Automating Cut-off for Multi-parameterized Systems*

Youssef Hanna, David Samuelson, Samik Basu, and Hridesh Rajan

Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA, USA
{ywhanna,sralmai,sbasu,hridesh}@iastate.edu

Abstract. Verifying that a parameterized system satisfies certain desired properties amounts to verifying an infinite family of the system instances. This problem is undecidable in general, and as such a number of sound and incomplete techniques have been proposed to address it. Existing techniques typically focus on parameterized systems with a single parameter, (i.e., on systems where the number of processes of exactly one type is dependent on the parameter); however, many systems in practice are multi-parameterized, where multiple parameters are used to specify the number of different types of processes in the system. In this work, we present an automatic verification technique for multi-parameterized systems, prove its soundness and show that it can be applied to systems irrespective of their communication topology. We present a prototype realization of our technique in our tool *Golok*, and demonstrate its practical applicability using a number of multi-parameterized systems.

1 Introduction

A large class of protocols described for concurrent systems, e.g., client-server protocols and multi-threaded locking protocols, do not enforce any bound on the number of processes that constitute the systems. Behavior of systems executing such protocols are modeled as parameterized systems where the parameter specifies the number of *homogeneous* processes in the system [1]. Verification of a parameterized system, therefore, amounts to verifying every instance of the system obtained by fixing the value of the parameter. In short, if $sys(n)$ is a parameterized system, where n specifies the number of homogeneous processes in the system, then verifying whether $sys(n)$ satisfies a certain desired property involves verifying that for all possible values of n, the system satisfies the property. This problem is undecidable in general [2].

Driving Problem. There is a rich body of work on parameterized system verification [3, 4, 5, 6] that focuses on providing sound and incomplete methods to verify a *singly*-parameterized system. For a singly-parameterized system, the parameter specifies the number of exactly one type of homogeneous processes. However, in practice, there are many systems that are inherently multi-parameterized; examples include wireless sensor networks consisting of multiple types of nodes: sensors and aggregators [7] and distributed producer-consumer based systems [8] involving multiple producers

* This work has been supported in part by the US National Science Foundation under grants CNS-06-27354, CNS-07-09217, and CCF-08-46059.

J.S. Dong and H. Zhu (Eds.): ICFEM 2010, LNCS 6447, pp. 338–354, 2010.

and consumers. On the one hand, in most cases, verification techniques for singly-parameterized systems are either not applicable to multi-parameterized systems, or it is not immediate how one can extend these techniques to analyze systems with multiple parameters. On the other hand, the few techniques that can indeed verify multi-parameterized systems suffer from the drawback that they require non-trivial human guidance to obtain the appropriate protocol specification (e.g. [9, 10]) and/or work only for systems with certain topologies (e.g. [11]).

Consider that a multi-parameterized system with t different types of processes is described by $sys(\bar{n}_t)$ where $\bar{n}_t := n_1, n_2, \ldots, n_t$ and the parameter n_p denotes the number of processes of type p. The objective is to verify whether the system satisfies a given property for all possible valuations of each n_p in $\bar{n}_t$. This can be realized by identifying a specific instance of the system: $sys(\bar{k}_t)$ (where $\bar{k}_t := k_1, k_2, \ldots, k_t$) such that $sys(\bar{k}_t)$ satisfies the given property if and only if $sys(\bar{n}_t)$ satisfies the same, for all $\bar{n}_t \geq \bar{k}_t$ (i.e., $\forall p : n_p \geq k_p$). The parameter values in $\bar{k}_t$ corresponding to the specific instance of the system are referred to as the *cut-off*.

Our Solution. In this paper, we propose a technique, leveraging on our previous work [12], for automatically identifying such a cut-off. The technique, unlike the existing ones, is independent of both the communication topology and the property to be verified, and relies on simple input/output automata based representation of different types of processes in the system.

We consider a set of behavioral automata (introduced in [12]) to describe the input/output behavior of different types of processes in the system. The central theme of our technique is to automatically

1. compute the set of *maximal* behavior of the system (in terms of input/output) that can be induced by output action of each type of processes in the parameterized system, and
2. identify the *minimal* instance of the parameterized system that includes all such maximal behavior.

We prove that the parameter values corresponding to this minimal instance is the cut-off; more precisely, for any $\texttt{LTL}\setminus\texttt{X}$ (Linear Temporal Logic without "next" operator) properties which involve either actions of exactly one process or actions of two or more directly communicating processes, the instance of the parameterized system with the cut-off valuation for the parameters satisfies the property if and only if any other larger (in terms of parameter values) instance satisfies the same property.

