Symmetry Reduction and Compositional Verification of Timed Automata

Hoang Linh Nguyen
University of Waterloo
Waterloo, Canada
Email: nhoangli@uwaterloo.ca

Richard Trefler
University of Waterloo
Waterloo, Canada
Email: trefler@cs.uwaterloo.ca

***Abstract*—Timed automata provide a model for studying the behavior of finite-state systems as they evolve over time. We describe a technique that incorporates automatic symmetry detection and symmetry reduction in the analysis of systems modeled by timed automata. Our prototype extends the real-time model checker PAT with symmetry reduction using *state swaps* to reduce time and memory consumption. Moreover, our approach detects structural symmetries arising from process templates of real-time systems, requiring no additional input from the user. The technique involves finding all variables of type *process identifier* and computing the largest subgroup of *candidate* symmetries that induce automorphisms. Our technique is fully automatic, and not restricted to fully symmetric systems. We then combine elements of compositional proof, abstraction and local symmetry to decide whether a safety property holds for every process instance in a parameterized family of real-time process networks. Analysis is performed on a small cut-off network; that is, a small instance whose compositional proof generalizes to the entire parametric family. Our results show that verification is decidable in time polynomial in the state space of the cut-off instance. We apply these ideas to analyze Fischer's protocol and the CSMA/CD protocol.**

I. Introduction

Model Checking is an automated technique for the validation and verification of targeted hardware or software systems [1]. However, its application is limited since the state space may grow exponentially with the increase in the number of components, significantly limiting the size of systems that can be analyzed.

Many techniques have been proposed to cope with the *state space explosion*. One such technique is the exploitation of behavioral symmetries, known as *symmetry reduction* [2][3][4], since many systems consist of a set of replicated components. In those cases, we may obtain significant savings by checking a generally smaller *quotient* state space, which is constructed using knowledge of the symmetry in the original system.

Symmetry reduction has been implemented in UPPAAL [8], a model checker for networks of timed automata [7]. But UPPAAL requires a user to manually provide information on the presence of symmetry in a model, using a special datatype called *scalarset* [4]. First, this approach requires a user to have an in-depth knowledge of symmetry reduction theory and it is only applicable in the specific case, where systems are fully symmetric. Second, it compromises the automation of model checking.

In this work, we propose a prototype that extends the real-time model checker PAT [5] with symmetry reduction. Our work is based on the *state swap* technique [6] and focuses on tackling technical problems that UPPAAL faces with.

We propose a method to detect structural symmetries arising from parameterized process templates of real-time systems that requires no additional information from a user. It results in a group of *candidate* symmetries and then computes the largest possible subgroup of these candidates that induce automorphisms. Moreover, our approach could extract symmetry information from any parametric real-time systems (whether fully symmetric or not). Similar to UPPAAL, we have run experiments on Fischer's protocol [9] and CSMA/CD protocol [10]. As a result, we gain a considerable reduction in the cost of analysis, by a factorial magnitude.

State space explosion limits model checking to small protocol instances, however it becomes important to know whether a protocol is correct for an arbitrary number of components. Therefore, we propose a technique to determine whether a property holds for every instance of a parameterized family of real-time process networks. This work is directly motivated by attempts to verify Fischer's protocol with a large number of components.

Our technique incorporates elements of local symmetry reduction, compositional reasoning and abstraction. While symmetry reduction partitions network nodes into equivalence classes, compositional reasoning analyzes each representative node of an equivalence class separately along with an abstraction of its neighboring processes. In certain families of process networks that satisfy the conditions of local symmetry, then the verification is decidable and relatively effective. Our results show that verification is decidable in time polynomial in the state space of the cut-off instance for networks of Fischer's protocol and CSMA/CD protocol.

The rest of the paper is organized as follows: Next section summarizes the theory of timed automata. Section 3 describes how Fischer's protocol is modeled in PAT. Section 4 presents symmetry reduction in timed automata while Section 5 discusses the proposed symmetry detection technique. We explain about parameterized compositional model checking and how they can be used to significantly reduce the verification complexity in Section 6. Section 7 and 8 illustrate the method through 2 examples: Fischer's protocol and CSMA/CD

protocol. Section 9 summarizes and draws related works and conclusions.

II. Timed Automata

Timed automata is a formalism used for the modeling and verification of real-timed systems [11]. A timed automaton is a finite-state Buchi automaton extended with real-valued variables to model clocks. PAT uses *Timed Safety Automata* [12], which uses local clock constraints, called *location invariants* to force transition to be taken.

A clock constraint is a formula in the form of $x \sim n$ or $x-y \sim n$ where $\sim \in (=, \leq, <, >, \geq)$, $n \in \mathbb{N}$ and $x, y \in \mathbb{C}$ – a set of clocks. We use $B(\mathbb{C})$ to denote a set of clock constraints. A time automaton $\mathbb{A}$ is a tuple of $\langle L, l_0, E, I \rangle$ where:

- L is a finite set of locations.
- $l_0 \in L$ is the initial location.
- $E \subseteq L$ x $B(\mathbb{C})$ x Σ x $2^{\mathbb{C}}$ x L is the set of edges.
- $I : L \rightarrow B(\mathbb{C})$ is the set of location invariants.

A. Symbolic Semantics and Verfication

Because clocks are real-valued, the semantics of a timed automaton may have an infinite state-space and verification is generally undecidable. The notion of zone and zone-graphs has been developed to finitely partition the state-space into symbolic states [13]. By definition, a *zone* is the solution set of clock constraints. In many verification tools, including PAT, such sets are represented as *Difference Bounded Matrices* (DBMs) [14].

