

# Behavioral Automata Composition for Automatic Topology Independent Verification of Parameterized Systems

Youssef Hanna      Samik Basu      Hridesh Rajan  
Computer Science, Iowa State University  
226 Atanasoff Hall, Ames, IA, USA  
{ywhanna,sbasu,hridesh}@cs.iastate.edu

## ABSTRACT

Verifying correctness properties of parameterized systems is a long-standing problem. The challenge lies in the lack of guarantee that the property is satisfied for all instances of the parameterized system. Existing work on addressing this challenge aims to reduce this problem to checking the properties on smaller systems with a bound on the parameter referred to as the *cut-off*. A property satisfied on the system with the cut-off ensures that it is satisfied for systems with any larger parameter. The major problem with these techniques is that they only work for certain classes of systems with a specific communication topology such as ring topology, thus leaving other interesting classes of systems unverified. We contribute an automated technique for finding the cut-off of the parameterized system that works for systems defined with any topology. Given the specification and the topology of the system, our technique is able to automatically generate the cut-off specific to this system. We prove the soundness of our technique and demonstrate its effectiveness and practicality by applying it to several canonical examples where in some cases, our technique obtains smaller cut-off values than those presented in the existing literature.

## Categories and Subject Descriptors

D.2.4 [Software/Program Verification]: Formal Methods; D.2.4 [Software/Program Verification]: Model Checking

## General Terms

Verification

## Keywords

parameterized model checking

## 1. INTRODUCTION

Parameterized systems are systems consisting of homogeneous processes, where the parameter indicates the number of such processes in the system. A parameterized system, therefore, describes an infinite family of systems where instances of the family can be

obtained by fixing the parameter. Verification of correctness of such systems amounts to verifying the correctness of every member of the infinite family described by the system. For example, for distributed mutual exclusion protocols [31], the objective is to verify that the critical section is accessed in a mutually exclusive fashion regardless of the number of processes participating in the protocol.

Given a parameterized system  $sys(n)$  containing  $n$  processes and a safety or liveness  $LTL \setminus X$  property  $\varphi$ , verification of whether  $sys(n)$  satisfies  $\varphi$  (denoted by  $\forall n : sys(n) \models \varphi$ ) is undecidable in general [4]. A number of sound but incomplete verification techniques has been proposed and developed in the recent past, e.g. those that rely on abstraction [13, 15, 23, 29] and/or smart representation [1, 2, 6, 7, 10–12, 22, 26, 30] of the system behavior and the property. In essence, these techniques depend on computing the *invariant* or the common global behavior of  $sys(n)$  for all  $n$  and identifying the smallest  $k < n$  such that  $sys(k)$  exhibits that behavior. It can be shown that  $sys(k) \models \varphi \Leftrightarrow \forall n \geq k : sys(n) \models \varphi$ , i.e., verification of an infinite family of systems is reduced to verification of a single instance (the  $k$ -th instance) of the family; where  $k$  is referred to as the *cut-off*.

**Our solution.** We propose a new technique for identifying such a cut-off  $k$  for a parameterized system. Unlike most existing work, our technique is independent of the communication topology between the processes in the parameterized system. Furthermore, our technique does not depend on actual properties to be verified for the parameterized system; the results of our technique are applicable to any properties of the form  $\varphi(i)$  (involving any one process) and  $\varphi(i, j)$  (involving any two processes dependent on each other). Our technique is automatic and uses standard automata-representation of the protocol behavior. There are two steps in our technique. First, using the fact that  $sys(n) = P \parallel P \parallel \dots \parallel P$  is the parallel composition of  $n$  processes each with behavioral specification  $P$ , we introduce the notion of  $1E$ -behavior capturing the behavior of any *one* process in *any* environment. In the second step, we enumerate the behavior of  $sys(m)$  for  $m = 2, 3, \dots, n$ , and identify the smallest  $sys(k)$  whose projected behavior on any one of the participating processes simulates the  $1E$ -behavior. We prove that for all properties involving one or two processes,  $sys(k)$  satisfies the properties if and only if  $\forall n \geq k : sys(n)$  satisfies the same properties, i.e.,  $k$  is the cut-off for the parameterized system. Note that such a  $k$  may not exist in general, which will result in non-termination of our technique rendering it incomplete as expected.

**Contributions.** In summary, contributions of this work are:

1. We present an automated technique for verification of parameterized system which is independent of the communication topology of the processes in the system. We prove the soundness of our technique, i.e., if our method terminates then it terminates with the smallest cut-off  $k$ .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ESEC-FSE'09, August 23–28, 2009, Amsterdam, The Netherlands.  
Copyright 2009 ACM 978-1-60558-001-2/09/08 ...\$5.00.

2. Our technique is system specific and as such the computed cut-off is also system specific. This allows us to obtain different bounds to different types of parameterized systems even when the underlying communication topology of the systems under consideration are identical. Emerson and Namjoshi [19] proved that for parameterized systems with ring topology where processes communicate through a token, the cut-off  $k$  is 4 for properties of the form  $\varphi(i, j)$ . We show that tighter bounds can be obtained if the behavior of the participating processes in the parameterized system is considered. For example, using our technique, simple parameterized token ring protocol has the cut-off  $k = 2$ , while dining philosopher problem has the cut-off  $k = 3$  for properties  $\varphi(i, j)$  where  $i$  and  $j$  are dependent on each other.
3. We present a number of case studies of parameterized systems with different communication topologies and show the practical applicability of our technique.

**Organization.** The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our technique for specifying a protocol using the Distributed Mutual Exclusion protocol as an illustrative example. Section 4 describes how the maximum behavior of a process in the context of any environment is generated. Section 5 shows the procedure for generating the cut-off size. Proof of soundness of our technique is presented in Section 6. Section 7 describes our case studies and Section 8 concludes.

## 2. RELATED WORK

Techniques for verifying parameterized systems can be categorized as follows. One class of solutions [13, 15, 23, 29] relies on reducing the problem of parameterized system verification with infinite states to verification of its finite-state abstractions. Another class of techniques analyze the behavior of parameterized systems using smart representation and verification mechanisms such as regular languages [1, 2, 10, 22], petri-nets and graph-grammars [6, 7, 11, 12, 26, 30]. Others [5, 28, 31] apply automatic induction to generate and verify invariants of the parameterized systems. Closest to our technique is the class of solutions that focus on computing a cut-off of the system parameter [8, 17–21, 25].

Abstraction techniques include counter abstraction [23, 29] and environmental abstraction [13, 15]. The idea behind counter abstraction is to abstract process identities, where every abstract state contains an abstract counter denoting the number of processes in the state. Environmental abstraction follows a similar approach; however, the counting is done for the number of processes satisfying a given predicate. Typically these techniques either require human guidance for obtaining the appropriate abstraction or are applicable to a certain restricted class of systems and/or properties (e.g., universal path properties).

Among the techniques that rely on smart representation mechanism are techniques based on regular language [1, 2, 10, 22] or graph-based [6, 7, 26, 30] representations of the state-space of the parameterized system. These approaches are typically applicable for the verification of safety/reachability properties of parameterized systems. Recently, petri net based representation has been proposed [12] where tokens in the petri net are used to denote the parameter of the system and a new logic, colored markings logic (CML), is developed to reason about such petri nets. The work provides a generic framework for representing parameterized systems and identifies a fragment of CML for which the satisfiability problem is decidable.