Significant extension of [12]. While the core of the technique described in this paper is same as the one proposed and developed in [12], there are several important and non-trivial issues that are addressed in the current paper. In [12], cut-off valuation is computed for parameterized system where the parameter specifies the number of exactly one type of homogeneous process. The computed maximal behavior, therefore, is induced by one type of process. In the current paper, as there are multiple types of processes whose number is parameterized, in Step 1, it is necessary to compute the maximal behavior induced by all of them. We show that the collection containing the induced maximal behavior by *each* type of process is equivalent to the induced maximal

behavior by processes of *all* types. We further show that, it is *sufficient* to compute the induced maximal behavior for only those types of processes that are capable of making an autonomous move (without requiring external stimuli/input) from their initial states. These two conditions reduce the complexity of identifying the induced maximal behavior in Step 1 and thereby, reduce the complexity of the overall technique. Finally, for Step 2, we use a simple breadth-first strategy for incrementing parameter values to identify system instances; such strategy was not needed in [12] as the system, under consideration, was singly parameterized.

Contribution. The summary of contributions of our technique are:

- To the best of our knowledge, we present the first automatic technique for verifying multi-parameterized systems that has the following features:
  1. the technique is applicable for verifying $\texttt{LTL}\setminus\texttt{X}$ properties over arbitrary homogeneous processes (in contrast to [11] which focuses on resource allocation systems);
  2. the technique is automatic, requires no human intervention (unlike several methods, e.g. [9], that rely on smart representation of the system being verified);
  3. the technique is independent of the communication topology (unlike [11] that works only for systems with ring topology), which, along with automation, broadens the scope of its application in practical settings.
- We present the implementation of our technique in a tool, *Golok* and discuss several optimizations deployed to speedup the cut-off generation process. We demonstrate the robustness and scalability of our technique and implementation using different canonical multi-parameterized systems.

Organization. This paper is organized as follows. Section 2 discusses related work. Section 3 describes our technique for specifying a system using a variant of the Dining Philosophers protocol as an illustrative example. Section 4 describes how the maximal behavior induced by a process of some type p in the context of any environment is generated and shows the procedure for generating the cut-off. Proof of soundness of our technique is presented in Section 5. Section 6 describes our tool. Section 7 presents the different case studies we used to evaluate our technique and the obtained results from our tool and Section 8 offers final remarks.

2 Related Work

There exists a large body of sound and incomplete techniques for verifying parameterized systems. Solutions proposed in [3,14] reduce the problem of parameterized system verification to verification of a corresponding property-preserving finite-state abstraction, where instead of the state of each process, constraints on the number of processes at a each state are considered. Several other techniques rely on smart representation of the behavior of parameterized systems using regular grammars [4], petri-nets and graph-grammars [5]. Another class of techniques [9,10,15,16] involves identifying the invariant of a parameterized system. The invariant captures the common behavior exhibited

by all instances of the parameterized system. A property is satisfied by the parameterized system if the invariant conforms to the property. While techniques proposed in [10] apply induction to generate such invariant for singly-parameterized systems, [16] employ context-free grammars to generate the invariants for multi-parameterized systems. Most of these techniques require user guidance to obtain the grammars and/or appropriate abstraction mapping [17].

Emerson and Kahlon [11] were the first to develop a verification technique for multi-parameterized systems based on computing a cut-off. They propose solutions in the context of resource allocation systems where each homogeneous process has a specific behavior (zero or more internal transitions followed by *acquire* followed by zero or more internal transitions followed by *release*). They provide efficient methods for obtaining cut-offs when the system under consideration has a ring communication topology and the properties being considered are over adjacent processes (one process relaying a token to another).

Sun *et al.* [18] show that appropriate counter abstraction can be used to deal with state-space explosion without compromising fairness in model checking. The technique has been further applied in the context of parameterized systems by considering some pre-specified cut-off of the parameter, and any counter valuations greater than the cut-off are abstracted in the abstract model. Note that the cut-off valuation is not computed based on the model and/or the property under consideration; instead cut-off valuation is selected by the user.

Unlike these existing techniques, our technique does not rely on smart representations and/or abstractions that may require user-guidance. Our technique is fully automatic, applicable to any communication topology and is not developed in the context of any specific application domain (e.g., resource allocation).

