *Virtual Symmetry Reductions*, Proc. of LICS 2000.
8. Gyuris, V., Sistla, A. P.: *On-the-Fly Model Checking under Fairness that Exploits Symmetry.* CAV97, LNCS **1254** Springer-Verlag, 1997; To appear in Formal Methods in System Design.
9. Godefroid, P.: *Exploiting Symmetry when Model-Checking Software*, Proceedings of FORTE/PSTV'99, Beijing, 1999.
10. Ip, C. N., Dill, D. L.: *Better Verification through Symmetry.* Formal Methods in System Design **9** 1/2, pp41–75, 1996.
11. Jensen, K.: *Colored Petri Nets: Basic Concepts, Analysis Methods, and Practical Use, Vol2.* Analysis Methods, EATCS Monographs, Springer-Verlag, 1994.
12. Jha, S.: *Symmetry and Induction in Model Checking*, Ph. D. Thesis, Computer Science Department, Carnegie-Mellon University, 1996.
13. Kurshan, R. P.: *Computer Aided Verification of Coordinated Processes: The Automata Theoretic Approach*, Princeton University Press, Princeton NJ, 1994.
14. Sistla A. P., Gyuris V., Emerson E. A., *SMC: A Symmetry based Model Checker for Verification of Safety and Liveness Properties*, ACM Transactions on Software Engineering Methodologies, Vol 9, No 2, pp 133-166, April 2000.