The symbolic semantics of a timed automaton is defined as a symbolic transition system. A symbolic state is a pair $\langle l, D \rangle$ where l is a location and D is a zone. Therefore, a symbolic state represents a set of states. The symbolic transition relation over symbolic states are defined as following:

$$\langle l, D \rangle \xrightarrow{d} \langle l, D^{\uparrow} \wedge I(l) \rangle \text{ where } D^{\uparrow} = \{v+d \mid v \in D, d \in \mathbb{R}_+\}$$

$$\text{or } \langle l, D \rangle \xrightarrow{a} \langle l', D' \wedge I(l') \rangle \text{ where } D' = \textit{reset(D)}.$$

reset(D) is an operation on zones that resets selected clock values to zero. However, zones may contain arbitrarily large constants and thus the zone-graph may be infinite [14]. The solution is to transform zones into their normalized representatives [15] to guarantee termination.

B. Symbolic Reachability Analysis

Reachability analysis is used to verify properties on timed systems by traversing the state-space. The process consists of two steps: compute the normalized zone-graph *on-the-fly*, and check if a current symbolic state contradicts or satisfies given properties. **Algorithm 1** presents the reachability analysis algorithm in PAT. **Algorithm 1** will generate the entire state-space to prove invariant properties.

III. Fischer's Protocol

Fischer's protocol ensures mutual exclusion of access to commonly used resources via a shared variable **id**. The protocol relies on location invariants and suitable updates of the variable **id**.

Algorithm 1 Forward Reachability Analysis Algorithm

```
1:  Visited = ∅
2:  ⟨l_f, D_f⟩ = Violated States
3:  Next = ⟨l_0, D_0⟩
4:  while Next ≠ ∅ do
5:      remove ⟨l, D⟩ from Next
6:      if l = l_f and D ∩ D_f ≠ ∅ then return Failed
7:      if ⟨l, D⟩ ⊈ Visited then
8:          add ⟨l, D⟩ to Visited
9:          for ⟨l', D'⟩ such that ⟨l, D⟩ → ⟨l', D'⟩ do
10:             add ⟨l', D'⟩ to Next
11: return SATISFIED
```

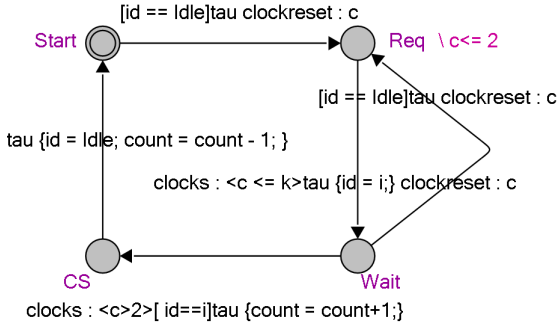

Fig. 1. The process template for Fischer's protocol

A. Model for Fischer's Protocol

We model Fischer's protocol by the real-time system module in PAT.

1) Modeling Process Behavior: Processes of the protocol are instances of a template depicted in Figure 1. The template has one local clock **c** and no local variables.

Initially, a process is in location $Start$. The default value of **id** is $Idle$ and clock **c** is set to 0. The transition from $Start$ to Req is always enabled.

In location Req, the process sets **id** to its process identifier and then goes to $Wait$ before 2 time units have elapsed.

In location $Wait$, the process waits for at least 2 time units and then reads **id** again. If **id** has kept the old value, the process may enter its critical section CS. Otherwise, the attempt has failed and the process must go back to Req and try again.

2) Modeling Fischer's Protocol: The whole protocol consists of N processes interleaving with each other. In PAT, we model this as follows:

$$Fischer = ||| \text{ i} : \{0, ..., N-1\} \ @ \ Process(i)$$

IV. Symmetry Reduction In Timed Automata

Many systems consist of identical or similar components. In those cases, we may obtain significant savings by exploiting symmetry of the underlying state space. This technique is known as *symmetry reduction* in model checking.

A. A Theory of Symmetry

This subsection summarizes the work of Ip and Dill presented in [4]. They consider state graphs, which are tuples containing a set of states S and a set $S_0 \subseteq S$ of initial states, and a transition relation $\Delta \subseteq S$ x S.

Algorithm 1 verifies given properties on an input model by traversing the state space. If the set S is finite and small, then this algorithm halts. Otherwise, it may not halt due to the state space explosion as the model become larger.

Symmetry reduction exploits structural properties of transition systems to speed up **Algorithm 1**. An automorphism [4], which is used to characterize symmetry in a state graph, is defined as a tuple (S, S_0, Δ), is a bijection $h : S \to S$ such that:

- $s \in S_0$ if and only if $h(s) \in S_0$.
- If $(s, s') \in \Delta$ if and only if $(h(s), h(s')) \in \Delta$ for all $s, s' \in S$.

For any set of graph automorphisms H, H induces a relation $\approx_H$ such that $s_1 \approx_H s_2$ if there exists an element $h \in H$ such that $h(s_1) = s_2$. We say that s_1 and s_2 are equivalent and they belong to the same equivalence class.

B. Quotient Graph

Using equivalence classes within a state graph, we can define a quotient graph. Let $G = (S, S_0, \Delta)$ be a state graph, H be a symmetry group for G and $[s]$ be an equivalence class of state s. The quotient graph of G induced by H is the graph $Quot(G) = (S', S'_0, \Delta')$, where: $S' = \{[s] \mid s \in S\}$; $S'_0 = \{[s] \mid s \in S_0\}$; and $\Delta' = \{([s], [k]) \mid (s, k) \in \Delta\}$.