3 Multi-parameterized System

Illustrative Example. The terminology used in this paper and the salient aspects of the proposed technique are explained using a variant of the Dining Philosophers protocol [19] (a model illustrating a classic multi-process synchronization problem). We use a variant of this protocol referred to as the Right-Left Dining Philosophers (RLDP) algorithm [11], where there are two types of philosophers: "Left" philosophers grab the left fork first and "Right" philosophers grab the right fork first. In this protocol, adjacent philosophers are of different types; therefore, the number of "Left" and "Right" philosophers is equal. Our technique is based on the notion of *behavioral automata* introduced in [12].

3.1 Processes as Behavioral Automata

Definition 1 (Behavioral Automaton). *A behavioral automaton A is a tuple (q_I, q_F, Δ, E), where q_I is the initial state, q_F is the final state, $\Delta \subseteq \{E \times \{q_I\}\} \cup \{\{q_F\} \times E\} \cup \{(q_I, q_F)\}$ is the transition relation, and E is a nonempty set of events (including the empty event ϵ). We write $q_I \rightarrow q_F$ if $(q_I, q_F) \in \Delta$, $\bullet \xrightarrow{e} q_I$ if $(e, q_I) \in \Delta$ and $q_F \xrightarrow{e} \bullet$ if $(q_F, e) \in \Delta$.*

```
1  # This diner type picks up left fork first
2  process left-diner {
3   L-START:   [init, epsilon] ->[neating, begin]
4   L-ASKL:    [neating, begin]->[waitl, askl2]
5   L-REASKL:  [waitl, ltaken] ->[waitl, askl2]
6   L-FREEL-NE:[neating, askl]->[neating, lfree2]
7   L-FREEL-WL:[waitl, askl]  ->[waitl, lfree2]
8   L-BUSYL-WR:[waitr, askl]  ->[waitr, ltaken2]
9   L-BUSYL-EAT:[eat, askl]   ->[eat, ltaken2]
10  L-ASKR:     [waitl, lfree] ->[waitr, askr2]
11  L-REASKR:  [waitr, rtaken]->[waitr, askr2]
12  L-FREER-NE:[neating, askr]->[neating, rfree2]
13  L-FREER-WL: [waitl, askr] ->[ waitl, rfree2]
14  L-BUSYR-WR: [waitr, askr] ->[waitr, rtaken2]
15  L-BUSYR-EAT:[eat, askr]   ->[eat, rtaken2]
16  L-EAT: [waitr, rfree]     ->[eat, rel-forks]
17  L-EAT-DONE:[eat, rel-forks]->[neating, begin]
18 }
```

(a)

```
1  # This diner type picks up right fork first
2  process right-diner {
3   R-START: [init, epsilon]   ->[neating, begin2]
4   R-ASKR:  [neating, begin2] ->[waitr, askr]
5   R-REASKR:  [waitr, rtaken2]->[waitr, askr]
6   R-FREER-NE:[neating, askr2]->[neating, rfree]
7   R-FREER-WR:[waitr, askr2]  ->[waitr, rfree]
8   R-FREER-WL:[waitl, askr2]  ->[waitl, rfree]
9   R-BUSYR-EAT:[eat, askr2]   ->[eat, rtaken]
10  R-ASKL:  [waitr, rfree2]   ->[waitl, askl]
11  R-REASKL:[waitl, ltaken2]  ->[waitl, askl]
12  R-FREEL-NE:[neating, askl2]->[neating, lfree]
13  R-BUSYL-WL: [waitl, askl2] ->[waitl, ltaken]
14  R-BUSYL-WR: [waitr, askl2] ->[waitr, ltaken]
15  R-BUSYL-EAT:[eat, askl2]   ->[eat, ltaken]
16  R-EAT: [waitl, lfree2]     ->[eat, rel-forks2]
17  R-EAT-DONE:[eat, rel-forks2]->[neating, begin2]
18 }
```

(b)

Fig. 1. Behavioral Automata for (a) "Left" Philosophers (b) "Right" Philosophers

Figures 1(a), (b) display the behavioral automata for philosophers of both types "Left" and "Right" of the RLDP protocol respectively. The statement of the form `A: [q,e]->[q',e']` denotes an automaton with $\bullet \xrightarrow{e} q$, $q \xrightarrow{\epsilon} q'$ and $q' \xrightarrow{e'} \bullet$.

