Symmetry for the Analysis of Dynamic Systems

Zarrin Langari and Richard Trefler*

David R. Cheriton School of Computer Science
University of Waterloo, ON, Canada
{zlangari,trefler}@cs.uwaterloo.ca

Abstract. Graph Transformation Systems (GTSs) provide visual and explicit semantics for dynamically evolving multi-process systems such as network programs and communication protocols. Existing symmetry reduction techniques that generate a reduced, bisimilar model for alleviating state explosion in model checking are not applicable to dynamic models such as those given by GTSs. We develop symmetry reduction techniques applicable to evolving GTS models and the programs that generate them. We also provide an on-the-fly algorithm for generating a symmetry-reduced quotient model directly from a set of graph transformation rules. The generated quotient model is GTS-bisimilar to the model under verification and may be exponentially smaller than that model. Thus, analysis of the system model can be performed by checking the smaller GTS-bisimilar model.

1 Introduction

Model checking is used to analyze finite state program models. Many of these models are composed of similar components. In practice, the number of components in these models may be dynamically changing within a given upper bound. For instance, for many communication protocols, the given bound arises naturally due to inherent limitations on system size. Examples of dynamic systems composed of similar components include communication protocols such as IP-telephony protocols where telephony features are dynamically assembled in a call over the Internet [15], network programs with a variable number of clients, and object-oriented systems such as dynamic heap allocation programs [12].

Due to the use of similar components, symmetry is often a feature of the above system models that can be exploited to reduce the state space of a model under verification. Unfortunately, existing symmetry-reduction methods [13,7,9] are not applicable to dynamic systems. In addition, they may offer only limited reduction to system models that are not fully symmetric. *Full symmetry* causes the system model to be invariant under arbitrary rearranging of the components, resulting in an exponential reduction by defining an equivalence relation on symmetric states of the system model. An example of a fully symmetric system model

* The authors' research is supported in part by the NSERC of Canada. Zarrin Langari is currently in McMaster University, Hamilton, ON, Canada; and is supported in part by Mathematics of Information Technology and Complex Systems (MITACS).

M. Bobaru et al. (Eds.): NFM 2011, LNCS 6617, pp. 252–266, 2011.

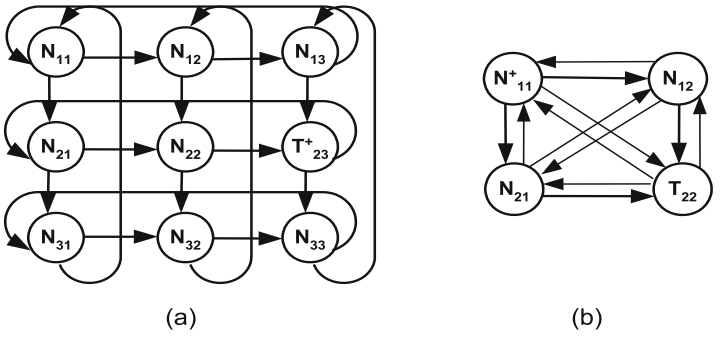

Fig. 1. a) A non-fully symmetric 3×3 and b) a fully symmetric 2×2 toroidal mesh

with four components is illustrated in Figure 1-b. We propose a symmetry reduction method for analyzing visual models of dynamically evolving systems. Our symmetry reduction approach is applicable to non-fully symmetric system architectures such as hypercube, ring, and torus (used in metropolitan area networks that need high scalability) used for modelling next-generation communication and hardware protocols.

Motivation: Graphs provide visual and explicit operational semantics for presenting states and demonstrating structural symmetries of a system. GTSs, which use this graph-based semantics, are straight forward formalisms that offer several key advantages over naive methods in modelling the dynamic evolution of multi-process systems [15,16]. Recently, the GTS formalism has been used to perform reasoning, including verification and error detection, on multi-component, reactive systems [16,11,2,5]. Our motivation is to exploit the advantages that graph-based models provide for the modelling and analysis of dynamically evolving systems.

When systems are composed of several similar components, it is often convenient to identify the various components by their process indices. In a Kripke model of these systems, a state consists of the values of all global variables and the local states of each process. For example, consider a 3×3 toroidal mesh network of processes, as in Figure 1-a. A toroidal mesh is a grid network with wrap-around links, where each process can communicate to two other processes. A shared token is used to show the access of processes to some resource. In this example, the local state T_{23}^{+} describes that the process in row 2, column 3 possesses a token (denoted by a plus sign) and is trying to access a shared resource (denoted by T), and the other processes are in their non-trying modes (denoted by N). Symmetries in these models are then represented as permutations of the process indices. Symmetry-reduction methods [13,7,9,21] use the index permutation to build a symmetry-reduced quotient model that is equivalent, up to permutation, to the behaviour of the original model.

In Kripke models, the labelling of each state does not explicitly show the architecture of the system. On the contrary, in a GTS model of the system, each global state is represented by a graph that explicitly provides the architecture in

which processes are connected together. Since index permutations do not respect the architecture of states, they cannot be used directly to represent symmetries of graph semantics and build equivalence classes of state graphs in non-fully symmetric GTS models. Instead, in graph-based semantic models, symmetries are represented as graph isomorphisms [19] that are used to define an equivalence relation on the set of states presented as graphs.

Contribution: Having several sets of permutations for graphs with different number of nodes, we define a notion of symmetry for a dynamically evolving symmetric multi-process system modelled as a GTS that may grow to a given maximum size. The explicit GTS semantic modelling can directly be exploited for reducing symmetric systems. Our symmetry reduction technique is based on generating a reduced state space directly from the set of graph transformation rules that define the model under verification. For this purpose, we define the notions of *GTS symmetry*, and *GTS bisimulation* based on graph isomorphism. With GTS bisimulation, we describe an on-the-fly algorithm that builds a symmetry-reduced model using the set of graph transformation rules that describe the full dynamic behaviour of the system.