Techniques based on induction use network invariants to reduce the problem of parameterized system verification to a finite state

model checking problem [14, 31]. The idea behind these techniques is to find a network invariant  $I$  where the invariant is preserved by all computation steps of the system. Therefore, if  $I$  satisfies the desired property specification  $\varphi$  then the parameterized system also satisfies  $\varphi$ . While in most settings the invariant generation requires manual guidance, Pnueli *et al.* [28] present a technique where invariants are computed automatically once the appropriate abstraction relation is provided.

Another interesting approach, which aims to reduce the parameterized system verification problem to an equivalent finite-state one, is based on finding an appropriate cut-off  $k$  of the parameter of the system. The objective is to establish that a property is satisfied by the system with  $k$  processes if and only if it is satisfied by any number ( $\geq k$ ) processes. Emerson and Namjoshi [19] provide such cut-off values for different types of properties of parameterized systems with ring topology.

In contrast to the above techniques, our approach is fully automatic, does not depend on a specific representation mechanism of the system and/or property and is independent of the communication topology of the processes in the system. We present an algorithm (sound but incomplete) which takes as input the description of the parameterized system in terms of standard input/output automata and establishes the cut-off of the parameter value. While Emerson and Namjoshi [19] establish for the first time the cut-off bound for any parameterized system with ring topology given a specific type of property; we show that by considering the parameterized system being verified in the computation of the cut-off, a tighter bound of the cut-off can be obtained. For instance, using Emerson and Namjoshi’s approach [19], to verify the property that in a parameterized system with ring topology two processes cannot enter the critical section at the same, the cut-off size is 4; while the cut-off value identified using our approach for a specific parameterized system with ring topology (token ring protocol) is only 2. In short, while Emerson and Namjoshi [19] focus on obtaining a generic cut-off for parameterized systems with a specific topology, the central theme of our technique is to develop a generic approach that can be applied to parameterized systems independent of the communication topology.

## 3. PARAMETERIZED SYSTEM

A parameterized system can be described by the collective behavior of  $n$  *homogeneous* processes interacting with each other, where  $n$  is the system parameter. The key idea behind our approach is to provide a mechanism for specifying the behavior of a process in the parameterized system as a collection of *atomic steps*, which we call *behavioral automaton*. An important property of our specification technique is that it enables automatic composition of these behavioral automata to obtain the full-behavior of a process in an arbitrary environment.

The direct benefit of this property is that it helps us reduce the problem of finding the cut-off value  $k$  for the system parameter to an equivalence detection problem between the full-behavior of a process in an arbitrary environment and the parameterized system of size  $k$ . As we show in Section 5, by providing a sound (but incomplete) algorithm, this problem can be easily automated.

To illustrate the terminology used in this paper, we will use the distributed mutual exclusion (DME) protocol [31] as the running example. The goal of this protocol is to ensure that for a distributed system of  $n$  processes in a network with ring topology, only one process in the system is in the critical section at a given point of time. A token is passed between the different processes in the ring. The process who holds the token is the only process able to enter the critical section. Once it is out of the critical section, it can pass

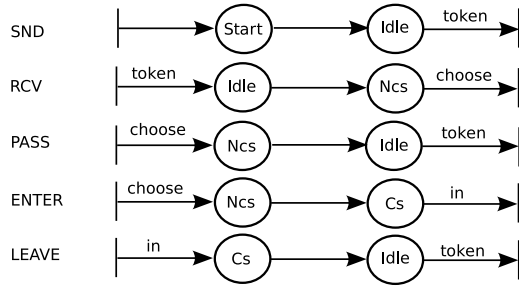


Figure 1: Behavioral Automata for the DME Protocol

the token to its neighbor. The process may receive the token and pass it directly to its neighbor without entering the critical section.

### 3.1 A Process as Behavioral Automata

A homogeneous process in our work is specified in terms of a *behavioral automaton*, which describes an atomic side-effect free action that the process can do. Each behavioral automaton is defined in terms of the input/output behavior and the corresponding observable events of the atomic action of the process as follows:

DEFINITION 3.1 (BEHAVIORAL AUTOMATON). A *behavioral automaton*  $A = (Q, Q_I, Q_F, \Delta, \Delta_I, \Delta_F, M)$ , where

- $Q$  is a nonempty set of states,
- $Q_I \subseteq Q$  is a nonempty set of initial states,
- $Q_F \subseteq Q$  is a nonempty set of final states,
- $\Delta \subseteq Q \times M \times Q$  is the transition relation,
- $\Delta_I \subseteq M \times Q_I$  is the initial transition relation,
- $\Delta_F \subseteq Q_F \times M$  is the final transition relation,
- $M$  is a nonempty set of events.

**Notations.** We write  $q \xrightarrow{e} q'$  if  $(q, e, q') \in \Delta$ ,  $\bullet \xrightarrow{e} q$  if  $(e, q) \in \Delta_I$  and  $q \xrightarrow{e} \bullet$  if  $(q, e) \in \Delta_F$ .

To illustrate, consider the behavioral automata for the DME protocol shown in Figure 1. In the figures, epsilon labels on transitions are omitted. The first automaton SND in the figure models the protocol initiation, where a process generates and passes the token to the next process. Typically the process that would generate the token is decided using other distributed algorithms such as a leader-election algorithm, which we do not model here. Upon receiving a token (modeled as the automaton RCV) a process may choose to exhibit the behavior described by either the automaton PASS or ENTER, effectively passing the token forward (modeled as the output event `token`) or entering the critical section (modeled as generating the output event `in` that can only be consumed by a process in the critical section). The last automaton LEAVE models the behavior of a process leaving the critical section by consuming event `in` and passing forward the token.

Formally, the first automaton SND would be represented by the set of states  $Q = (\text{Start}, \text{Idle})$ . The transition relations for SND are  $\bullet \xrightarrow{\epsilon} \text{Start} \in \Delta_I$ , where the leader process generates a token without any input event,  $\text{Start} \xrightarrow{\epsilon} \text{Idle} \in \Delta$ , where the process changes its state `Start` of being the leader process to `Idle` where it will wait for the token from its neighbor, and finally  $\text{Idle} \xrightarrow{\text{token}} \bullet \in \Delta_F$ , where the process sends the token to its neighbor. Now the process is in state `Idle`, it can do nothing but wait for the token (represented by the automaton RCV).

Proceeding further, a protocol specification is defined as follows:

DEFINITION 3.2 (PROTOCOL SPECIFICATION). A *Protocol Specification* is a set  $\text{Prot} = \{A_1, A_2, \dots, A_m\}$  such that  $\exists i, 1 \leq i \leq m : \bullet \xrightarrow{\epsilon} q \in \Delta_{I_i}$

### 3.2 A Parameterized System

Next, we describe the behavior of a parameterized system,  $\text{sys}(n)$ , containing  $n$  processes each executing according to a given protocol  $\text{Prot}$ . Intuitively, in  $\text{sys}(n)$  we consider that at any instance of the parameterized system the processes can be at any state in any automaton in  $\text{Prot}$ . The interaction between the two processes occur when one of them is at a state ready to initiate an output event and the other is at a state that can consume that event. For example, if a process is at state  $q_1$  of the automaton SND in Figure 1 and the other is at a state  $q_0$  of the automaton RCV, they can communicate via the event `token`. However, to initiate the behavior of the system, it is necessary to have at least one process to be in a state of an automaton in  $\text{Prot}$  which can move without any external trigger, i.e., without being triggered by any input event provided by the output of another process. The condition in Definition 3.2 establishes that there exists such an automaton in the specification which can make a move without any external trigger (absence of input trigger being represented by  $\epsilon$ ). Initially, at least one of the processes in  $\text{sys}(n)$  must follow this automaton behavior.