Since state s is reachable in G if $[s]$ is reachable in $Quot(G)$, $Quot(G)$ is at most as large as G and in many cases when models clearly exhibit considerable symmetry, the use of $Quot(G)$ can speed up the verification process.

C. Symmetry Reduction

Symmetry reduction in model checking involves replacing a set of equivalent states in G by a single representative, *rep(s)*, from each equivalence class. Moreover, for any pair of equivalent states (s, s'), either all satisfy certain properties φ or none of them do. So we have: $s \models \varphi$ if and only if $rep(s) \models \varphi$.

Consequently, we may improve **Algorithm 1** to store and explore only a single representative $rep(s)$ of each equivalence class. **Algorithm 2** presents the modified reachability analysis algorithm in PAT, where θ is a *representative function* that converts a state s to its representative $rep(s)$.

Since many equivalent states are projected onto the same representative $rep(s)$, the number of visited states may decrease dramatically.

D. State Swap

Let $Fischer$ be an instance of Fischer's protocol with N identical processes. Processes are instantiated from the parameterized process template given in Figure 1. $Fischer$ is defined in PAT as follows:

$$Fischer = ||| \text{ i} : \{0, ..., N-1\} \ @ \ P(i)$$

Algorithm 2 Modified Forward Reachability Analysis Algorithm

```
1: Visited = ∅
2: ⟨l_f, D_f⟩ = Violated States
3: Next = θ(⟨l_0, D_0⟩)
4: while Next ≠ ∅ do
5:     remove ⟨l, D⟩ from Next
6:     if l = l_f and D ∩ D_f ≠ ∅ then return Failed
7:     if ⟨l, D⟩ ⊈ Visited then
8:         add ⟨l, D⟩ to Visited
9:         for ⟨l', D'⟩ such that ⟨l, D⟩ → ⟨l', D'⟩ do
10:            add θ(⟨l', D'⟩) to Next
11: return SATISFIED
```

A state of $Fischer$ is a tuple (L, V, D), where L is a N-component location vector, V is a set of variable valuations and D is a set of clock valuations. Based on the concept of *state swap* [6], we define permutations on the state graph, in our case, a PAT model. In a model of a concurrent system with many replicated processes, we restrict attention to automorphisms given by permutations of process identifiers. A state swap $swap_{i,j}$: $(L, V, D) \to (L', V', D')$ is defined as follows:

- Process Swap: swaps the contributions to the state of all pairs of processes $P(i)$ and $P(j)$. Swapping such a pair of symmetric processes consists of interchanging the active locations and the values of the local variables and clocks (note that this is not a problem since the processes originate from the same template).
- Data Swap: swaps array entries i and j of all dimensions that are indexed by variables of type pid. Moreover, it swaps the value i with the value j for all variables of type pid.

Example 1 Assume N = 3, now we consider the following state s of the model $Fischer$:

- L: $l_0 = Start, l_1 = Wait, l_2 = CS$.
- V: id = 3.
- D: $c_0 = 4, c_1 = 3, c_2 = 2$.

When we apply $swap_{0,2}$ into this state, it results a new state s' by interchanging l_0 with l_2 and c_0 with c_2 (Process Swap), and setting the value of id to 1 (Data Swap). s' is given as follows:

- L: $l_0 = CS, l_1 = Wait, l_2 = Start$.
- V: id = 1.
- D: $c_0 = 2, c_1 = 3, c_2 = 4$.

E. Action of $swap_{i,j}$ on a Template

A process template in PAT is given in the following syntax:

$$P(x_0, x_1, ..., x_{n-1}) = \{Body\}$$

where P is the template name, $(x_0, ..., x_{n-1})$ is an optional list of template parameters and $Body$ determines the computational logic of the process. $Body$ consists of local variables v,

guards g, update u and program statements ps over variables and channels [5].

In a model of a concurrent system with many replicated processes, although processes are obtained as instances of a same parameterized process template, they are not necessarily identical up to renaming. Let P be a process template. We define $swap_{i,j}(P)$ as a new template, where a guard g, an update u and a program statement ps of P is replaced by $swap_{i,j}(g)$, $swap_{i,j}(u)$ and $swap_{i,j}(ps)$ respectively. Therefore, $swap_{i,j}(P)$ is the same as P, except that:

- Any assignment statement $x \sim val$ is replaced by $x = swap_{i,j}(val)$, where $\sim \in (=, \neq)$, $type(x) = pid$ and val is a value.
- Any boolean expression $x \sim val$ is replaced by $x \sim swap_{i,j}(val)$, where $\sim \in (=, \neq)$, $type(x) = pid$ and val is a value.

In other words, $swap_{i,j}(P)$ is a syntactic operation on the template P. If $swap_{i,j}(P)$ and P are identical, we say that $P(i)$ and $P(j)$ are identical up to renaming.

Example 2: Assume that $Fischer$ does not allow $P(2)$ to enter its critical section. This is achieved by modifying the template P given in Figure 1. A guard g on an edge from location *Wait* to location *CS* is now given as $g : id == i$ and $i \neq 2$. By applying $swap_{0,2}$ into P, it results in $swap_{0,2}(g) : id == i$ and $i \neq 0$. It is clear that $swap_{0,2}(P)$ is not identical to P since $P(2)$ is enabled to enter its critical section in the template $swap_{0,2}(P)$.