To improve the reduction for symmetric GTS models, we define *vertex bisimulation*. Vertex bisimulation describes an equivalence relation on state graphs based on their set of vertices and can be used in our algorithm for symmetry reduction resulting in an exponential state space saving (*cf* [21]). We also show that two vertex-bisimilar GTS models can prove the same reachability properties given by a subset of CTL. In our method, we use *proposition graphs* to indicate Boolean expressions of atomic propositions. We use proposition graphs, which provide an abstraction of the process indices, to encode symmetric Boolean expressions describing local system states.

Related Work: Ip and Dill [13], Emerson and Sistla [9], and Clarke et al. [7] have been the first who explored symmetry reduction for systems with a fixed number of similar processes. These methods offers only polynomial reductions for most non-fully symmetric systems; thus, in [10,21] the authors have addressed those systems, however, those methods do not apply to graph-based models and, furthermore, are restricted to models with a fixed number of components.

Our approach is also different than approaches such as regular model checking [6] or parameterized verification (*cf* [1]). These methods provide abstractions that generally are not an equivalent representation of the original model. Our method provides an abstraction with an equivalence between the models.

In the area of GTS models, it is only Rensink's [18,19] work that has directly addressed symmetry in GTS models. In [18], a generalized definition of bisimulation is used. This bisimulation is defined for graphs and for developing efficient algorithms to check if two graphs are isomorphic, and not for the GTSs.

The rest of the paper is organized as follows: an overview of GTS modelling is given in Section 2 and is followed by definitions of GTS symmetry and GTS bisimulation in sections 3 and 4. We present vertex bisimulation for symmetric dynamic GTSs in Section 5 and conclude in Section 6.

2 Graph Transformation System Modelling

GTS is a powerful formalism for modelling the semantics of distributed reactive systems [20,8]. In this formalism, graphs are used as the most natural representation of a system [11], where each node represents a process in the system and edges show the direct communication between processes. In previous work [15], GTS modelling was used to represent the dynamic behaviour of a telecommunication system that included the creation and deletion of processes. In a GTS model, each state of the system is specified as a graph. Transformation rules are then used to describe how one state may change to another. The GTS formalism that we use to describe multi-process systems is defined below.

Definition 1 (Graph). *A graph* $G = (V, E, Src, Trg, Lab)$ *consists of a set* V *of nodes, a set* E *of edges, and functions* $Src, Trg : E \to V$*, that define the source and the target of a graph edge, and the labelling function* $Lab : E, V \to l$*, where* l *belongs to a set of labels.*

Definition 2 (Graph Morphism). *Let* $G = (V_G, E_G, Src_G, Trg_G, Lab_G)$ *and* $H = (V_H, E_H, Src_H, Trg_H, Lab_H)$*. A graph morphism* $f : G \to H$ *maps nodes (V) and edges (E) of graph G to nodes and edges of graph H where* $f = (f_v, f_e)$*,* $f_v : V_G \to V_H$*, and* $f_e : E_G \to E_H$ *are structure-preserving functions. That is, we have for all edges* $e \in E_G$*,* $f_v(Src_G(e)) = Src_H(f_e(e))$*,* $f_v(Trg_G(e)) = Trg_H(f_e(e))$*, and* $Lab_H(f_e(e)) = Lab_G(e)$*,* $Lab_H(f_v(v)) = Lab_G(v)$*. If* f_v, f_e *are total functions, then we have a total morphism, and if these are partial functions, and* f_e *is defined on* e*, i.e. there is an* $e' \in E_H$*, such that* $f_e(e) = e'$*,we have a partial morphism.*

Definition 3 (Graph Isomorphism). *In the above definition, if* f*, respectively* f_v *and* f_e*, are bijective functions, then we have a graph isomorphism. We write* $G \cong H$ *if there exists an isomorphism between graphs* G *and* H*.*

If f_v and and f_e map the set of all nodes and edges of graph G respectively, then the morphism is called a *total morphism*. On the other hand, f_v and f_e are *partial morphisms* iff the mapping is not from the whole source graph nodes and edges. Note that in a structure-preserving mapping, the shape and the edge labelling of the original graph are preserved.

Definition 4 (Graph Transformation Rule). *A transformation rule* r *is defined as* $r : L \to R$*, where* L *and* R *are graphs, called the left side graph and the right side graph of the rule, and there is a partial morphism between them.*

To transform a graph, a rule is applied to the graph. The application of a rule r to a graph G, is based on a total morphism between L and G. We write $G_0 \xrightarrow{r} G_1$ to show that the graph G_0 is transformed to G_1 by the application of rule r. In general, the result of applying a rule to a graph is as follows: everything in the left side graph (L) but not in the right side graph (R) will be *deleted*, everything in R which is not in L will be *created*, and everything that is in both sides will be *preserved* [20]. A total match between the left side subgraph of a rule and a

subgraph in the source graph is made, and then the source subgraph is deleted and replaced by the right side subgraph R.

To describe how the states of a system defined as graphs transform as the transformation rules are applied to them repeatedly starting from the initial state, we give the definition of a graph transition system $\mathcal{G} = \langle S, T, I \rangle$.

Definition 5 (Graph Transition System). *A graph transition system is defined as, $\mathcal{G} = \langle S, T, I \rangle$, such that:*

1. *S is a set of states, where each state $s \in S$ has a graph structure denoted as G_s .*
2. *T is a set of transitions : $T \subseteq S \times P \times S$ where P is a set of transformation rules and for all $t \in T$, t is given by $s_1 \xrightarrow{r} s_2$, there is a graph transformation rule $r \in P$ that transforms G_{s_1} to G_{s_2}.*
3. *I is a set of initial state graphs.*

The transformation sequence $s_0 \xrightarrow{r_1} s_1 \xrightarrow{r_2} ... \xrightarrow{r_n} s_n$ is called a GTS derivation. We write $s_0 \xrightarrow{r^*} s_n$ to denote that such a derivation from s_0 to s_n exists. Since in our modelling all the transitions are made by the application of rules, we sometimes omit the r superscript and show a transition as $s_1 \rightarrow s_2$ and a sequence of transitions as a *path*, denoted by $\rightsquigarrow$, e.g. $s_0 \rightsquigarrow s_n$ shows that there is a path between the state s_0 and s_n in the graph transition system.