DEFINITION 3.3 (TOPOLOGY). Given a protocol specification  $\text{Prot} = \{A_1, A_2, \dots, A_m\}$ , a topology is a set of tuples,  $\text{Topo} \subseteq \mathcal{M} \times N \times N$  such that  $\mathcal{M} = \bigcup_{i=1}^m \{M_i : M_i \text{ is set of events in } A_i \in \text{Prot}\}$  and  $N$  is the set of processes in a parameterized system. A tuple  $(e, i, j) \in \text{Topo}$  implies that output  $e$  from  $i$ -th process can be consumed by the  $j$ -th process.

A topology serves to restrict process communication patterns. For example, in the DME protocol, when a process  $i$  sends the event `token`, only the process on its right  $i + 1$  is able to consume this event,  $(\text{token}, i, i + 1)$ . For  $\text{sys}(2)$ , the topology is trivially simple; however, for other protocols (e.g. the dining philosophers protocol described in Section 7.1) the topology plays a major role in distinguishing between the processes.

DEFINITION 3.4 (CONFIGURATION OF  $i$ -TH PROCESS). Given a protocol specification  $\text{Prot} = \{A_1, A_2, \dots, A_m\}$ , a configuration of the  $i$ -th process  $s = (Q \times \text{Prot} \times \mathcal{M})$ , such that

$$Q = \bigcup_{i=1}^m \{Q_i : Q_i \text{ is set of states in } A_i \in \text{Prot}\}$$

$$\mathcal{M} = \bigcup_{i=1}^m \{M_i : M_i \text{ is set of events in } A_i \in \text{Prot}\}$$

The configuration of a process determines what the process is able to do at a certain point of time. For instance, the initial configuration for the process that is generating the token in the distributed mutual exclusion protocol is  $(\text{Start}, \text{SND}, \emptyset)$ , where  $\emptyset$  is the set of output events produced by this process, and it is empty because it has not produced the output event `token` to be consumed by another process yet. The initial configuration of the rest of the processes is  $(\text{Idle}, \text{RCV}, \emptyset)$ , where the processes are idle and waiting to receive the token.

In the following, we present the formal definition of  $\text{sys}(n)$ .

We use the following notation: for a set  $\prod_{i=1}^n \{(Q \times \text{Prot} \times \mathcal{M})\}$ ,

any member  $s$  in the set contains  $n$  tuples where the first, second and third elements of the tuple belong to the sets  $\mathcal{Q}$ ,  $\mathbf{Prot}$  and  $\mathcal{M}$  respectively. We use  $q_i(s)$ ,  $a_i(s)$  and  $m_i(s)$  to denote the value  $q \in \mathcal{Q}$ ,  $A \in \mathbf{Prot}$  and  $e \in \mathcal{M}$  respectively of the  $i$ -th tuple, i.e., the  $i$ -th tuple of  $s$  is  $(q_i(s), a_i(s), m_i(s))$ .

**DEFINITION 3.5 (PARAMETERIZED SYSTEM).** Given a protocol specification  $\mathbf{Prot} = \{A_1, A_2, \dots, A_m\}$ , a parameterized system  $sys(n)$  with  $n$  processes, each of which behaves according to  $\mathbf{Prot}$ , is defined as  $(S, S_I, T, \mathbf{Topo})$ , where  $s \in S$  contains  $n$  tuples and the  $i$ -th element of the tuple represents the configuration of the  $i$ -th process in  $sys(n)$ ,  $s_I \in S_I$  represents the initial configuration of the processes,  $T$  represents the transition relation between one configuration of the processes to another and finally  $\mathbf{Topo}$  is the topology of the system. Here,  $S$ ,  $S_I$ , and  $T$  are defined as:

- $S \subseteq \prod_{i=1}^n (\mathcal{Q} \times \mathbf{Prot} \times \mathcal{M})$
- $S_I \subseteq S$ , where  $\forall s \in S_I, \forall 1 \leq i \leq n : m_i(s) = \emptyset$
- $T \subseteq S \times \mathcal{M} \times \mathcal{M} \times S$

We say that  $s \xrightarrow{e/e'} s' \in T$  if one of the following holds:

1.  $\exists i \in [1, n] : \exists A_x :$ 

$$\left[ \begin{array}{l} e = \epsilon \wedge \bullet \xrightarrow{e} q_i(s) \in \Delta_{I_x} \wedge \\ q_i(s') = q \wedge q \xrightarrow{e'} \bullet \in \Delta_{F_x} \wedge a_i(s') = A_x \wedge \\ m_i(s') = m_i(s) \cup \{e'\} \end{array} \right]$$

$$\wedge \forall j \in [1, n], j \neq i :$$

$$q_j(s) = q_j(s') \wedge a_j(s) = a_j(s') \wedge m_j(s) = m_j(s')$$
2.  $\exists i, j \in [1, n] : \exists A_x :$ 

$$\left[ \begin{array}{l} e \in m_j(s) \wedge \bullet \xrightarrow{e} q_i(s) \in \Delta_{I_x} \wedge \\ q_i(s') = q \wedge q \xrightarrow{e'} \bullet \in \Delta_{F_x} \wedge a_i(s') = A_x \wedge \\ m_i(s') = m_i(s) \cup \{e'\} \wedge m_j(s') = m_j(s) \setminus \{e\} \\ \wedge q_j(s') = q_j(s) \wedge a_j(s') = a_j(s) \wedge \\ (e, j, i) \in \mathbf{Topo} \end{array} \right]$$

$$\wedge \forall k \in [1, n], k \neq i, k \neq j :$$

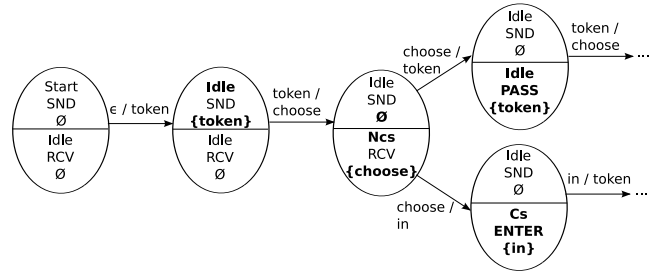
$$q_k(s) = q_k(s') \wedge a_k(s) = a_k(s') \wedge m_k(s) = m_k(s')$$

The transition relations in the above definition can be explained as follows. The first condition represents an autonomous move of the  $i$ -th process without any input event. Whatever event the  $i$ -th process sends due to that autonomous move is kept in the set of output events to be consumed by processes (following the topology). The configurations of the other processes  $j \neq i$  remain unaltered. The second condition represents the case where  $i$ -th process consumes an output event in the list of outstanding events of the  $j$ -th process, where it is stated in  $\mathbf{Topo}$  that process  $i$  is the neighbor to process  $j$  that can consume such an event. Process  $i$  puts in its set of outputs the output event resulting from such an action.