F. Extraction of Automorphisms

Let P be a parameterized process template and there are N processes $P(i)$ instantiated from P, where $0 \leq i < N$. We define $Swap(P)$ as a group of *candidate* symmetries, which consists of all permutations of the set of process identifiers $\{0, 1, ..., N-1\}$. Therefore,

$$Swap(P) = \{swap_{i,j} \mid i < j \leq N\}$$

We say that $swap_{i,j} \in Swap(P)$ is **valid** if $swap_{i,j}(P)$ and P are identical templates. In other words, they have identical behavior and they are identical up to underlying structures. Let $ValidSwap(P)$ be the largest subgroup of $Swap(P)$:

$$ValidSwap(P) = \{swap_{i,j} \mid swap_{i,j}(P) \equiv P\}$$

Theorem 1 (Soundness) [6]. Every valid state swap is an automorphism.

G. Representative Function

The technical challenge is to find an appropriate representative function θ in **Algorithm 2**. By defining the representative state $rep(s)$ as the minimal element in the equivalence class of the state s, our approach consists of sorting the symbolic states in lexicographical order using knowledge of a group of automorphisms H. A state is defined as a tuple (L, V, D). (L, V, D) is smaller than (L', V', D') if and only if:

$$(L, V, D) < (L', V', D') \Longleftrightarrow (L < L') \vee (L = L' \wedge V < V')$$

$rep(s)$ is computed by applying the quick-sort algorithm to both locations and variable valuations. Since the objective of this work focuses on the automatic symmetry detection, the approach does not considering sorting the zones.

V. The Symmetry Detection Algorithm

In this section, we introduce a technique to automatically detect structural symmetries arising from process templates. The approach operates in three stages. Note that variables of type pid play a key role in describing system symmetries and symmetry reduction. Therefore, the first stage is to recognize all variables of type pid in a template P. In the second stage, we compute a group of *candidate* symmetries $Swap(P)$. Then each element $\alpha \in Swap(P)$ is checked for validity and finally obtain a largest subgroup of these symmetries $ValidSwap(P)$ that induce automorphisms.

A. Detection of pid Variables

We explain in detail here how to automatically detect all variables of type pid in a given template P without explicitly defining a special datatype pid. P takes a process identifier i as a process parameter. We introduce *pid rules*, which only allow variables of type pid to be used in certain ways as following:

- (1) The process identifier i has type pid by default.
- (2) Given a variable x, $type(x) = pid$ if and only if it is assigned to or compared for equality with another variable val of type pid, such that $x \sim val$ where $type(val) = pid$ and $\sim \in [=, \neq, ==]$.
- (3) It is not allowed to perform any arithmetical operations on variables of type pid.
- (4) Variable x of type pid is only allowed to used in the form of $x \sim val$ where val is either a variable or value and $\sim \in [=, \neq, ==]$.
- (5) $A[N]$ is an array of N elements of type pid if and only if $type(A[i]) = pid$.

These restrictions are similar to those applied to variables of type $scalarset$ [4]. **Algorithm 3** presents the pid-type inference algorithm in PAT. **Algorithm 3** runs on a particular template, say P.

In PAT, the template is stored as the syntax tree. Each while-loop iteration involves one pass over the syntax tree. Inside each loop, the tool extracts a variable x from $Next$, checks whether it is used appropriately and then adds x to $ValidPids$ - the set of valid variables of type pid. The algorithm also adds any new variable y, which is related to x by (2), to $Next$. The algorithm only halts when $Next$ is empty.

For any template P whose variables of type pid are used inappropriately, P is **invalid**. Our technique is not restricted to only apply to fully symmetric systems (consist of a single template). We can extract symmetry information from any real-time systems that compose of multiple templates. If a model consist of M templates, **Algorithm 3** runs M times.

B. Automatic Symmetry Detection

For each valid template P computed in **Algorithm 3**, we compute a set of structural symmetries $Swap(P)$. Finally

Algorithm 3 Pid-type Inference Algorithm

```
ValidPids = ∅
Visited = ∅
Next = {i}
isInvalidTemplate = false
while Next ≠ ∅ do
    remove x from Next
    for each guard / statement / update in P do
        if (y = x or y ≠ x or y == x) then
            if (y not ∈ Visited) and (y not ∈ Next) then
                add y to Next
        if x violates (4) then
            isInvalidTemplate = true
            break
    add x to ValidPids
    add x to Visited
return YES
```

each generated structural symmetry $\alpha \in Swap(P)$ is checked for validity. Using **Algorithm 4**, the largest valid subgroup of $Swap(P)$, denoted as $ValidSwap(P)$ is computed. The group $ValidSwap(P)$ indicates the permutations of processes, arrays and shared variables which preserve the structure of the template P.

If a model Sys consists of 3 valid templates A, B, and C. Then a group of automorphisms $H(Sys)$ is given as follows:

$$H = ValidSwap(A) \cup ValidSwap(B) \cup ValidSwap(C)$$

Algorithm 4 Compute $ValidSwap(P)$

```
S = Swap(P)
H = ∅
while S ≠ ∅ do
    remove α from S
    if isEquivalent(α(P), P) then
        H = H ∪ α
```

VI. Parameterized Compositional Model Checking

In general, state space explosion limits model checking to small protocol instances. Therefore, it becomes important to know whether a protocol is correct for an arbitrary number of components. This is known as *the parameterized model checking problem* (PMCP). The problem is, however, generally undecidable [16].