A behavioral automaton describes the state in which a process can be, and what action it can perform when it is in that state. Automaton `L-ASKL` in Figure 1(a) (Line 4) presents the behavior of a philosopher of type "Left" who, while not eating (i.e. state `neating`), receives event `begin`, changes its state to `waitl` (i.e. waiting for the left fork) and sends the request for the left fork (event `askl2`). Since neighbor philosophers are of different types, the request of the left fork requested by a "Left" philosopher is received by a philosopher of type "Right". Automaton `R-FREEL-NE` in Figure 1(b) (Line 12) models the behavior of a "Right" philosopher who receives the request for the left fork while not eating, and replies that the fork is free (event `lfree`) so that the neighbor can take it.

An automaton with ϵ input event captures the behavior of a process where, if the process is at the initial state of this automaton, it can make a move without any external stimuli. For instance, automaton `L-START` in Figure 1(a) (Line 3) states that if a philosopher is in state `init`, she can generate the event `begin` without any input events. She changes her state to `neating` after this action.

Definition 2 (Process and System Specification). *A process specification for some type p, denoted by Prot_p, is a set of behavioral automata that represents the possible actions of a process of that type. A system specification, Prot, is the union of the process specifications for all types present in the system. At least one automaton in at least one process specification in Prot must have a transition of the form $\bullet \xrightarrow{\epsilon} q$, which represents an action without input event.*

In our example, there are two process specifications; one for the "Left" philosophers Prot_l defined by the automata in Figure 1(a), and the other for the "Right" philosophers Prot_r defined by the automata in Figure 1(b), where the types "Left" and "Right" are represented by the letters l and r respectively. Both types can initiate the protocol since the specification of each one contains an automaton with transition of the form $\bullet \xrightarrow{\epsilon} q$.

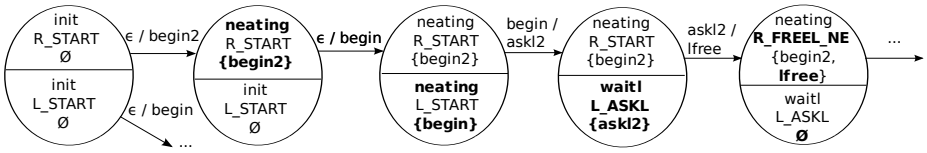

Fig. 2. Part of $sys(1_r, 1_l)$ for the RLDP Protocol

3.2 Behavior of Multi-parameterized System

Any system behavior is constrained by the topology that describes which processes in the system can directly communicate with each other.

Definition 3 (Communication Topology). *Given a system protocol specification* $\mathit{Prot} = \bigcup_{1 \le p \le t} \mathit{Prot}_p$*, where* t *is the number of different types of processes and* $\mathit{Prot}_p = \{A_{1_p}, \ldots, A_{l_p}\}$*, a topology is a set of tuples,* $\mathit{Topo} \subseteq E \times (\mathcal{I} \times \mathcal{T}) \times (\mathcal{I} \times \mathcal{T})$*, where* $E = \bigcup_{1 \le p \le t} \bigcup_{1 \le r \le l_p} \{E_r : E_r \text{ is set of events in } A_{r_p} \in \mathit{Prot}_p\}$*,* $\mathcal{I} \in \mathbb{N}$ *is the domain of number of processes of any type, and* $\mathcal{T}$ *is the domain of types. A tuple* $(e, i_{p1}, j_{p2}) \in \mathit{Topo}$ *implies that output* e *from* i*-th process of type* p_1 *is consumed by the* j*-th process of type* p_2*.*

For our example, we enforce two such constraints on communication patterns: first that adjacent philosophers are of different types (therefore there is an equal number of "Left" and "Right" philosophers), and second that it is a ring topology. For instance, the topology for the system instance containing one "Left" and one "Right" philosophers is $\mathsf{Topo} = \{(\mathtt{begin}, 1_l, 1_l), (\mathtt{begin2}, 1_r, 1_r),\ (\mathtt{askl2}, 1_l, 1_r),\ (\mathtt{lfree}, 1_r, 1_l), \ldots\}$.