Later, in sections 4 and 5 we need to prove that a GTS and its bisimilar quotient satisfy the same set of properties. Thus, at first, it is required that we describe how these properties and their propositional formulas are expressed in terms of graphs, and how the property satisfaction is defined for graphs. In our previous work [16], we have defined the notion of *graph satisfaction* and extended the definitions of graph and graph morphism to *regular expression graph (REG)* and *regular expression graph morphism*. Here, we briefly present these definitions again. REGs are used for expressing Boolean expressions of propositions as graphs (called *proposition graphs*) with edges labelled as regular expressions (e.g. Kleene-star labels). Using regular expression graphs in the proposition graphs and the transformation rule graphs makes these graphs more expressive. REGs are used to compactly express component connectivity patterns, for instance, to show that between two components of interest there may be an arbitrary length sequence of intervening components.

Definition 6 (Regular Expression Graph (REG) [16]). *An REG is a graph G where for a set of labels, L, the labelling function Lab is defined as $Lab : E_G \rightarrow \{l^+ \mid l \in L\} \cup \{l^* \mid l \in L\} \cup L$ where l^* and l^+ represent Kleene closure and the positive Kleene closure of l.*

For REG morphism we need to define the notion of a *graph path*. On a graph, a path is defined as a sequence of nodes connected by edges. Hence, the sequence of edge labels in a graph path specifies a string (language).

Definition 7 (REG Morphism [16]). *An REG morphism between G and H, when either G or H is an REG or a graph is defined as, for a path $p = \{v_1, ..., v_n\}$ in G there is a path $q = \{u_1, ..., u_n\}$ in H such that:*

- *There is a graph morphism $m : V_G \rightarrow V_H$ between the beginning and the end nodes of these two paths.*
- *For $2 \leq i \leq n-1$, these cases may occur:*
  1. *If both G and H are REGs, then the language specified by the sequence of corresponding labels over the edges connecting nodes v_i in p is a subset of the language specified by the sequence of labels over the edges connecting nodes u_i in q.*
  2. *If H is an REG, and G is a graph without Kleene-star-labelled elements, then the string specified by the sequence of corresponding labels over the edges connecting nodes v_i in p is a member of the language specified by the sequence of labels over the edges connecting nodes u_i in q.*

We have total or partial REG morphisms, if the mappings are respectively total or partial.

Definition 8 (Graph Satisfaction [16]). *An REG or a graph G satisfies an REG or a graph ϕ, written as $G \models \phi$, iff there exists a total graph or REG morphism m between ϕ and G written as $m : \phi \rightarrow G$.*

We adopt a GTS model with attributed graphs and node identification [20,3], in which nodes are uniquely identified by their attributes.

3 Symmetry in Dynamic GTS Models

We define symmetry for dynamic GTS models of systems which may not be fully symmetric, but that show some symmetry in their structure. Traditionally, for a fixed size system, symmetries are represented by a group of index permutations [7,9]. For GTS systems, we consider states to be symmetric if their associated graphs are isomorphic.

Definition 9 (Graph Permutation). *A permutation $\pi : G \rightarrow H$ is an isomorphism between a graph G and a graph H, $G \cong H$.*

Since π is an isomorphism, it associates vertices and edges of H (V_H, E_H) to vertices and edges of G (V_G, E_G), such that $V_H = V_G$. For example, for a ring graph with three labelled nodes 1, 2, 3, with edges: $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 1$, a permutation π that maps nodes 1 to 2, 2 to 3, and 3 to 1 permutes the graph to the one that has the same set of vertices. Also π associates the edges $1 \rightarrow 2$, $2 \rightarrow 3$, and $3 \rightarrow 1$, respectively, to the edges $2 \rightarrow 3$, $3 \rightarrow 1$, and $1 \rightarrow 2$ in the permuted graph.

In dynamic systems, where the number of components may change, we consider sets of such permutations to define symmetries for different state sizes. In fact, there are different groups of permutations for graphs with different sizes. The state graph permutation implicitly considers the number of nodes in a graph because graph isomorphism is used to define these permutations and isomorphism is based on a bijection on the sets of nodes and edges of the graph. For specific graphs of n nodes, we use the notion π_n to show a permutation on those

graphs. For a ring of size n this permutation is a rotation on an n-node ring. For a $k \times k$ toroidal mesh (where $n = k \times k$), a permutation is either the rotation of k horizontal rings, the rotation of k vertical rings, or a mix of these rotations. $k-1$ horizontal rotations followed by a vertical one or $k-1$ vertical rotations followed by a horizontal one is actually a flip for the $k \times k$ toroidal mesh, where the flip α is defined as $\alpha(i,j) = (j,i)$. These permutations are automorphisms of a toroidal mesh network.

For a specific topology, consider a set composed of a disjoint union of graphs with different sizes. We use the notation $\mathcal{A}_i$ to show a group of graph symmetries, where i denotes size of the graph. The number of groups is finite as we work with GTS models with an upper-bound $maxsize$ on the number of graph nodes. Γ is defined as a new generalized group of symmetries built from the product of groups of permutations of graphs with different sizes. For details on this product and for the reason on why this product forms a group we refer the reader to [14]. Each element of Γ is a tuple $(\pi_1, \pi_2, ..., \pi_n)$ where $\pi_i \in \mathcal{A}_i$. Each π_i can be an identity permutation indicated as e_i, which is a morphism that maps each graph of size i to itself, where $1 \leq i \leq maxsize(\mathcal{G})$. Note that the group $\mathcal{A}_k$ is isomorphic to the subgroup of elements $(e_1, e_2, ..., \pi_k, ..., e_n)$; therefore, for simplicity we indicate $(e_1, e_2, ..., \pi_k, ..., e_n)$ as π_k from now on.