To illustrate our definition of a parameterized system, let us consider a system of two processes ( $sys(2)$ ) that are running the DME protocol. Figure 2 shows part of this system. For this system, the topology is defined as follows.

$$\mathbf{Topo} = \{(\text{token}, 1, 2), (\text{token}, 2, 1), (\text{in}, 1, 1), (\text{in}, 2, 2), \\ (\text{choose}, 1, 1), (\text{choose}, 2, 2)\}$$

The first transition in the figure is an example of a transition following item 1 of the transition relation in Definition 3.5. The process that is generating the token is at the initial state **Start**, its



**Figure 2: Part of  $sys(2)$  for DME. Full  $sys(2)$  is in our TR [24].**

automaton is SND and its set of output events to be consumed by others is empty. The other process is waiting for an output event, so it is in state **Idle**, in automaton **RCV**, also with an empty set of output events. Without any input, the first process starts by sending the output event **token**. The set of output events for that process now has the event **token**. The configuration of the other process configurations remains the same as this transition concerns only the first process.

The second transition in the figure is an example where a process consumes an event sent by another process. Process 1 has the event **token** pending in the set of output events, so according to the topology of the protocol, the process on its right (2) can consume this event. Therefore, process 2 takes **token** as input, so now the set of output events of process 1 is  $\emptyset$ . Process 2 does not need to change its automaton since the current automaton is allowed to receive a token event while in state **Idle**. After consuming this event, process 2 makes an autonomous transition that changes its state from **Idle** to **Ncs** and produces the output event **choose** that allows to choose either to pass the token (top right state of the figure) or enter the critical section (bottom right state).

**DEFINITION 3.6 (PROJECTION).** Given a parameterized system  $sys(n) = (S, S_I, T, \mathbf{Topo})$ , its projected behavior w.r.t. processes in  $R$  is  $sys(n) \downarrow R = (S \downarrow R, S_I \downarrow R, T \downarrow R, \mathbf{Topo})$ , such that

- $S \downarrow R \subseteq \bigcup_{i \in R, s \in S} \{q_i(s)\} \times \bigcup_{i \in R, s \in S} \{a_i(s)\} \times \bigcup_{i \in R, s \in S} \{m_i(s)\}$
- $S_I \downarrow R \subseteq \bigcup_{i \in R, s \in S_I} \{q_i(s)\} \times \bigcup_{i \in R, s \in S_I} \{a_i(s)\} \times \bigcup_{i \in R, s \in S_I} \{m_i(s)\}$
- $s \downarrow R \xrightarrow{m/m'} s' \downarrow R \in T \downarrow R \Leftarrow \\ s \xrightarrow{m/m'} s' \in T \wedge q_i(s) \neq q_i(s') \vee a_i(s) \neq a_i(s')$

In Figure 2, the projection against the 1st process  $sys(2) \downarrow \{1\}$  will include the first transition ( $q \xrightarrow{\epsilon/\text{token}} q'$ ) and the top right transition ( $q \xrightarrow{\text{token}/\text{choose}} q'$ ) in the figure because these transitions affect the state and/or automaton of the first process (the effect of the top right transitions is not shown in the figure, where the first process receives the token and changes its automaton from **SND** to **RCV** and its state from **Idle** to **Ncs**). Projection against the second process  $sys(2) \downarrow \{2\}$  will include all the transitions in the figure except for the first one and the top right one ( $q \xrightarrow{\text{token}/\text{choose}} q'$ ), since these 2 transitions do not affect the state and/or automaton of process 2.

## 4. PROCESS WITH ANY ENVIRONMENT

At its core, our approach depends on the computation of the behavior of one process in the parameterized system in the context of any environment. If  $sys(n) = P_1 \parallel P_2 \parallel P_3 \parallel \dots \parallel P_n$  is a parameterized system containing  $n$  number of processes, each of which



of verifying the properties against  $sys(k)$ . We focus on properties that involve one parameterized process or two parameterized processes that are dependent on each other. Properties of concern are safety (something bad does not happen) and liveness (something good will eventually happen).

**DEFINITION 5.1 (CUT-OFF).** *Given a protocol specification  $Prot = \{A_1, A_2, \dots, A_m\}$  and a parameterized system  $sys(k)$  containing  $k$  processes behaving as per  $Prot$ ,  $k$  is said to be the cut-off if and only if the following holds:*

$$\forall i, 1 \leq i \leq k : sys(k) \models \varphi(i) \Leftrightarrow \forall n \geq k, \forall i, 1 \leq i \leq n : sys(n) \models \varphi(i)$$

$$\forall i, j, 1 \leq i, j \leq k, i \neq j : sys(k) \models \varphi(i, j) \Leftrightarrow \forall n \geq k, \forall i, j, 1 \leq i, j \leq n, i \neq j : sys(n) \models \varphi(i, j)$$

where  $\varphi(i)$  and  $\varphi(i, j)$  represent properties involving one (the  $i$ -th process) and two ( $i$ -th and  $j$ -th processes) in the parameterized system respectively, and in case of  $\varphi(i, j)$ , the  $i$ -th and  $j$ -th processes are dependent on each other or communicate through a non-parameterized process.

In the next section, we will prove that  $k$  is a cut-off if and only if  $sys(k)$  can replicate all possible behavior captured by 1E-behavior obtained from the  $Prot$  specification. Specifically, if  $sys(k)$  “simulates” 1E-behavior, then  $k$  is a cut-off. The simulation relation between two states is defined as follows.

**DEFINITION 5.2 (SIMULATION [27]).** *Given a labeled transition system (a system where transitions are labeled with events), state  $s$  is said to be simulated by a state  $t$ , denoted by  $s \prec t$ , if and only if*

$$\forall e, s' : s \xrightarrow{e} s' \Rightarrow \exists t' : t \xrightarrow{e} t' \wedge s' \prec t' \quad (1)$$

We say that a labeled transition system  $LTS_1$  is simulated by a labeled transition system  $LTS_2$ , denoted by  $LTS_1 \prec LTS_2$ , if and only if for all start states  $s$  of  $LTS_1$ , there exists a start state  $t$  in  $LTS_2$ , such that  $s \prec t$ .

Proceeding further, our algorithm for computing the cut-off based on the above simulation relation is implemented in Procedure CutOff.

---

#### Procedure CutOff (Prot)

```

Compute 1E-behavior from Prot
 $k \leftarrow 2$ 
while true do
  if  $\exists$  state  $s$  in  $sys(k) : 1E\text{-behavior} \prec s$  then
    return  $k$ ;
  else
     $k++$ ;
  end if
end while

```

---

Procedure CutOff enumerates for different values of  $k$  the behavior of  $sys(k)$ , where each process behaves according to protocol specification  $Prot$ , and checks whether  $sys(k)$  contains a state that simulates 1E-behavior for the given  $Prot$ . If for a specific value of  $k$  such a state is present, then that value of  $k$  is the cut-off. Recall from Definition 4.2 that 1E-behavior contains  $\tau$  transitions representing chaining of output from one automaton to the input of another. However, such transitions are not present in the  $sys(k)$  definition (Definition 3.5). Furthermore, for some parameterized systems, there can be one process that is not parameterized. In other

words, there is only one process of that type in all instances of the parameterized system. For instance, if  $n$  processes have access to a shared memory, then the shared memory is a non-parameterized process in the system. Therefore, since we only care for properties related to parameterized processes, any transition  $q \xrightarrow{e/e'} q'$  in the 1E-behavior and the  $sys(k)$  of these systems such that

$$\exists A_x : q' \xrightarrow{e'} \bullet \in \Delta_{F_x} \wedge \bullet \xrightarrow{e} q \in \Delta_{I_x}$$

where  $A_x$  is a behavioral automaton for the non-parameterized process is substituted with a  $\tau$  transition. An example of a parameterized system with a non-parameterized process is presented in Section 7.2. We use the following variation of Equation 1 in our definition of simulation.

$$\forall e, s' : s \xrightarrow{\tau^*e} s' \in 1E\text{-behavior} \Rightarrow \exists t' : t \xrightarrow{\tau^*e} t' \in sys(k) \wedge s' \prec t'$$

In the above,  $\tau^*e$  represents moves of zero or more  $\tau$  transitions followed by an  $e$  transition.