Definition 4 (Multi-Parameterized System). *Given a specification* Prot *with* t *different types of processes, a multi-parameterized system containing* n_p *number of processes of type* p *(*$p \in [1, t]$*) is defined as* $sys(\bar{n}_t) = (S, S_I, T, \mathit{Topo})$*, where* $\bar{n}_t := n_1, n_2, \ldots, n_t$*,* S *is the set of states,* $S_I \subseteq S$ *is the set of initial states and* $T \subseteq S \times E \times E \times S$ *is the transition relation. A state in* S *contains* $\sum_{p=1}^{t} n_p$ *tuples of the form* $(q_{i_p}, A_{i_p}, \mathcal{E}_{i_p})$*; the tuple represents the configuration of the* i*-th process of type* p *such that* q_{i_p} *is the state of the process in the behavioral automata* A_{i_p} *and* $\mathcal{E}_{i_p}$ *denotes the set of output events from the process that have not been consumed yet.*

We use $s \xrightarrow{e/e'} s'$ *to denote* $(s, e, e', s') \in T$*.*

1. *A transition of the form* $\langle (q_{i_p}, A_{i_p}, \mathcal{E}_{i_p}), C \rangle \xrightarrow{\epsilon/e} \langle (q'_{i_p}, A_{i_p}, \mathcal{E}'_{i_p}), C \rangle \in T$*, if*
   $\{\bullet \xrightarrow{\epsilon} q_{i_p}, q_{i_p} \rightarrow q'_{i_p}, q'_{i_p} \xrightarrow{e} \bullet\} = \Delta \in A_{i_p} \ \wedge\ \mathcal{E}'_{i_p} = \mathcal{E}_{i_p} \cup \{e\}$.
   In the above C *represents the configurations of the remaining processes in the state.*
2. *A transition of the form* $\left\langle \begin{matrix} (q_{i_{p1}}, A_{i_{p1}}, \mathcal{E}_{i_{p1}}) \\ (q_{j_{p2}}, A_{j_{p2}}, \mathcal{E}_{j_{p2}}) \\ C \end{matrix} \right\rangle \xrightarrow{e/e'} \left\langle \begin{matrix} (q'_{i_{p1}}, A'_{i_{p1}}, \mathcal{E}'_{i_{p1}}) \\ (q_{j_{p2}}, A_{j_{p2}}, \mathcal{E}'_{j_{p2}}) \\ C \end{matrix} \right\rangle \in T$*, if*

$$\{\bullet \xrightarrow{e} q_{i_{p1}}, q_{i_{p1}} \rightarrow q'_{i_{p1}}, q'_{i_{p1}} \xrightarrow{e'} \bullet\} = \Delta \in A_{i_{p1}} \wedge$$
$$\mathcal{E}'_{i_{p1}} = \mathcal{E}_{i_{p1}} \cup \{e'\} \ \wedge\ \mathcal{E}_{j_{p2}} = \mathcal{E}'_{j_{p2}} \cup \{e\} \ \wedge\ (e, j_{p2}, i_{p1}) \in \mathit{Topo}$$

Figure 2 shows part of the system instance with one philosopher of type "Right" and one of type "Left", $sys(1_r, 1_l)$. Each state (system configuration) contains two process configurations for processes 1_l, 1_r, respectively. Two possible transitions (see Rule 1 in Definition 4) can happen from the initial configuration: the transition on ϵ/`begin2` belongs to move done by philosopher 1_r and the transition on ϵ/`begin` belongs to the one by philosopher 1_l. As these moves require no external stimuli (no input event), we call these moves *autonomous moves*. The second state in the figure shows the effect of the autonomous move done by philosopher 1_r on her configuration, where her state changes and her set of output events has the produced event. The transition on `begin/askl2` in the figure illustrates a non-autonomous move (see Rule 2 in Definition 4) with intra-process communication, where the philosopher 1_l consumes the event `begin` that she has produced from her own previous autonomous move, and produces the event `askl2` as a result (to ask the left fork). In the figure, the last transition on `askl2/lfree` illustrates a non-autonomous move with inter-process communication, where the request `ask12` for the right fork produced by philosopher 1_l is received by philosopher 1_r (according to Topo). Philosopher 1_r tells her neighbor that she can take the fork by sending the event `lfree`.

4 Cut-off Computation for Multiple Parameters

In this section, we describe our technique for computing the cut-off value for a multi-parameterized system. Given the specification for all process types in the system as behavioral automata and the topology as input, our technique consists of two steps. First, it computes the maximal behavior a process of each type can induce when it autonomously produces an event to be consumed by the environment. Second, it finds a multi-parameterized system instance whose behavior exhibits all the maximal behaviors that can be induced by processes of different types (if such an instance exists). We prove that the size of this system instance is the cut-off for the multi-parameterized system. We proceed with the computation of the maximal behavior induced by a process.

4.1 Maximal Behavior Induced by a Process

Intuitively, the maximal behavior of a system induced by a process of type p is all possible sequences of input/output events that can be caused by an autonomous move done by the process. We will use π (with appropriate subscripts) to denote sequence of input/output events.