Definition 10 (GTS Symmetry). *A GTS $\mathcal{G} = \langle S, T, I \rangle$ is symmetric with respect to the set of graph permutations Γ if:*

1. *For all $s_1, s_2 \in S$, where s_1 has an associated n-node graph and s_2 has an associated m-node graph, if t is a transition in T such that $t : s_1 \rightarrow s_2$, then for π_n, an n-node symmetry in Γ, there is a path $p \in T^+$, $p : \pi_n(s_1) \rightsquigarrow \pi_m(s_2) \in T^+$ where π_m is an m-node symmetry in Γ.*
2. *For all $s_0 \in I$ where G_{s_0} is an n-node graph associated with state s_0 and for all $\pi_n \in \Gamma$, $\pi_n(G_{s_0}) \in I$.*

A GTS model is fully symmetric if for all transitions and for all arbitrary index permutations on state graphs (not just isomorphisms), the GTS model is invariant. GTS symmetry differs from architectural symmetry defined for fixed-size systems in [21], because in the case that G_{s_1} (an n-node graph associated with state s_1) and G_{s_2} (an m-node graph associated with state s_2) are of the same size, then in the above definition, $m = n$ and $\pi_n = \pi_m$, which means that we have the same permutation for graphs with the same number of nodes. The reason is that both π_n and π_m are isomorphisms on graphs of the same size and architecture. In this case, the path p would be of length one, because for each transition between two state graphs, there is one symmetric transition between their isomorphic state graphs. In addition, the set of symmetries in architectural symmetry differs from those in GTS symmetry, which are based on graph isomorphisms.

In dynamic GTSs, it is important to describe the way the system evolves within a maximum bound. Our methods are applicable when evolution of the system does not change the architecture describing the model structure. For example, the basic building block of a toroidal mesh is a ring, and the toroidal

mesh evolves by the addition of these building blocks. Therefore, the dynamic evolution is done by adding a certain number of k nodes to form a new vertical or horizontal ring to keep a balanced toroidal mesh network. Therefore, in toroidal mesh, $m = n$ (when the toroidal mesh is not dynamic), or $m = n + k$ (when k nodes are added), or $m = n - k$ (when k nodes are deleted).

We use graph isomorphism to build a bisimilar quotient of a GTS model. It is notable that graph isomorphism requires that graphs be of the same size and structure. We can use graph isomorphism as an equivalence relation on a GTS model with state graphs of different sizes. Thus, in state-space reduction, we are looking to cut down the number of isomorphic state graphs belonging to the same equivalence class that are represented during verification.

4 GTS Bisimulation

Using graph isomorphism (Definition 3), we now define GTS bisimulation, and then give an algorithm to generate a reduced bisimilar quotient of a GTS model. Isomorphism provides a strong equivalence relation for generating the quotient, because the same set of transformation rules are applicable to a state in the quotient and the isomorphic state in the original model.

Definition 11 (GTS Bisimulation). *Given two GTSs* $\mathcal{G}_1 = \langle S_1, T_1, I_1 \rangle$ *and* $\mathcal{G}_2 = \langle S_2, T_2, I_2 \rangle$*, a relation* $\sim \subseteq S_1 \times S_2$ *is a GTS bisimulation if* $s_1 \sim s_2$ *implies:*

1. $G_{s_1} \cong G_{s_2}$.
2. *For every* $t_1 \in T_1$, $t_1 : s_1 \to s_1'$, *there is a path* $p_2 \in T_2$ *of length at least one, such that* $p_2 : s_2 \rightsquigarrow s_2'$ *and* $s_1' \sim s_2'$.
3. *For every* $t_2 \in T_2$, $t_2 : s_2 \to s_2'$, *there is a path* $t_1 \in T_1$ *of length at least one such that* $p_1 : s_1 \rightsquigarrow s_1'$ *and* $s_2' \sim s_1'$.

Quotient of a GTS: Let $\mathcal{G} = \langle S, T, I \rangle$ be a GTS with a set P of transformation rules and $\equiv$ is an equivalence relation on S such that $s_1 \equiv s_2$ implies $G_{s_1} \cong G_{s_2}$. If each equivalence class of state graphs is shown as $[s]$, then the quotient structure of a GTS is represented by $\bar{\mathcal{G}} = \langle \bar{S}, \bar{T}, \bar{I} \rangle$ such that

$\bar{S} = \{[s] : s \in S\}, G_{[s]} \cong G_s$,
$\bar{T} = \{[s] \xrightarrow{r} [t] \in \bar{S} \times P \times \bar{S} : \exists s_0 \in [s], t_0 \in [t] : s_0 \xrightarrow{r} t_0 \in T\}$, and
$\bar{I} = \{[s] : s \in I\}$.

4.1 Generating a Bisimilar Quotient

In this section, we present an algorithm for generating a symmetry-reduced GTS model of a dynamically evolving multi-process system. This algorithm (*cf* [9]) provides an on-the-fly generation of the symmetry-reduced model of a GTS-based labelled-transition system. The algorithm may provide an exponential savings in the cost of system analysis for fully symmetric GTS models, but for GTS models with some symmetry we get a polynomial-size reduction.

```
GenerateQuotient(state s_0, T, int n)
Input: s_0: initial state, T: set of GTS rules, n: initial number of processes
Output: E: equivalence classes of states, R: quotient transition relation
1  E[1].st ← s_0
2  CurrentState ← 1, LastState ← 1
   // loops over E to apply the transformation rules
3  while CurrentState ≤ LastState do
4   forall r ∈ T applicable to E[CurrentState].st do
       // applies rule r to a representative state st in table E
5      temp ← Apply(r, E[CurrentState].st)
6      s̄ ← temp.st
7      n̄ ← temp.n
8      stateFound ← false
       // checks if the transformed state is a permutation of the
          existing representative states
9      for i ← 1 to LastState do
          // finds the equivalence class based on the graph size
10      if (E[i].n = n̄) and (s̄ = E[i].st or IsAPermutation(E[i].st, s̄, E[i].n))
        then
11         stateFound ← true
12         AddTransition(R, CurrentState, i)
13         exit for loop
14     endfor
       // the newly found equivalence class is inserted in E
15     if stateFound = false then
16        LastState ← LastState + 1
17        E[LastState].st ← s̄
18        E[LastState].n = n̄
19        AddTransition(R, CurrentState, LastState)
20  endforall
21  CurrentState ← CurrentState + 1
22 endwhile
23 return E, R
```