As the verification of parameterized system is undecidable [4], our procedure may not terminate; however, if it terminates, it will return the smallest cut-off value  $k$  for the corresponding protocol specification  $Prot$ . We will prove the soundness of the procedure in Section 6.

Figure 2 shows part of the  $sys(2)$ , two processes executing DME protocol as specified by Figure 1. The system  $sys(2)$  simulates the corresponding 1E-behavior (see Figure 3) of the protocol. Therefore, for the DME protocol in a ring topology,  $k = 2$  is the cut-off. It is worth mentioning that [19] provided a general cut-off valuation of 4 for verifying mutual exclusion property of any parameterized system with ring topology (e.g., DME protocol). However, as we consider the system description ( $sys(k)$ ) in our technique, we are able to identify a tighter cut-off value for the DME protocol.

## 6. PROOF OF SOUNDNESS

We prove that given a protocol specification  $Prot$  and a parameterized system  $sys(n)$ , the output  $k$  of Procedure CutOff described in Section 5 is the cut-off for protocol  $Prot$  as per the Definition 5.1. The following lemmas will form the basis of our proof.

**LEMMA 1.** *For any parameterized system  $sys(k)$ ,*

$$\forall i, 1 \leq i \leq k : sys(k) \models \varphi(i) \Leftrightarrow sys(k) \downarrow \{i\} \models \varphi(i)$$

*Proof:* The proof is immediate from the nature of the property  $\varphi(i)$  and the projection operation (Definition 3.6). The property is only concerned with the configurations and the events related to the  $i$ -th process and as such, configurations and the events related solely to the processes  $j \neq i$  are irrelevant for the satisfiability of the property by  $sys(k)$ . ■

**LEMMA 2.**  $\forall k, 1 \leq k \leq n, \forall i, 1 \leq i \leq k : sys(k) \downarrow \{i\} \prec sys(n) \downarrow \{i\}$

*Proof:* The proof is realized by contradiction. Assume that there exists a state  $s \downarrow \{i\}$  in  $sys(k) \downarrow \{i\}$  reachable from its start state after a sequence of events (of the  $i$ -th process) such that  $s \downarrow \{i\}$  is not simulated by any of the states in  $sys(n) \downarrow \{i\}$  reachable from its start state via the same sequence of events.

As the same event sequence is considered from the respective start states, there exists some  $t \downarrow \{i\}$  in  $sys(n) \downarrow \{i\}$  reachable via this event sequence, such that the configuration of the  $i$ -th process in  $t \downarrow \{i\}$  is the same as that of the  $i$ -th process in  $s \downarrow \{i\}$ . However,

as  $s \downarrow \{i\}$  is not simulated by  $t \downarrow \{i\}$ , there exists at least one action of the  $i$ -th process from  $s \downarrow \{i\}$  that is not present from  $t \downarrow \{i\}$ .

This action cannot solely be output event (of the form  $\epsilon/e$ ) of the  $i$ -th process; these types of events do not depend on the environment of the process and can always occur as long as the process is in a suitable configuration. Therefore, the action must involve an input event where the  $i$ -th process relies on its environment to provide such an input.

In other words, at state  $s \downarrow \{i\}$  of  $sys(k) \downarrow \{i\}$ , the  $i$ -th process can move on an input event while at state  $t \downarrow \{i\}$  of  $sys(n) \downarrow \{i\}$ , the  $i$ -th process cannot make a move on the same input event. This implies that the environment of the  $i$ -th process at state  $s$  in  $sys(k)$  provides the required input, while the environment of the  $i$ -th process at any state  $t$  in  $sys(n)$  is unable to provide the same input. Let the neighbor of the  $i$ -th process, as per the topology of the system, responsible for providing this input be  $j_1$ -th process. Therefore, in  $sys(k)$ , the  $j_1$ -th process is able to provide the input at state  $s$ , while the  $j_1$ -th process in  $sys(n)$  is unable to do so as it is waiting for its own neighbor, say  $j_2$ -th process.

Proceeding further, the input to the  $i$ -th process at all states  $t$  (s.t.  $t \downarrow \{i\} = s \downarrow \{i\}$ ) in  $sys(n)$  is disabled as it is waiting for  $j_1, j_2, j_3, \dots$  processes to move to their respective configurations such that  $j_1$ -th process can provide the input. However, that is not the case at state  $s$  in  $sys(k)$ . As processes in both  $sys(k)$  and  $sys(n)$  behave according to the same protocol specification  $\text{Prot}$ , the above can only happen when  $n < k$ . This leads to contradiction proving that our initial assumption is incorrect. ■

**THEOREM 2.** *Given a protocol specification  $\text{Prot}$  and a parameterized system  $sys(n)$  where each process behaves as described in  $\text{Prot}$ ,*

$$\begin{aligned} \exists \text{ state } s \text{ in } sys(k) : 1E\text{-behavior} \prec s &\Leftrightarrow \\ (\forall i, 1 \leq i \leq k : sys(k) \models \varphi(i) &\Leftrightarrow \\ \forall n \geq k, \forall i, 1 \leq i \leq n : sys(n) \models \varphi(i)) & \end{aligned}$$

*In the above, 1E-behavior is computed from the specification  $\text{Prot}$ .*

*Proof:* Let  $s = (c_1, c_2, \dots, c_k)$  be the state in  $sys(k)$  that simulates 1E-behavior and  $c_j$  ( $1 \leq j \leq k$ ) be the configuration (Definition 3.4) of the  $j$ -th process at  $s$ . Therefore, using Theorem 1,  $s$  captures all possible behavior of some process in  $sys(k)$  and its environment. For the  $l$ -th process with configuration  $c_l$  in  $s$ , let  $E_l$  denote its environment, i.e.,

$$E_l = \bigcup_{p=1}^k \{c_p : p \neq l\}. \quad (2)$$

As all processes in the parameterized system  $sys(k)$  behave according to the same protocol specification  $\text{Prot}$ , for a specific process, the  $i$ -th process, there exists at most  $k$  different states  $s_1, s_2, \dots, s_k$  in  $sys(k)$  such that for each  $l$  ( $1 \leq l \leq k$ ), the  $i$ -th process is in the configuration  $c_l$  with the environment  $E_l$  (Equation 2) at state  $s_l$ . Therefore, all possible behavior of some process and its environment as captured by 1E-behavior of the  $\text{Prot}$ , is exhibited by the  $i$ -th process in  $sys(k)$ . I.e.,

$sys(k) \downarrow \{i\}$  exhibits all possible behavior of  $i$ -th process in  $sys(k)$

$$\Leftrightarrow (\forall i, 1 \leq i \leq k : sys(k) \downarrow \{i\} \models \varphi(i) \Leftrightarrow sys(k) \models \varphi(i))$$

(From Lemma 1)

$$\Leftrightarrow (\forall i, 1 \leq i \leq k : sys(k) \downarrow \{i\} \models \varphi(i) \Leftrightarrow \forall n, n \geq k, \forall i, 1 \leq i \leq k : sys(n) \downarrow \{i\} \models \varphi(i))$$