Definition 5 (Maximal Behavior induced by type p process). *Given a multi-parameterized system $sys(\bar{k}_t)$, the maximal behavior induced by a process of type $p \in [1, t]$, denoted by $\textsc{Max}_p(sys(\bar{k}_t))$, is*

$$\textsc{Max}_p(sys(\bar{k}_t)) = \{\pi_p \mid \forall i \geq 0 : \pi_p[i] = \pi[h_p^{\pi}(i)] \wedge \eta_0 \in S_0 \wedge \forall j \geq 0 : \eta_j \overset{\pi[j]}{\rightarrow} \eta_{j+1}\}$$

In the above, $h_p^{\pi}(i) = k$ such that

1. $\pi[k] = \epsilon/e$, $\eta_k \overset{\pi[k]}{\rightarrow} \eta_{k+1}$ *is an autonomous move of type* p *process and* $\forall j \in [h_p^\pi(0), h_p^\pi(i-1)] : \pi[h_p^\pi(j)] \neq \epsilon/e'$.
2. $\pi[k] = e_1/e_2$, $\pi[h_p^\pi(i-1)] = e/e_1$ *and* $\forall j \in [h_p^\pi(i-1), k-1] : \pi[j] \neq e_1/e'$.

For instance, for the system $sys(1_r, 1_l)$ displayed in Figure 2, the set of maximal behavior that can be induced by a philosopher of type "Left", denoted as $\textsc{Max}_l(sys(1_r, 1_l))$, is composed of all possible sequences of input/output events that can occur as a result of a "Left" philosopher that autonomously makes a move. Let $\pi_l \in \textsc{Max}_l(sys(1_r, 1_l))$. The first input/output event in such a sequence π_l belongs to a "Left" philosopher making an autonomous move to initiate the protocol by sending event `begin` (i.e. $\pi_l[0] =$ `ϵ/begin`). The second input/output event of is of the same philosopher receiving this event and sending the request for left fork (i.e. $\pi_l[1] =$ `begin/askl2`). The third input/output event belongs to a philosopher of type "Right" that responds to the request of the left fork (i.e. $\pi_l[2] =$ `askl2/lfree`), and so on. Since the event `ϵ/begin2` which belongs to the move done by a "Right" philosopher is not induced by the autonomous move of the "Left" philosopher, this event does not belong to any sequence in $\textsc{Max}_l(sys(1_r, 1_l))$.

Computing the Induced Maximal Behavior. The computation proceeds by *chaining* of output from one behavior automata (present in the system specification) with the input (having the same name as the output) to another behavioral automata. For computing sequences in $\textsc{Max}_p(sys(\bar{k}_t))$, the first automata used in this chaining contains the initial state of the process of type p from where the process can make an autonomous move. We refer the result of such chaining as 1E_p and we show that 1E_p includes all possible behavior induced by the process of type p.

Definition 6 (1E_p). *Given a specification* Prot $= \{A_1, A_2, \ldots, A_m\}$ *with* t *different types of processes,* 1E_p *of the process type* p *is defined as a tuple* $(Q_{1\text{E}_p}, Q_{I1\text{E}_p}, \Delta_{1\text{E}_p})$, *where* $Q_{1\text{E}_p} = \{(q_I, A), (q_F, A) \mid A = (q_I, q_F, \Delta, E)\}$, $Q_{I1\text{E}_p} = \{(q_I, A) \mid A = (q_I, q_F, \Delta, E) \wedge \bullet \overset{\epsilon}{\rightarrow} q_I \in \Delta\}$, *and*

$$\Delta_{1\text{E}_p} = \left\{ (q_I, A) \overset{e_1/e_2}{\rightarrow} (q_F, A) \mid A = (q_I, q_F, \Delta, \{e_1, e_2\}), \{\bullet \overset{e_1}{\rightarrow} q_I, q_2 \overset{e_2}{\rightarrow} \bullet\} \subseteq \Delta \right\}$$
$$\bigcup \left\{ \begin{array}{r} (q_F, A) \overset{\tau}{\rightarrow} (q_I', A') \mid A = (q_I, q_F, \Delta, E), A' = (q_I', q_F', \Delta', E'), \\ q_F \overset{e}{\rightarrow} \bullet \in \Delta, \bullet \overset{e}{\rightarrow} q_I' \in \Delta' \end{array} \right\}$$

Figure 3 presents a partial view of 1E_l for the "Left" philosopher processes. Automaton `L-START` is chained to automaton `L-ASKL` as their corresponding output and input events match (`begin`). Similarly, automaton `L-ASKL` is chained with automata `R-FREEL-NE` and `R-BUSYL-WR` (defined in Figure 1(b), Lines 6 and 7 respectively) due to matching output and input events (`askl2`).