Fig. 2. Quotient Generation Algorithm

The algorithm GenerateQuotient in Figure 2 accepts a set of graph transformation rules, an initial-state graph labelling, and the initial number of processes as input. As output, it generates a table E: the representatives of the equivalence classes of state graphs, and a table R: the quotient transition relation. Each element in E consists of a single representative state graph (st), and the number of processes in that state graph (n). Table R is a two-dimensional table consisting of pointers to table E. There is a transition between each state in $E[i]$ and the state in $E[R[i, j]]$, where j is an index iterating over all transitions of the state in $E[i]$. By keeping track of the number of processes in each representative state: $E[i].n$, our algorithm works correctly for dynamic architectures in which processes can be added or deleted in the execution path.

In line 10, the algorithm checks that two state graphs with the same size are a permutation of each other. For more clarity, here we consider the node labelling of a state graph instead of the graph itself. The function IsAPermutation iterates over permutations to find the right permutation, and it can be specialized for different topologies. For example, for ring networks, the permutations are circular ones. For a toroidal mesh, they are appropriate horizontal or vertical rotations, or flips. As an example, we have implemented the GTS modelling of mutual exclusion for both a dynamic toroidal mesh and a dynamic token ring in [14].

Theorem 1. *Let $\mathcal{G} = \langle S, T, I\rangle$ be a GTS and symmetric with respect to the set of graph permutations Γ, and $\bar{\mathcal{G}} = \langle \bar{S}, \bar{T}, \bar{I}\rangle$ be the quotient of $\mathcal{G}$, then $\mathcal{G}$ and $\bar{\mathcal{G}}$ are GTS-bisimilar: $\mathcal{G} \sim \bar{\mathcal{G}}$.*

Proof. Consider $\pi_n, \pi_m \in \Gamma$ as graph permutations for a set of state graphs with different number of nodes. The proof considers two claims: 1) for every graph transformation $\bar{s_0} \to \bar{s_1} \in \bar{T}$, there is a corresponding path $p = s_0 \rightsquigarrow s_1$ in $\mathcal{G}$, and 2) for every $s_0 \to s_1 \in T$, there is a corresponding path $\bar{p} = \bar{s_0} \rightsquigarrow \bar{s_1}$ in $\bar{\mathcal{G}}$. We prove the first claim, and the other follows similarly. The proof for each claim is broken into two cases: one for transformations $\bar{s_0} \to \bar{s_1}$ that do not add or delete components (nodes) to or from the start graph $\bar{s_0}$ of size n. The second case considers a transformation that changes the number of components in the source state graph. For the second case, we only consider the addition of components, as proof for the deletion is similar.

Case 1: Choose an arbitrary reachable state $s_0 \in S$ such that $G_{s_0} \cong G_{\bar{s_0}}$. Using on-the-fly generation of the quotient, we know that there exists a transition $\bar{s_0} \to \bar{s_1} \in \bar{T}$ such that $\bar{s_0}$ and $\bar{s_1}$ are equivalence classes of state graphs. Thus, there is a graph $u \in S$ that belongs to the equivalence class of $\bar{s_0}$ and there is a graph $v \in S$ that belongs to the equivalence class of $\bar{s_1}$. Therefore, $u \to v \in T$. Thus, $G_u \cong G_{\bar{s_0}}$ and $G_v \cong G_{\bar{s_1}}$. Since $G_{s_0} \cong G_{\bar{s_0}}$ and $G_u \cong G_{\bar{s_0}}$, by transitivity $G_{s_0} \cong G_u$. Now let G_{s_1} be isomorphic to a permutation of graph v, i.e. $G_{s_1} \cong \pi_n(G_v)$ which implies $G_{s_1} \cong G_v$ and because we had $G_v \cong G_{\bar{s_1}}$, thus $G_{s_1} \cong G_{\bar{s_1}}$. From $u \to v \in T$, $G_{s_0} \cong G_u$, and $G_{s_1} \cong G_v$ we deduce $\pi_n(u) \to \pi_n(v) = s_0 \xrightarrow{r} s_1 \in T$. Inductively, we can prove for each $\bar{s_i} \to \bar{s_{i+1}} \in \bar{T}$, there is a transformation $s_i \xrightarrow{r} s_{i+1} \in T$.

Case 2: In $\bar{s_0} \to \bar{s_1} \in \bar{T}$, we know that $G_{\bar{s_0}}$ is of size n and $G_{\bar{s_1}}$ is of size m, where $m > n$. Choose an arbitrary state $s_0 \in S$ such that $G_{s_0} \cong G_{\bar{s_0}}$. Since $\bar{s_0} \to \bar{s_1} \in \bar{T}$, then based on the quotient generation algorithm, there is a transformation rule $u \xrightarrow{r} v \in T$ in GTS $\mathcal{G}$ such that $G_{\bar{s_0}} \cong \pi_n(G_u)$, $G_{\bar{s_1}} \cong \pi_m(G_v)$. Since $\mathcal{G}$ is GTS-symmetric, then for each transition $u \to v \in T$ there exist permutations π'_n and π'_m such that ${\pi'}_n(G_u) \rightsquigarrow {\pi'}_m(G_v) \in T$, and since permutation is based on isomorphism then every permutation of a graph is isomorphic to it, so ${\pi'}_n(G_u) \cong \pi_n(G_u)$. From $G_{s_0} \cong G_{\bar{s_0}}$, $G_{\bar{s_0}} \cong \pi_n(G_u)$, and ${\pi'}_n(G_u) \cong \pi_n(G_u)$ we have $G_{s_0} \cong \pi_n(G_u)$. Therefore, $s_0 \rightsquigarrow {\pi'}_m(v)$. Let s_1 be the permutation of graph v; hence, ${\pi'}_m(G_v) \cong G_{s_1}$. We had $G_{\bar{s_1}} \cong \pi_m(G_v)$, also we know all permutations of a graph are isomorphic with each other, thus ${\pi'}_m(G_v) \cong \pi_m(G_v)$; hence, we have $G_{\bar{s_1}} \cong G_{s_1}$, and conclude $s_0 \rightsquigarrow s_1$. Inductively, we can prove for each $\bar{s_i} \to \bar{s_{i+1}} \in \bar{T}$ that there is a path in T. □