We propose a technique that extends the previous work [17] to parameterized real-time protocols, asking whether a parameterized family has a *compositional proof* that the specification is met for all instances. This work is directly motivated by attempts to verify Fischer's protocol with a large number of components.

A. Preliminaries

In this paper, we extends some of key concepts that have been described in [17][18] to timed automata.

1) Internal State: An *internal state* of a real-time process P is its symbolic state, defined as a pair (l, D) where l is the location and D is the zone.

2) Neighborhood: The neighborhood of a real-time process P is the set of variables which are shared between P and other processes.

3) Local State: An asynchronous, interleaved composition of processes $P_1, P_2, ..., P_N$ is written as $P = P_1||P_2||...||P_N$. Local state of a node P_i is written in the form (l_i, D_i, y), where l_i is a current location; D_i is a current zone and y is a vector of its neighborhood valuations.

4) Inductive Invariant: An *invariant* is a *predicate* which holds of all reachable symbolic states. And an *inductive invariant* is a predicate that includes all initial states and is closed under the transition relation.

5) Compositional Invariants: Define θ_i as a set of local states of P_i. θ_i is called a compositional invariant if: (init) θ_i includes the initial states of P_i; (step) θ_i is inductive for P_i; (non-interference) the actions of a neighboring process, P_j, do not falsify θ_i.

Theorem 1: If the set $\{\theta_i\}$ is a compositional invariant, then $(\forall i : \theta_i)$ is a global inductive invariant of the program $(||i : P_i)$.

6) Compositional Cutoff: Define $\theta_{>K}$ be a compositional invariant in a network of size greater than K and $\theta_{\leq K}$ be a compositional invariant in a network of size at most K. If $\theta_{>K}$ is identical (up to neighborhood isomorphism) to $\theta_{\leq K}$, then we refer to K as a *compositional cutoff*.

B. Local Symmetry

Two nodes m and n are locally similar, written $m \simeq n$, if there is a bijective function β that the neighborhood of m is isomorphic to the neighborhood of n through β [17]. A tuple (m, β, n) is called *local symmetry*.

For a local symmetry (m, β, n), β maps the neighborhood of m onto the neighborhood of n. Moreover, for every (m, β, n) that respects the local symmetries, it should hold that β maps a local state (x, y) of m to a local state $(x, \beta(y))$ of n. If two nodes m and n have isomorphic invariants, we say that they are balanced or they respect a balance relation B.

Corollary 1: Let θ be the strongest compositional invariant and φ be a property of local states. If (m, β, n) is a local symmetry, and φ is invariant under β, then $[\theta_m \Longrightarrow P]$ if and only if $[\theta_n \Longrightarrow P]$.

Because we focus on parameterized real-time protocols that exhibit clearly global symmetries, so it becomes important to know whether global symmetry induces local symmetry and a balance relation.

Theorem 2: For a network with global symmetry group G, the set $Local(G) = \{(m, \beta, n) \mid \beta \in G \wedge \beta(m) = n\}$ is a balance relation [18].

Corollary 2: A network where any pair of nodes is connected by a global automorphism is called *vertex-transitive*. In a vertex-transitive network, any pair of nodes is balanced and there is a single equivalence class [17].

In the next two sections, we show that the PCMCP is decidable in polynomial time for Fischer's protocol and CSMA/CD protocol.

VII. Verification for Fischer's Protocol

A. Verification Properties

MutualExclusionFail is a boolean condition $true$ of global states, where more than one process are in the local state CS at the same time. In PAT, *MutualExclusionFail* is defined as follows: $define\ MutualExclusionFail\ count > 1$;

In order to formally verify our model of Fischer's protocol is correct, we check whether the model reaches to *MutualExclusionFail*: $assert\ Fischer\ reaches\ MutualExclusionFail$

B. Symmetry Reduction on Fischer's Protocol

Our symmetry detection algorithm first detects that the variable **id** has type pid. Moreover, $Fischer$ is fully symmetric since $Process(1), Process(2), ..., Process(N)$ are identical up to renaming. Therefore, we have:

$$ValidSwap(Fischer) = \{swap_{i,j}(Fischer) \mid 0 \leq i < j < N\}$$

From Theorem 2, $Fischer$ consists of N balanced processes.

C. Experimental Results

We have run experiments on PAT for different numbers of processes and Table 1 summarizes the verification results. The environment is an i5-dual-core machine with 4 GB memory.

Processes	8		10		15		20		30	
Mode	Sym	No	Sym	No	Sym	No	Sym	No	Sym	No
Time (s)	0.04	3.3	0.13	122	0.85	N/A	4.35	N/A	53	N/A
Visited State	190	85495	360	1827331	1379	N/A	3880	N/A	17717	N/A
Memory (Mb)	9.3	57.7	11	1347.5	11.8	N/A	22	N/A	109	N/A

TABLE I
Verification Results for Fischer's Protocol

To demonstrate the effectiveness of symmetry reduction, we ran each experiment twice, with and without symmetry reduction. Experiments were run with a 300 second timeout. We focus on three criteria: processing time (s), the number of visited states and memory usage (MB). The data shows that the regular PAT's limit for Fischer's protocol is less than 15 processes while the verification for 30 processes can be done within 53 seconds using less than 109MB of memory with symmetry reduction.

For Fischer's protocol, the verification tool gains a considerable reduction in processing time and memory usage, by a factorial magnitude, however model checkers are still impractical in verify an instance of Fischer's protocol with a very large number of processes (> 40).