(From Lemma 2)

$$\Leftrightarrow (\forall i, 1 \leq i \leq k : sys(k) \models \varphi(i) \Leftrightarrow \forall n, n \geq k, \forall i, 1 \leq i \leq k : sys(n) \models \varphi(i))$$

(From Lemma 1) ■

Next we prove the following theorem

**THEOREM 3.** *Given a protocol specification  $\text{Prot}$  and a parameterized system  $sys(n)$  where each process behaves as described in  $\text{Prot}$ ,*

$$\begin{aligned} \exists \text{ state } s \text{ in } sys(k) : 1E\text{-behavior} \prec s &\Leftrightarrow \\ (\forall i, j, 1 \leq i, j \leq k, i \neq j : sys(k) \models \varphi(i, j) &\Leftrightarrow \\ \forall n \geq k, \forall i, j, 1 \leq i, j \leq n, i \neq j : sys(n) \models \varphi(i, j)) & \end{aligned}$$

*where the behavior of one process (say the  $j$ -th process) is dependent on the other (say the  $i$ -th process) or both processes  $i$  and  $j$  communicate through a non-parameterized process.*

*Proof:* The proof of the theorem relies on the following observations. For brevity, we omit the proofs of the statements; the proofs being similar to Lemmas 1 and 2.

$$\begin{aligned} \forall i, j, 1 \leq i, j \leq k, i \neq j : sys(k) \models \varphi(i, j) &\Leftrightarrow \\ sys(k) \downarrow \{i, j\} \models \varphi(i, j) & \\ \forall k, 1 \leq k \leq n, \forall i, j, 1 \leq i, j \leq k, i \neq j : & \\ sys(k) \downarrow \{i, j\} \prec sys(n) \downarrow \{i, j\} & \end{aligned} \quad (3)$$

Recall that 1E-behavior captures the behavior of any process  $i$  and its environment. It does not distinguish between the processes in the environment, i.e., the environment comprises of all the processes with which the process  $i$  interacts directly or indirectly. Therefore, a state  $s$  in  $sys(k)$  simulating 1E-behavior implies that the state captures all possible behavior of a process  $i$  and its environment containing another process  $j$  dependent on  $i$ . As in Theorem 2, we consider environments of pairs of processes in state  $s$ . Proceeding further, we fix the pair  $i, j$  and state that there exists at most  $k$  different states for this pair of processes which collectively captures all possible behavior of pair of processes as per the 1E-behavior. Finally, the proof follows from the statements in Equation 3. ■

**THEOREM 4 (SOUNDNESS).** *If Procedure  $\text{CutOff}$  terminates, the return value  $k$  is the cut-off as per the Definition 5.1.*

*Proof:* The proof follows from Theorems 2 and 3. ■

**REMARK 1.** *The proposed algorithm addresses the problem of verifying properties of the form  $\varphi(i, j)$  where  $i$  and  $j$  are parameterized processes and the behavior of process (say the  $j$ -th process) is dependent on the other process (say the  $i$ -th process). In other words, actions by  $j$ -th process are done as a direct result (or in response) to actions done by the  $i$ -th process, or they communicate with each other through a non-parameterized process (example in Section 7.2). Note that processes  $i$  and  $j$  need not be neighbors. For instance, in the DME protocol, process  $j$  cannot start any behavior until receiving a token generated by process  $i$ , therefore the behavior of process  $j$  is dependent on process  $i$ ; however process  $j$  need not be directly connected to process  $i$  where the token can be passed by  $n - 2$  processes before reaching process  $j$ .*

## 7. CASE STUDIES

We have worked several nontrivial examples to validate our approach: the dining philosopher protocol [16] and Spin lock, a locking protocol for mutual exclusive access to an object [3]. We compared our results with that obtained using existing work. We found that the cut-off obtained by our technique is either smaller or equal to the cut-off produced by existing techniques. Furthermore, in contrast to existing techniques which are either applicable to parameterized systems with specific topology or rely on smart representation and/or abstraction of the system behavior, our technique

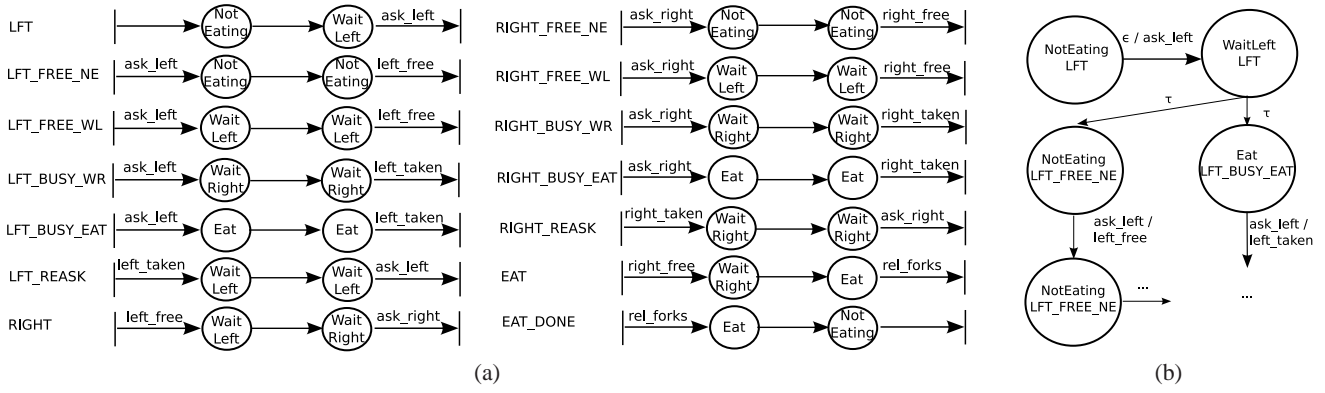


Figure 4: (a) The Behavioral Automata for the Dining Philosophers Protocol. (b) Part of the Behavior of a Philosopher Process

uniformly handles systems with different topologies and is based on a standard transition system based representation.

## 7.1 Dining Philosophers Protocol

Dining philosophers protocol [16] models a classic multi-process synchronization problem. Among others, Emerson and Kahlon [17] have used it as a candidate parameterized system. For this protocol, the number of philosophers is the system parameter. The standard definition models processes as philosophers sitting in a circle (a ring topology) with a fork between each 2 philosophers. The main objective of a philosopher process is to acquire the fork to its left and right and start eating. We model this protocol using the behavioral automata shown in Figure 4 (a).

The first automaton LFT in this figure represents the behavior of a philosopher when it decides that it wants the left fork and asks its neighbor for it. The automaton contains  $Q = (q_0, q_1)$  where  $q_0 = \text{NotEating}$ , and  $q_1 = \text{WaitLeft}$ . The transition relations are  $\bullet \xrightarrow{\epsilon} q_0 \in \Delta_I$ , where the philosopher initiates this behavior without any input event,  $q_0 \xrightarrow{\epsilon} q_1 \in \Delta$ , where the philosopher decides that it wants the left fork so it changes its state from  $\text{NotEating}$  to  $\text{WaitLeft}$ , and finally  $q_1 \xrightarrow{\text{ask\_left}} \bullet \in \Delta_F$ , where the philosopher sends the request for the left fork to its neighbor. Other automata are similar in nature.

Given this protocol specification, using our technique, we would like to find the smallest number of processes ( $k$ ) for this parameterized system such that verifying any correctness property on a system with  $k$  processes is necessary and sufficient to say that the property is true for any system involving  $n$  processes for any  $n > k$ .