As a result, we have a theorem about satisfaction of reachability properties, EFf (eventually a long a path) and $\neg$EFf, where f is a propositional formula. In GTS-bisimilar models, a transition matches with a path; therefore, neither X (next-time) nor U (until) operators can be expressed in properties.

Theorem 2. *Let ϕ be an EF formula over a set of atomic graph propositions defined as graphs or REGs. For the graph transition system $\mathcal{G}$, and its quotient $\bar{\mathcal{G}}$ and the property ϕ and state graphs $s_1 \in \mathcal{G}$ and $\bar{s_1} \in \bar{\mathcal{G}}$, where $s_1 \sim \bar{s_1}$ we have $\mathcal{G}, s_1 \models \phi$ iff $\bar{\mathcal{G}}, \bar{s_1} \models \phi$.*

Proof idea. This theorem is a direct consequence of exploiting symmetry and GTS symmetry, and the proof is done using the bisimulation between the GTS $\mathcal{G}$ and its quotient $\bar{\mathcal{G}}$, and it is similar to the proof given in [21].

5 Vertex Bisimulation

In this section, we introduce vertex bisimulation to be used for GTSs that are not fully symmetric. Vertex bisimulation enables an exponential reduction with respect to full symmetry group for GTS models. We require that the transformations of these GTSs preserve the architecture, which is the case that usually occurs in practice, i.e., if the initial state graph architecture is a toroidal mesh, then this architecture is preserved in all state graphs of the model and in the state graphs of the symmetry-reduced model. Even if the structure dynamically evolves, the evolution of components preserve the overall system structure.

Definition 12 (Vertex Bisimulation). *For GTSs $\mathcal{G}_1 = \langle S_1, T_1, I_1 \rangle$ and $\mathcal{G}_2 = \langle S_2, T_2, I_2 \rangle$ a relation $\sim^v \subseteq S_1 \times S_2$ is a vertex bisimulation if $s_1 \sim^v s_2$ implies:*

1. *G_{s_1} and G_{s_2} have the same set of vertices and the same architecture.*
2. *for every $t_1 \in T_1$, $t_1 : s_1 \rightarrow s_1'$, there is a path $p_2 : s_2 \rightsquigarrow s_2' \in T_2$ and $s_1' \sim^v s_2'$.*
3. *for every $t_2 \in T_2$, $t_2 : s_2 \rightarrow s_2'$, there is a path $p_1 \in T_1$ such that $p_1 : s_1 \rightsquigarrow s_1'$ and $s_2' \sim^v s_1'$.*

From a GTS-symmetric model with respect to full symmetry group, we derive a vertex-bisimilar quotient. Thus, we can apply all permutations and obtain full symmetry reduction resulting in an exponential reduction. To be able to gain this reduction without the application of the large set of all permutations, there are techniques that allow the representation of full symmetry-reduced state spaces by a program translation into a symmetry-reduced program text [10,4]. Vertex bisimulation for GTS models is comparable to safety-bisimulation for Kripke models [21], but unlike safety-bisimulation it can be used for dynamic graph models. In the following theorem, we show the vertex-bisimilarity of a model and its quotient. The proof of this theorem is similar to the proof of Theorem 1.

Theorem 3. *Let $\bar{\mathcal{G}} = \langle \bar{S}, \bar{T}, \bar{I} \rangle$ be the quotient model of a GTS-symmetric system $\mathcal{G} = \langle S, T, I \rangle$, then $\mathcal{G}$ is vertex-bisimilar to $\bar{\mathcal{G}}$.*

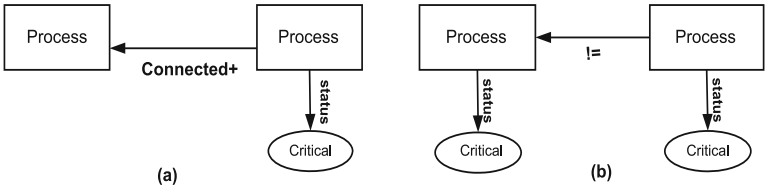

Fig. 3. Two atomic proposition graphs

5.1 Property Preservation

If we prove that the quotient of a GTS-symmetric system is vertex-bisimilar to the original model, then we can use the quotient to prove interesting properties of the system. As stated in [21], one of the problems of verifying properties on the quotient models is that the property should have *symmetric atomic propositions*, that is, permutations of process indices in the property formula leaves the formula and its significant sub-formulas invariant. Expressing Boolean expressions of atomic propositions as graphs and using graph satisfaction (Definition 8) [16] provides an abstraction on the process indices that solves this problem. The reason is that when we use an REG in an atomic proposition and a generic node that represents, for example, any of the processes appearing in the proposition, then we do not need to specify each symmetric part of the atomic proposition explicitly. For example, in a model with three processes, an atomic proposition for expressing that at least one of the processes is in the *Critical* state is: $Critical_1 \vee Critical_2 \vee Critical_3$. Figure 3-a illustrates such an expression in which one process (any of 1, or 2, or 3) is in the *Critical* state and connected to at least one other process. The condition "at least one" has been modelled as an edge labelled with $Connected^+$ between two processes.