D. Arbitrary Large Number of Processes

Consider a family of instances of Fischer's protocol $\{R_i\}$, where each instance consists of i identical processes. We combine elements of compositional proofs, abstraction and local symmetry to verify whether mutual exclusion holds for every instance of Fischer's protocol.

Since Fischer's protocol is fully symmetric, every instance R_i is *vertex-transitive*. In a instance R_x, we define Rep as a single representative process and θ_x as a compositional invariant for Rep. A state in θ_x is a tuple $(l, D, \mathbf{id})$, where (l, D) is an internal state of Rep and **id** is a $neighborhood$ ranging from $\{0, ..., x-1\}$. Since x could have any possible positive value, θ_x may unbounded. We define an abstraction of Rep, denoted $\widehat{Rep}$ and show that its compositional invariant is sufficiently precise to solve the PCMCP.

In the abstract representative process $\widehat{Rep}$, the transitions are the same as in Rep, except that **id** has a value in the set of $\{k, \hat{k}, -1\}$, where k is the process identifier of Rep and $\hat{k}$ is the abstract process identifier of any process in R_x that its process identifier is not equal to k. The abstraction is a Galois connection (α, γ) where $\alpha(s, \mathbf{id}) = (s, a)$ where a is the set of three possible values $\{k, \hat{k}, -1\}$ and $\gamma(s, a) = \{(s, \mathbf{id}) \mid \alpha(s, \mathbf{id}) = (a, s)\}$.

We define Δ_x is the strongest compositional invariant on the abstract process $\widehat{Rep}$, the first lemma says that the compositional invariant of the abstract process over-approximates the concrete one.

Lemma 1 For each state in θ_x, there is an α-related state in Δ_x.

Proof: We define D_i as a finite set of zones at the location i. In location $Start$, Δ_x^{Start} is just the state $(Start, D_{Start}, -1)$ with the $location = Start$, and **id** $= -1$. So $\Delta_x^{Start} = \theta_x^{Start}$. Hence, the hypothesis holds for Δ_x^{Start}.

In Req, $\theta_x^{Req} = \{(Req, D_{Req}, i) \mid i \in \{-1, .., x-1\}/k\}$, where x is the size of the instance R_x. Δ_x^{Req} has one of two possible values including $(Req, D_{Req}, -1), (Req, D_{Req}, \hat{k})$. It is clear that each state of Req in the set of $\{(Req, D_{Req}, i) \mid i \in \{0, ..., x-1\}/k\}$ is related to $Req, D_{Req}, \hat{k})$ by α. So it satisfies the condition required.

In location $Wait$, there are two possible cases. If **id** $= k$, $(Wait, D_{Wait}, k) \in \theta_x^{Wait}$ and also $(Wait, D_{Wait}, k) \in \Delta_x^{Wait}$. Otherwise, $\{(Rep, D_{Wait}, i) \mid i \in \{-1, ..., x-1\}/k\} \in \theta_x^{CS}$, which are related to $(Rep, D_{Wait}, \hat{k}) \in \Delta_x^{Wait}$ by α. Hence, the hypothesis holds for Δ_x^{Wait}.

In location CS, $\theta_x^{CS} = \Delta_x^{CS} = (CS, D_{CS}, k)$. **End Proof**

Theorem 3 The PCMCP is decidable in polynomial time for Fischer's protocol.

Proof Consider an instance R_x and an instance R_y. Assume R_x is the smallest verified instance ($x = 2$). From Lemma 1, all states in Δ_x also satisfy the mutual exclusion property. Δ_x and Δ_y are isomorphic since they have the same local states. We can say Δ_x and Δ_y are locally symmetric and hence, from Corollary 1, all states in Δ_y also ensure mutual exclusion. Since for each state in θ_y, there is an α-related state in Δ_y. It is clear that θ_y satisfies mutual exclusion as expected. So the PCMCP is decidable in polynomial time for Fischer's protocol. **End Proof**.

VIII. CSMA / CD Protocol

The Carrier Sense, Multiple Access with Collision Detection (CSMA/CD) protocol describes one solution to the problem in Ethernet network, when several agents compete for a

$$
\begin{aligned}
WaitFor(i) &= (cd?i \rightarrow WaitFor(i)) \\
&\quad \Box\ (newMess!i \rightarrow ((begin!i \rightarrow Trans(i)) \\
&\qquad \Box\ (busy?i \rightarrow Retry(i)) \\
&\qquad \Box\ (cd?i \rightarrow Retry(i)))); \\
Trans(i) &= (cd?i \rightarrow Retry(i) within [0, 52]) \\
&\quad \Box\ (atomic\{end!i \rightarrow Skip\} within [808, 808]; \\
&\quad\ WaitFor(i)); \\
Retry(i) &= (newMess!i \rightarrow ((begin!i \rightarrow Trans(i) within [0, 52]) \\
&\quad \Box\ (busy?i \rightarrow Retry(i) within [0, 52]) \\
&\quad \Box\ (cd?i \rightarrow Retry(i) within [0, 52])));
\end{aligned}
$$

Fig. 2. Model for a Sender i [19]

single bus. The research group in PAT has successfully done modeling and verification on CSMA/CD protocol [19]. In this section, we extend their work to verify the protocol with our symmetry reduction technique.

A. Model for CSMA/CD Protocol

CSMA/CD protocol consists of two components, namely $Sender$ and Bus. Two components communicate by pair-wise synchronization channels.