**Behavior of a Process in Any Environment.** To that end, the first step in our technique is to find the behavior of a process in any environment (see Section 4). To compute this, one would start by composing the automata described in Figure 4(a) in accordance with the definition of the composition operator ( $\otimes$ ). This will be repeated until all the automata in Figure 4(a) are composed and the composition satisfies the Definition 4.2 for 1E-behavior.

Figure 4(b) shows part of this 1E-behavior. The transition between state ( $\text{NotEating}, \text{LFT\_FREE\_NE}$ ) on the bottom left of the figure and state ( $\text{NotEating}, \text{LFT\_FREE\_NE}$ ) on the bottom right models an internal transition in automaton LFT\_FREE\_NE in Figure 4(a). The  $\tau$  transition between the states ( $\text{WaitLeft}, \text{LFT}$ ) and ( $\text{NotEating}, \text{LFT\_FREE\_NE}$ ) models the chaining between automata LFT and LFT\_FREE\_NE, where the output event of the former is equal to the input event of the latter.

**Cut-off Value for the System Parameter.** Once the 1E-behavior describing the behavior of one philosopher in the context of any en-

vironment is generated, the next step is to find the smallest network that a philosopher can actually exhibit this behavior. The size of such network is the cut-off. In order to find this cut-off, we follow Procedure CutOff described in Section 5, where we start building a system of 2 philosophers ( $\text{sys}(2)$ ) and check if there is a state in this system that simulates 1E-behavior. Until we find a system with a state that simulates 1E-behavior, we increase the number of philosophers of the system.

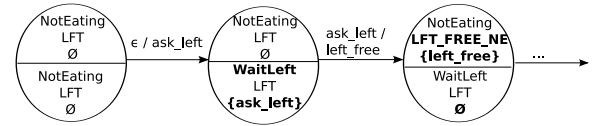


Figure 5: Part of  $\text{sys}(2)$ . Full  $\text{sys}(2)$  is in our TR [24].

Part of the system  $\text{sys}(2)$  for 2 philosophers is displayed in Figure 5. The first state in the figure is the initial configuration where all philosophers are not eating. The first transition is an example where the second philosopher sends a request for the left fork. The changes in the state configurations are highlighted in bold.

Unlike the DME protocol, 1E-behavior for dining philosophers is not simulated by  $\text{sys}(2)$ . The reason is that for 2 philosophers, some states in 1E-behavior are not simulated in  $\text{sys}(2)$ . For instance, in 1E-behavior in Figure 4(b), from state ( $\text{WaitLeft}, \text{LFT}$ ) both the states ( $\text{NotEating}, \text{LFT\_FREE\_NE}$ ) and ( $\text{Eat}, \text{LFT\_BUSY\_EAT}$ ) are reachable through  $\tau$  transitions where transitions ( $\text{NotEating}, \text{LFT\_FREE\_NE}$ )  $\xrightarrow{\text{ask\_left}/\text{left\_free}}$   $q'$  and ( $\text{Eat}, \text{LFT\_BUSY\_EAT}$ )  $\xrightarrow{\text{ask\_left}/\text{left\_taken}}$   $q''$  are then possible. Therefore, for  $\text{sys}(2)$  to simulate 1E-behavior, there should be a state  $s$  in  $\text{sys}(2)$  with both output transitions  $s \xrightarrow{\text{ask\_left}/\text{left\_free}}$   $s'$  and  $s \xrightarrow{\text{ask\_left}/\text{left\_taken}}$   $s''$ . No state in  $\text{sys}(2)$  can have both these output transitions. Since only 2 philosophers and 2 forks exist in the system, the reply for a request of the left fork will either be that the fork is busy or free, but no one state can have both replies as output transitions. In case there are 3 philosophers, this is possible because of the asynchronous nature of our modeling; so 2 philosophers can receive 2 requests for left fork from their neighbors, where one of the recipients is eating and the other is not eating. Therefore, there will be one state in  $\text{sys}(3)$  where it is possible to have both output transitions: one corresponding to the philosopher eating that replies that the fork is busy and the other to the one that is not eating that replies that the fork is free.

$\text{Sys}(3)$  can simulate 1E-behavior, therefore the cut-off for this protocol is 3 ( $\text{sys}(3)$  is presented in our report [24]).



**Topology of the System.** For the parameterized system  $sys(k)$  to exhibit the intended behavior of the protocol, the protocol topology needs to be stated while building the system. Consider in the Dining philosopher protocol if we have 3 philosophers and no topology was stated, then a philosopher can ask its same neighbor for left and right forks, while this is not possible. The reason is that the behavioral automata does not enforce any kind of topology in order to ensure that the composition of automata will present the behavior of one process in the context of *any* environment. Therefore, the topology of the system needs to be stated when building the parameterized system  $sys(k)$ . For example, when philosopher  $i$  sends the event `ask_left`, only the philosopher on its left  $i - 1$  is able to consume this event, where  $(ask\_left, i, i - 1) \in \text{Topo}$ . As for event `ask_right`, the philosopher on the right of philosopher  $i$ , the  $(i + 1)$ -th philosopher, is the one who can receive this event from philosopher  $i$ , denoted as  $(ask\_right, i, i + 1)$ .

**Comparison with other work.** Emerson and Kahlon [17] present the first fully automated verification for the dining philosophers protocol. A system in their technique is defined using two sets, one for processes (philosophers) and one for tokens (forks). They prove that reasoning about deadlock characteristics, safety and liveness properties for a pair of *adjacent processes* for arbitrary rings can be reduced to a ring of size at most 5. However, we obtain a tighter cut-off value 3 due to the fact that the behavior of the participating processes is considered in our technique. While Emerson and Namjoshi [19] found that the cut-off for systems with ring topologies for properties  $\varphi(i, i + 1)$  (properties of neighboring pairs of processes) is 3, the technique in [19] requires a token passing model, where a single token is transmitted in a clockwise direction, thus it is not applicable to the dining philosophers that does not follow such a model.

## 7.2 Spin Lock

Spin locks [3] offer a simple mechanism to realize mutually exclusive access of objects by threads. The object can have two states: not-busy (when is not accessed by any thread, state NB) and busy (when it accessed by some thread, state B). A not-busy object, upon receiving a `req` from a thread, replies back with an `ack` message and behaves like a busy object. A busy object, on the other hand, denies all requests from threads using a `nack` message or goes to a not-busy state on receiving a `rel` (release) signal from the lock releasing thread. Each thread process can lock an object if it receives `ack` in response to a `req` signal. This system follows a star topology, where all processes are connected to the object.

The behavioral automata for threads and the objects in the Spin Lock are displayed in Figure 6(a) and (b) respectively.

**Behavior of a Process in Any Environment.** Unlike the DME and Dining Philosophers protocol, the behavior of the system in Spin Lock is dependent on an object that is not a parameter of the system. In other words, there can be  $n$  processes in the system, but only one object can be present in the system. Since the behavior of the processes is dependent on the behavior of the object, building the 1E-behavior of the system requires the presence of the object. Therefore, building the 1E-behavior of such system with processes that are not parameters is done in 2 steps. First, the 1E-behavior of the system that includes the object is built. Then, for any transition that belongs to the object consuming an event and producing another event, these object transitions are replaced by a  $\tau$  transition.