Note that not all the behavioral automata of the philosophers of type "Left" in Figure 1(a) will be included in 1E_l. For instance, automaton `L-FREEL-NE` (Figure 1(a), Line 12) models the behavior of the philosopher of type "Left" that replies to a request for the left fork that comes from its neighbor (i.e., a "Right" philosopher). This

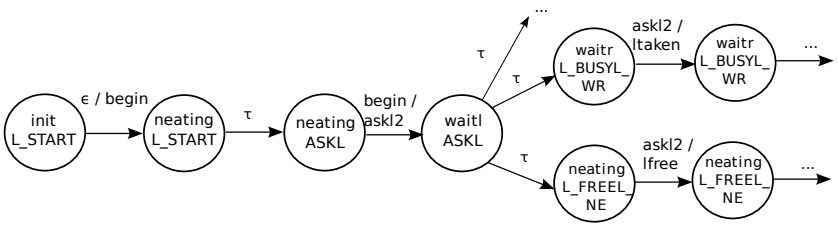

Fig. 3. Part of 1E_l

automaton will be present in the automata chain used in the construction of 1E_r (all possible behavior induced by the autonomous output from a "Right" philosopher).

We prove that every sequence in $\text{MAX}_p(sys(\bar{k}_t))$ is present as a path in 1E_p. Note that, paths in 1E_p contains τs obtained due to chaining of automata (over same output-input event pairs). We discard these events as they are connectors between the automata that do not contribute to any action. Given a sequence of events (say π) obtained from a path in 1E_p, the corresponding sequence $\pi_{-\tau}$ is obtained by removing τ events from π as follows: $\forall i \geq 0$,

$$\pi_{-\tau}[i] = \pi[g(i)] \text{ where } g(i) = \begin{cases} 0 \text{ if } i < 0 \\ k \text{ otherwise; } g(i-1) \leq j < k : \pi[j] = \tau \wedge \pi[k] \neq \tau \end{cases} \quad (1)$$

Proceeding further, we define the set of sequences of input/output events in $1\text{E}_p = (Q_{1\text{E}_p}, Q_{I1\text{E}_p}, \Delta_{1\text{E}_p})$ as

$$\text{PATH}(1\text{E}_p) = \{\pi_{-\tau} \mid \zeta_0 = (q, A_x) \in Q_{I1\text{E}_p} \ \wedge \ \forall i \geq 0 : \zeta_i \overset{\pi[i]}{\rightarrow} \zeta_{i+1} \in \Delta_{Q_{1\text{E}_p}}\}$$

Theorem 1. *Given a protocol specification* Prot *with* t *different types of processes,* $\forall \bar{k}_t, \forall p \in [1,t] : \text{MAX}_p(sys(\bar{k}_t)) \subseteq \text{PATH}(1\text{E}_p)$.

For ease of explanation and brevity of the proof, we introduce the following functions.

$$F_1(\pi, \Pi) = \{\pi' \mid \pi' \in \Pi \wedge \pi' \sqsubset \pi \wedge \ \nexists \pi'' \in \Pi : (\pi' \sqsubset \pi'' \sqsubset \pi) \vee (\pi'' = \pi)\} \quad (2)$$

where $\sqsubset$ denotes the strict substring relationship, i.e., $\pi' \sqsubset \pi$ implies π' is a substring of π and $\pi' \neq \pi$. The above function computes a set of substrings π' of π such that there are no other substrings of π in Π that are longer than the elements in the resultant set. We define the following function over sequences of events.

$$F_2(\pi, \pi') = e_1/e_0 \text{ such that } \pi' \sqsubset \pi \ \wedge \ \pi[|\pi'|] = e_1/e_0 \quad (3)$$

The above function identifies the event on which the sequence π diverges from π'.

Proof. Assume that $\exists \bar{k}_t, \exists p \in [1,t] : \text{MAX}_p(sys(\bar{k}_t)) \not\subseteq \text{PATH}(1\text{E}_p)$. In other words, there exists a π such that $\pi \in \text{MAX}_p(sys(\bar{k}_t))$ and $\pi \notin \text{PATH}(1\text{E}_p)$. From Equations 2 and 3, $F_1(\pi, \text{PATH}(1\text{E}_p)) = \chi$ and $\forall \pi' \in \chi : \exists e_1/e_0 : F_2(\pi, \pi') = e_1/e_0$.