Roughly speaking, a $Sender$ must first listen to the Bus before transmitting messages. If the Bus is idle, the $Sender$ begins to transmit. Otherwise, it must wait and retry later. However, collision may occur when more than one $Sender$ are sending message via the Bus. Then the Bus informs all *Senders* of this collision, and abort their transmission immediately. Therefore, all transmitting messages are discarded and *Senders* should compete for the Bus again.

1) Model Sender Behavior: The behavior of component $Sender$ is showed in Figure 1. Initially, a $Sender$ is in location $WaitFor$. When there is a message to send, if the Bus is idle, the $Sender$ goes to location $Trans$. Otherwise, if the Bus is busy or a collision is detected, it moves to location $Retry$. If a collision occurs while no message is arrived, the $Sender$ remains in location $WaitFor$.

In location $Trans$, the $Sender$ has two transitions. If a collision is detected before 52 time units have elapsed, the $Sender$ goes to location $Retry$. Otherwise, it terminates sending the message after exactly 808 time units, then it goes to location $WaitFor$.

In location $Retry$, if the Bus is idle, the $Sender$ moves back to location $Trans$ within 52 time units. Otherwise, it remains in location $Retry$.

2) Modeling Bus Behavior: The behavior of component Bus is showed in Figure 2. Initially, a Bus is in location $Idle$. The transition from $Idle$ to $Active$ is enabled when one $Sender$ begins to transmit.

In location $Active$, there are three possible transitions. If the $Sender$ completes sending, the Bus goes back to the initial location. If another $Sender$ starts sending messages within 26 time units, the Bus moves to location $Collision$. Otherwise,

$$
\begin{aligned}
Idle &= newMess?i \rightarrow begin?i \rightarrow Active; \\
Active &= (end?i \rightarrow Idle) \\
&\quad \Box\ (newMess?i \rightarrow \\
&\qquad ((begin?i \rightarrow Collision)\ timeout[26] \\
&\qquad \Box\ (busy!i \rightarrow Active1))); \\
Active1 &= (end?i \rightarrow Idle) \\
&\quad \Box\ (newMess?i \rightarrow busy!i \rightarrow Active1); \\
Collision &= atomic\{BroadcastCD(0)\} within [0, 26];\ Idle;
\end{aligned}
$$

Fig. 3. Model for the Bus [19].

after at least 26 time units have elapsed, the Bus replies busy signal to any new attempt, then it moves to location $Active1$.

In location $Active1$, the Bus takes at most 26 time units to inform all $Sender$ of this collision, using *BroadcastCD* [19]. After that, the Bus moves to location $Idle$.

In location $Active1$, the Bus replies busy signal to any $Sender$ that attempts to send message until the active $Sender$ completes transmitting, then the Bus moves to location $Idle$.

3) Modeling CSMA/CD Protocol: The whole protocol consists of 1 Bus and N $Senders$ interleaving with each other. In PAT, we model this as follows:

$$CSMA = (|||\ \mathrm{i} : \{0, ..., N-1\}\ @\ Sender(i))\ |||\ Bus$$

B. Verification Properties

deadlockfree is a safety property, pre-defined in PAT, so that the model is always possible to move from one state to another. In order to formally verify our model of CSMA/CD protocol is correct, we check whether the model satisfies *deadlockfree*.

$$assert\ CSMA\ deadlockfree;$$

C. Symmetry Reduction on CSMA/CD Protocol

We denote $CSMA$ as the instance of CSMA/CD protocol with 1 Bus and N $Senders$. Our symmetry detection technique figures out that $Sender(1), Sender(2), ..., Sender(N)$ are identical up to renaming. Then we have:

$$ValidSwap(CSMA) = \{swap_{i,j}(Sender) \mid 0 \leq i < j < N\}$$

where $swap_{i,j}(Sender)$ applies on $Sender(i)$ and $Sender(j)$ and leaves the Bus intact. From Theorem 2, $CSMA$ consists of N identical and balanced $Senders$.

D. Experimental Result

Similar to Fischer's protocol, we have run experiments on PAT for different numbers of processes and Table 2 summarizes the verification results for CSMA/CD protocol. Experiments were run with a 300 second timeout. The data shows that the regular PAT's limit for CSMA/CD protocol is around 10 processes, while verification for 30 processes can be done less than 3 second using less than 23MB of memory with symmetry reduction.

Similar to Fischer's protocol, PAT is still impractical to verify an instance of CSMA/CD protocol with a very large number of *Senders*.

Processes	8		10		15		25		30	
Mode	Sym	No	Sym	No	Sym	No	Sym	No	Sym	No
Time (s)	0.037	5.6	0.07	99.8	0.26	N/A	1.45	N/A	2.76	N/A
Visited State	99	30953	149	291869	299	N/A	775	N/A	1067	N/A
Memory (Mb)	10.1	29.8	10.6	256	10.5	N/A	16.5	N/A	23.3	N/A

TABLE II
VERIFICATION RESULTS FOR CSMA/CD PROTOCOL

E. Arbitrary Large Number of Senders

Consider a family of instances of CSMA/CD protocol $\{R_i\}$, where each instance consists of one Bus and i identical $Senders$. Similarly, we verify whether $deadlockfree$ holds for every instance of CSMA/CD protocol.

In the simplest compositional formulation, we define two invariants: θ_{B_i}, which represents local states of the Bus in R_i and θ_{S_i}, which represents local states of the single representative $Sender$ in R_i (since all $Senders$ are identical up to renaming).