Figure 7 displays part of the 1E-behavior of Spin Lock with the object behavior included. The transitions highlighted by the box belong to the actions where the object receives a request and sends either an acknowledgement that it is not-busy or a `nack` saying it is busy. Since we only care about the behavior of parameterized

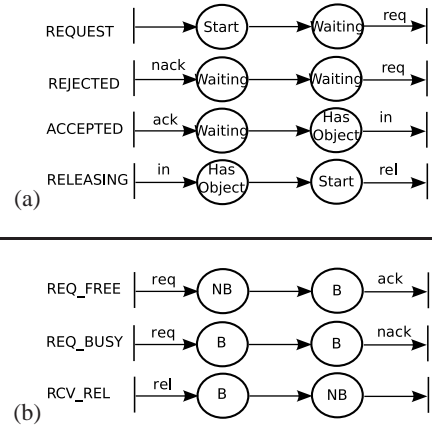


Figure 6: Automata for Spin Lock: (a) Process. (b) Object.

processes, these 2 transitions are going to be substituted by a  $\tau$  transition. The full 1E-behavior of this system is presented in our technical report [24].

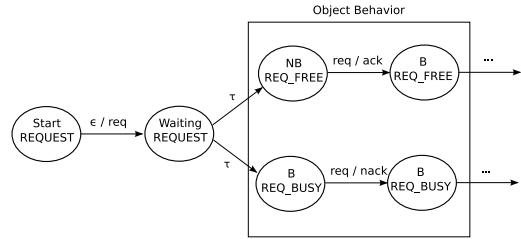


Figure 7: Part of the Behavior of a Process in Spin Lock

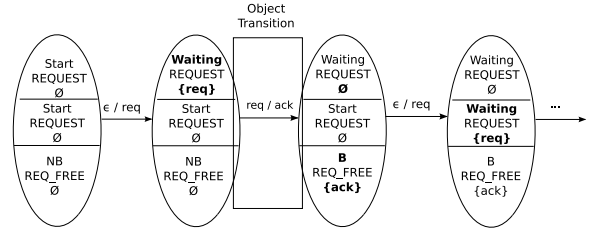


Figure 8: Part of  $sys(2)$  for Spin Lock

**Cut-off Value for the System Parameter.** Building  $sys(k)$  for systems with one or more processes that are not parameters to the system is similar to building the 1E-behavior of these systems. First, the  $sys(k)$  of the system with the object is built. Second, the transitions that belong to actions done by the process that is not a parameter are replaced with  $\tau$  transitions.

Part of  $sys(2)$  for the Spin Lock protocol with 2 threads and the object is displayed in Figure 8, where the transition  $q \xrightarrow{req/ack} q'$  is to be substituted by a  $\tau$  transition since it belongs to the non-parameterized process (the object). Our technique found that the cut-off value for this system is 2, while Basu and Ramakrishnan [9] tackling the same problem found the cut-off to be 3. In contrast to our technique, the method proposed in [9] is based on fixed point computation and abstraction-based acceleration of properties of environment. As such, results obtained using Basu and Ramakrishnan's technique [9] depend on the quality of the abstraction and may not always terminate with smallest cut-off value.

## 8. CONCLUSION AND FUTURE WORK

Verification of correctness properties for parameterized systems is an important problem [13, 15, 18, 19, 21, 23, 29]. Considering that this problem is undecidable in general [4], techniques and heuristics for solving it for a subset of scenarios is an equally important problem. To that end, computing the cut-off of the system parameter is shown to be an effective technique for solving the parameterized verification problem [18, 19, 21].

In contrast to the existing techniques, our approach, based on behavioral-automata composition, can be applied to any parameterized systems independent of the communication topology. It provides a fully-automatic method for obtaining system cut-off for a parameterized system expressed using a standard automata-based modeling approach. Furthermore, effectively utilizing system descriptions allows us to obtain a system-specific cut-off, which in at least 3 cases is found to be lower than previously discovered bounds (DME protocol, Dining Philosophers Protocol and Spin Lock). A system cut-off, to a large extent, dictates the state space that needs to be explored by a formal verification technique. The systematic approach of finding this cut-off that our approach provides is thus an important and foundational advance towards improved scalability of formal verification techniques.

Future work includes extending the theoretical and practical treatment of behavioral-automata composition in several directions. We plan to explore more expressive representation of protocols that can capture synchronous communication between processes and parameterized systems with infinite-domain data.

## 9. ACKNOWLEDGEMENTS

This work was supported in part by the US National Science Foundation under grants CNS-06-27354, CNS-07-09217, and CAREER-08-46059. Thanks to David Samuelson and anonymous ESEC/FSE '09 reviewers for critical feedback on the draft.

## 10. REFERENCES

- [1] P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezzina. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, pages 721–736, 2007.
- [2] P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d’Orso. Regular model checking made simple and efficient. In *CONCUR*, pages 116–130, 2002.
- [3] T. E. Anderson. The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 1(1):6–16, 1990.
- [4] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.
- [5] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV*, pages 221–234, 2001.
- [6] P. Baldan, A. Corradini, and B. König. A framework for the verification of infinite-state graph transformation systems. *Inf. Comput.*, 206(7):869–907, 2008.
- [7] P. Baldan, B. König, and A. Rensink. Graph grammar verification through abstraction. In *Dagstuhl Seminar Proceedings 04241*, 2005.
- [8] T. Ball, S. Chaki, and S. K. Rajamani. Parameterized verification of multithreaded software libraries. In *TACAS*, pages 158–173, 2001.
- [9] S. Basu and C. R. Ramakrishnan. Compositional analysis for verification of parameterized systems. *Theor. Comput. Sci.*, 354(2):211–229, 2006.
- [10] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, pages 403–418, 2000.
- [11] A. Bouajjani, Y. Jurski, and M. Sighireanu. Reasoning about dynamic networks of infinite-state processes with global synchronization. HAL - CCSD, 2006.
- [12] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS*, pages 690–705, 2007.
- [13] E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *VMCAI, 126–141*, 2006.
- [14] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *CONCUR*, pages 395–407, 1995.
- [15] E. M. Clarke, M. Talupur, and H. Veith. Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In *TACAS*, pages 33–47, 2008.
- [16] E. Dijkstra. Two starvation free solutions to a general exclusion problem. EWD 625, Plataanstraat 5, 5671 AL Neunen, The Netherlands.
- [17] E. A. Emerson and V. Kahlon. Model checking large-scale and parameterized resource allocation systems. In *TACAS*, pages 251–265, 2002.
- [18] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME*, pages 247–262, 2003.
- [19] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995.
- [20] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems (extended abstract). In *CAV*, pages 87–98, 1996.
- [21] E. A. Emerson, R. J. Treffer, and T. Wahl. Reducing model checking of the few to the one. In *ICFEM*, pp. 94–113, 2006.
- [22] D. Fisman, O. Kupferman, and Y. Lustig. On verifying fault tolerance of distributed protocols. In *TACAS*, 315–331, 2008.
- [23] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.
- [24] Y. Hanna, S. Basu, and H. Rajan. Behavioral automata composition for automatic topology independent verification of parameterized systems. Technical Report 09-17, Computer Sc., Iowa State U., 2009.
- [25] C. N. Ip and D. L. Dill. Verifying systems with replicated components in murphi. In *CAV*, pages 147–158, 1996.
- [26] M. Llorens and J. Oliver. Introducing structural dynamic changes in petri nets: Marked-controlled reconfigurable nets. In *ATVA*, pages 310–323, 2004.
- [27] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.
- [28] A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS '01*, 92–97.
- [29] A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, infly)-counter abstraction. In *CAV*, pp. 107–122, 2002.
- [30] M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *TACAS*, pages 18–32, 2008.
- [31] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 68–80, 1990.