As presented in Definition 6, we have used the regular-expression graph in which edges may be labelled with a Kleene-star operator over the set of labels. Therefore, all formulas with the existential process quantifier form, $\vee_i$, can be abstractly modelled as a proposition graph with nodes being an abstraction of process indices. Also, the universal process quantifier form, $\wedge_i$, in a graphical notation, is implicitly presented as all the process nodes that participate in the $\wedge_i$ formula connected together. For instance, in the property $\neg \mathrm{EF}(\exists i \neq j : Critical_i \wedge Critical_j)$ in a toroidal mesh or ring, the Boolean expression of propositions can be expressed as a graph illustrated in Figure 3-b. In this figure, two different processes are presented to be in the *Critical* state.

Thus reachability properties and all the properties that can be expressed in terms of EF, such as AG ϕ which is equal to $\neg$ EF $\neg\phi$, are verifiable on the symmetry-reduced GTS model. For these properties, we prove that for a GTS and its quotient that are vertex-bisimilar, they both satisfy the same properties. Based on this theorem, we can use the vertex-bisimilar reduced GTS model of a system to prove interesting properties of it.

Theorem 4. *Let $\mathcal{G} = \langle S, T, I\rangle$ be a symmetric GTS and $\bar{\mathcal{G}} = \langle \bar{S}, \bar{T}, \bar{I}\rangle$ be the quotient of $\mathcal{G}$ and vertex-bisimilar to it, $\mathcal{G} \sim^v \bar{\mathcal{G}}$. For $s_1 \in S$ and $\bar{s_1} \in \bar{S}$, where $s_1 \sim^v \bar{s_1}$ we have $\mathcal{G}, s_1 \models \phi$ iff $\bar{\mathcal{G}}, \bar{s_1} \models \phi$ where ϕ is an EF formula over a set of atomic propositions defined as graphs or REGs.*

To prove this theorem, first in the lemma below, we show that there is a matching path between two vertex-bisimilar GTSs for GTS symmetric models.

Lemma 1. *Let $\mathcal{G} = \langle S, T, I\rangle$ be a symmetric GTS and $\bar{\mathcal{G}} = \langle \bar{S}, \bar{T}, \bar{I}\rangle$ be its vertex-bisimilar GTS, $\mathcal{G} \sim^v \bar{\mathcal{G}}$. For $s_1 \in S$, $\bar{s_1} \in \bar{S}$, if $s_1 \sim^v \bar{s_1}$ then for any GTS derivation in $\mathcal{G}$, $s_1 \xrightarrow{r^*} s_m$, there is a derivation in $\bar{\mathcal{G}}$, $\bar{s_1} \xrightarrow{r^*} \bar{s_n}$, and vice versa.*

Proof. It is notable that there may not be a one-to-one correspondence between transformations of these two derivations, which means that the lengths of the two derivations may not be the same. We show the proof for ($\Leftarrow$), and the other direction will follow because $\mathcal{G}$ and $\bar{\mathcal{G}}$ are vertex-bisimilar.

For $\bar{p} : \bar{s_1} \xrightarrow{r^*} \bar{s_n}$ in $\bar{\mathcal{G}}$, we prove that there is $p : s_1 \xrightarrow{r^*} s_m$ in $\mathcal{G}$ such that $s_m \sim^v \bar{s_n}$. The proof is shown by breaking the derivation $\bar{p}$ into individual transformations and matching each graph transformation in the derivation $\bar{p}$ to a sequence of transformations in p. Later we match the concatenated transformations in $\bar{p}$ to the concatenated sequence of transformations in p.

For the first transition in $\bar{p}$, if the length of the GTS derivation p is zero, then $s_1 = s_m$, and we have a mapping to a path of length zero. If the length of the GTS derivation p is greater than or equal to one, then based on Definition 12, we have $s_1 \sim^v \bar{s_1}$ and for one transition $\bar{s_1} \to \bar{s_2}$ in $\bar{p}$, there is a derivation in p of length at least one, thus $\bar{s_2} \sim^v s_i$. We proved that for the first transformation in $\bar{p}$, there is a sequence of transformations in $p : s_1 \xrightarrow{r^i} s_i$ where $\bar{s_2} \sim^v s_i$. The same reasoning can be used for the second and subsequent transformations, e.g. $\bar{s_2} \to \bar{s_3}$ is matched to a path from s_i to s_j in p.

We now use induction. As to the hypothesis, consider for a sequence of k transformations in $\bar{p} : \bar{s_1} \xrightarrow{r^k} \bar{s_k}$, there is a sequence of l transformations in $p : s_1 \xrightarrow{r^l} s_l$, such that $s_1 \sim^v \bar{s_1}$ and $s_l \sim^v \bar{s_k}$. Based on the vertex bisimulation definition, for the transformation $\bar{s_k} \to \bar{s}_{k+1}$ in $\bar{\mathcal{G}}$, there is a path $s_l \xrightarrow{r^*} v$ in $\mathcal{G}$, where $\bar{s_k} \sim^v s_l$, and $\bar{s}_{l+1} \sim^v v$, let v be s_{l+1}. Therefore, for $\bar{p} : \bar{s_k} \xrightarrow{r} \bar{s}_{k+1}$ in $\bar{\mathcal{G}}$ there is a derivation $p : s_l \xrightarrow{r^*} s_{l+1}$ in $\mathcal{G}$. We consider the application of the first k transformations and the $k+1$th transformation in $\bar{\mathcal{G}}$ as one GTS derivation: $\bar{s_1} \xrightarrow{r^*} \bar{s_{k+1}}$, and also the first l sequences of transformations and the $l+1$th transformation in $\mathcal{G}$ as the derivation $p : s_1 \xrightarrow{r^*} s_{l+1}$. Let $k+1 = n$ and $l+1 = m$. Thus, we have matched the two derivations. □

Proof (Theorem 4). To prove this theorem, we use the fact that $\mathcal{G}$ is GTS-symmetric, to ensure the preservation of architecture in states of $\mathcal{G}$ and its quotient, even though s_1 and $\bar{s_1}$ only have the same set of vertices. The proof is given for different cases of ϕ. It is sufficient to show the proof for one direction ($\Rightarrow$). The other direction is similar.