Theorem 4 The PCMCP is decidable in polynomial time for CSMA/CD protocol.

Proof: A state in θ_S is a tuple $(l_S, D_S, cd, begin, end, busy)$, where (l_S, D_S) is an internal state of the $Sender$ and $(cd, begin, end, busy)$ is a vector of values for its neighborhoods.

A state in θ_B is a tuple $(l_B, D_B, cd, begin, end, busy)$, where (l_B, D_B) is an internal state of the Bus, and $(cd, begin, end, busy)$ is also a vector of values for its neighborhoods.

Clearly, the neighborhoods of the Bus and the $Sender$ are a group of synchronization channels and they are valueless. Therefore, the Bus and the $Sender$ have finite local states.

Consider an instance R_x and an instance R_y. Assume R_x is the smallest verified instance $(x = 2)$. Then we have that θ_{B_x} is isomorphic to θ_{B_y} and θ_{S_x} is isomorphic to θ_{S_y}.

The representative system consists of the smallest instance with 1 Bus and 2 $Senders$. Its strongest compositional invariant can be calculated automatically. So the PCMCP is decidable in polynomial time for CSMA/CD protocol. **End Proof**.

IX. CONCLUSION

There are two model checkers for timed systems that exploit symmetry: UPPAAL [7] and RED [20]. While RED only points out whether a given state is unreachable, UPPAAL requires a user to manually specify symmetry to be exploited via $scalarset$. Moreover, UPPAAL is only realistic in cases that there is full symmetry.

Several works have been done to support automatic symmetry detection. In [21], the authors introduce a new specification language, *Promela-Lite*, to the model checker SPIN [22]. Then they show how they can detect symmetry from specifications defined in *Promela-Lite*. However, they still need to explicitly define a special datatype named *process identifier* (pid) in a language to facilitate the process.

We have successfully implemented an automatic symmetry reduction package to PAT. We demonstrate its effectiveness, showing that the method requires no additional information from an user and is not restricted to fully symmetric systems. For Fischer's protocol and CSMA/CD protocol, the model checker gains a considerable reduction in time and memory consumption, by a factorial magnitude.

We also show the potential of incorporating local symmetry reduction, compositional proof and abstraction to verify parameterized real-time systems. Our results are very promising: the PCMCP is in polynomial time for many real protocols such as Fischer's protocol and CSMA/CD protocol.

As this is a new formulation of parameterized verification, there are still many works to do further. Future work includes exploring another popular family networks, such as Train-Bridge protocol. We have successfully modeled Train-Bridge protocol in PAT and verified with symmetry reduction. Moreover, we possibly extend the verification procedures to liveness properties. And to benefits more real protocols, it requires some implementation to allow the use of process identifiers in arithmetic operations.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. Peled, "Model checking. 2000," 2000.

[2] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha, "Exploiting symmetry in temporal logic model checking," *Formal methods in system design*, vol. 9, no. 1, pp. 77–104, 1996.

[3] E. Emerson and A. Sistla, "Symmetry and model checking," in *Computer Aided Verification*. Springer, 1993, pp. 463–478.

[4] C. N. Ip and D. L. Dill, "Better verification through symmetry," *Formal methods in system design*, vol. 9, no. 1-2, pp. 41–75, 1996.

[5] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," ser. Lecture Notes in Computer Science, vol. 5643. Springer, 2009, pp. 709–714.

[6] M. Hendriks, *Enhancing uppaal by exploiting symmetry*. Nijmegen Institute for Computing and Information Sciences, Faculty of Science, University of Nijmegen, 2002.

[7] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[8] M. Hendriks, G. Behrmann, K. Larsen, P. Niebert, and F. Vaandrager, "Adding symmetry reduction to uppaal," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2003, pp. 46–59.

[9] M. Abadi and L. Lamport, "An old-fashioned recipe for real time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1543–1571, 1994.

[10] S. Yovine, "Kronos: A verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 123–133, 1997.

[11] R. Alur and D. Dill, "Automata for modeling real-time systems," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1990, pp. 322–335.

[12] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," in *Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on*. IEEE, 1992, pp. 394–406.

[13] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *International Conference on Computer Aided Verification*. Springer, 1989, pp. 197–212.

[14] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on concurrency and petri nets*. Springer, 2004, pp. 87–124.

[15] P. Pettersson, *Modelling and verification of real-time systems using timed automata: theory and practice*. Department of Computer systems, Univ., 1999.

[16] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Information Processing Letters*, vol. 22, no. 6, pp. 307–309, 1986.

[17] K. S. Namjoshi and R. J. Trefler, "Parameterized compositional model checking," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 589–606.
[18] ——, "Local symmetry and compositional verification," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2012, pp. 348–362.
[19] L. Shi and Y. Liu, "Modeling and verification of transmission protocols: A case study on csma/cd protocol," in *Secure Software Integration and Reliability Improvement Companion (SSIRI-C), 2010 Fourth International Conference on*. IEEE, 2010, pp. 143–149.
[20] F. Wang and K. Schmidt, "Symmetric symbolic safety-analysis of concurrent software with pointer data structures," in *International Conference on Formal Techniques for Networked and Distributed Systems*. Springer, 2002, pp. 50–64.
[21] A. F. Donaldson and A. Miller, "Automatic symmetry detection for model checking using computational group theory," in *International Symposium on Formal Methods*. Springer, 2005, pp. 481–496.
[22] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.