Atomic Propositions. The propositional formula is built as a graph with an abstraction on process indices. Therefore, without considering specific indices, if the formula is true for s_1, it is symmetrically true for any other communication graph of processes with the same set of local states. Since s_1 and $\bar{s_1}$ are vertex-bisimilar, they have the same node labelling or the same set of possible local states, and both satisfy the same formula.

EF formula. From $\mathcal{G}, s_1 \models \text{EF } \varphi$, we deduce that there is a derivation $p : s_1 \xrightarrow{r^*} u$ in $\mathcal{G}$, where u is a state graph that satisfies the proposition graph φ. Since $s_1 \sim^v \bar{s_1}$ and based on Lemma 1, we know that for each derivation p in $\mathcal{G}$, there is a matching derivation $\bar{p} : \bar{s_1} \xrightarrow{r^*} v$ in $\bar{\mathcal{G}}$ such that $u \sim^v v$. Therefore, each property that is satisfied in u is satisfied in v as well, $\bar{\mathcal{G}}, v \models \varphi$, and v is a state along the path starting at $\bar{s_1}$. Hence, $\bar{\mathcal{G}}, \bar{s_1} \models \text{EF } \varphi$. □

6 Conclusion

We have described symmetry-reduction techniques for models that provide explicit visual semantics for dynamic multi-process systems. To generalize notions of symmetry for dynamic GTS models, we defined GTS symmetry and GTS bisimulation. Using these notions, we provided an on-the-fly algorithm for generating a symmetry-reduced GTS model based on graph isomorphism.

Determining if two graphs are permutations of each other needs graph isomorphism checking, which is a hard problem for unlabelled graphs, but it can be shown to have a polynomial complexity for deterministic labelled graphs [18]. Also, McKay [17] has developed an algorithm for graph isomorphism that works quite well in practice, handling graphs with up to millions of nodes.

We note that our work requires an upper bound on the number of nodes (components) that can be added to a state, because verification of systems for an arbitrary number of processes is generally undecidable [1]. We also have proved that the generated quotient is GTS-bisimilar to the original GTS model, and thus they both satisfy the same set of properties. To achieve better state-space savings for dynamic GTS models that are not fully symmetric, we have defined vertex bisimulation. The vertex-bisimilar GTS model provides exponential savings over the original model. Vertex bisimulation defines an equivalence relation on state graphs based on their vertices.

We showed that the vertex-bisimilar reduced model can prove an interesting subset of CTL properties satisfied by the original model. This subset includes all the properties expressed with the EF and ¬EF operators. This includes the important class of safety properties that are typically checked in an industrial verification setting. The propositional formula of these properties has been illustrated as a graph. Proposition graphs provide an abstraction on the process indices that take care of the symmetry of propositions. Currently, we are investigating the satisfaction of EF-CTL properties as well. These properties consist of all Boolean connectives and CTL's EF operator, including arbitrary nesting.

References

1. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Information Processing Letters 22, 307–309 (1986)
2. Baldan, P., Corradini, A., König, B.: Verifying finite-state graph grammars: an unfolding-based approach. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 83–98. Springer, Heidelberg (2004)
3. Baresi, L., Heckel, R.: Tutorial introduction to graph transformation: A software engineering perspective. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 402–429. Springer, Heidelberg (2002)
4. Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Symbolic counter abstraction for concurrent software. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 64–78. Springer, Heidelberg (2009)
5. Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: ICSE 2006, pp. 72–81 (2006)
6. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000)
7. Clarke, E.M., Enders, R., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. Form. Methods in Sys. Des. 9(1-2), 77–104 (1996)
8. Degano, P., Montanari, U.: A model for distributed systems based on graph rewriting. J. ACM 34(2), 411–449 (1987)
9. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Form. Methods Syst. Des. 9(1/2), 105–131 (1996)
10. Emerson, E.A., Trefler, R.J.: From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 142–157. Springer, Heidelberg (1999)
11. Heckel, R.: Compositional verification of reactive systems specified by graph transformation. In: Astesiano, E. (ed.) ETAPS 1998 and FASE 1998. LNCS, vol. 1382, p. 138. Springer, Heidelberg (1998)
12. Iosif, R.: Symmetry reduction criteria for software model checking. In: Bošnački, D., Leue, S. (eds.) SPIN 2002. LNCS, vol. 2318, pp. 22–41. Springer, Heidelberg (2002)
13. Ip, C.N., Dill, D.L.: Better verification through symmetry. Form. Methods Syst. Des. 9(1-2), 41–75 (1996)
14. Langari, Z.: Modelling and Analysis using Graph Transformation Systems. Ph.D. thesis, University of Waterloo, Waterloo, Canada (2010)
15. Langari, Z., Trefler, R.: Formal modeling of communication protocols by graph transformation. In: Misra, J., Nipkow, T., Karakostas, G. (eds.) FM 2006. LNCS, vol. 4085, pp. 348–363. Springer, Heidelberg (2006)
16. Langari, Z., Trefler, R.: Application of graph transformation in verification of dynamic systems. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 261–276. Springer, Heidelberg (2009)
17. McKay, B.: Practical graph isomorphism. Congressus Numerantium 30, 45–87 (1981)
18. Rensink, A.: Isomorphism checking in groove. ECEASST 1 (2006)
19. Rensink, A.: Explicit state model checking for graph grammars. In: Degano, P., De Nicola, R., Bevilacqua, V. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 114–132. Springer, Heidelberg (2008)
20. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1. World Scientific, Singapore (1997)
21. Trefler, R.J., Wahl, T.: Extending symmetry reduction by exploiting system architecture. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 320–334. Springer, Heidelberg (2009)