There are two possible cases.

Case 1. $e_1 = \epsilon$. According Definition 5, the event e_1/e_0 is an an autonomous move of a process of type p in $sys(\bar{k}_t)$. Since the first transition of 1E_p models with an autonomous move of type p process (Definition 6), this case is not possible, i.e., our assumption that $\pi \notin \text{PATH}(1\text{E}_p)$ is false.

Case 2. $e_1 \neq \epsilon$. The event e_1/e_0 must be preceded by an input/output event of the form e_2/e_1 in order to allow for the event to happen in the first place (Definition 5). I.e., if $\pi[i] = e_1/e_0$, $\pi[i-1] = e_2/e_1$. From Equation 3, $F_2(\pi, \pi') = e_1/e_0$ and therefore, $\pi'[i-1] = e_2/e_1$ and $\pi'[i] \neq e_1/e_0$. In order for this to be possible, we need to conclude that there exists no behavioral automata that can consume e_1 and produce e_0, as construction of 1E_p proceeds by chaining the output (e_1 in this case) of one automata with the matching input of another. If no automata can take as input e_1 and produce e_0, then it is not possible to have any sequence π in $\text{MAX}_p(sys(\bar{k}_t))$ that has $\pi[i-1] = e_2/e_1$ and $\pi[i] = e_1/e_0$. This contradicts our assumption that $\pi \in \text{MAX}_p(sys(\bar{k}_t))$.

4.2 Finding the Cut-Off Value

The cut-off of parameter values for a parameterized system is such that the instance of the parameterized system at the cut-off (cut-off instance) satisfies a property if and only if all instances of the parameterized system larger than the cut-off instance satisfies the same property. We will consider two types of properties in the logic of $\text{LTL}\setminus\text{X}$:

- ◇ TYPE I PROPERTY: Property that involves the states of exactly one process. For example, if a philosopher tries to pick the left fork, she is eventually in a state where she can eat.
- ◇ TYPE II PROPERTY: Property involving two adjacent processes that directly communicate via input/output events. For example, two adjacent philosophers do not eat at the same point of time.

We will use the standard notation $[\![\varphi]\!]$ to denote the semantics of an $\text{LTL}\setminus\text{X}$ property φ; it represents the set of sequence of states that satisfy φ. A system sys satisfies φ, denoted by $sys \models \varphi$, if and only if all paths starting from all start states of the system result in a set of sequence of states such that this set is a subset of $[\![\varphi]\!]$. For details of semantics of LTL, please refer to [13].

Definition 7 (Cut-off). *Given a protocol specification* Prot *for t different types of processes and a topology* Topo*, for any $\text{LTL}\setminus X$ properties of Type I and Type II, denoted by φ, $\bar{k}_t := k_1, k_2, \ldots, k_t$ is said to be cut-off if and only if the following holds: $sys(\bar{k}_t) \models \varphi \Leftrightarrow \forall \bar{n}_t \geq \bar{k}_t : sys(\bar{n}_t) \models \varphi$ where $\bar{n}_t \geq \bar{k}_t \Leftarrow \forall p \in [1,t] : n_p \geq k_p$.*

To automatically identify the cut-off, we iteratively compute specific instances of the system under consideration and compute all possible sequences of input/output events in the system-instance. Such a set of sequence in a system-instance $sys(\bar{k}_t) = (S, S_I, T, \text{Topo})$ is defined as $\text{PATH}(sys(\bar{k}_t), S) = \{\pi_{-\tau} \mid \eta_0 = s \in S \wedge \forall i \geq 0 : \eta_i \overset{\pi[i]}{\rightarrow} \eta_{i+1} \in T\}$. We prove that $\bar{k}_t$ is the cut-off if $\forall p \in [1,t] : \text{PATH}(1\text{E}_p) \subseteq \text{PATH}(sys(\bar{k}_t), S^{\bar{k}_t})$ where $S^{\bar{k}_t}$ denotes the set of states in $sys(\bar{k}_t)$. Procedure CutOff presents our automatic method for obtaining the cut-off.

Procedure CutOff (Prot, t, Topo, initial system config)
Construct initial $sys(\bar{k}_t)$ from initial system config and Topo
for all $p \in [1, t]$ Compute 1E_p from Prot **do**
while $\text{PATH}(1\text{E}_p) \not\subseteq \text{PATH}(sys(\bar{k}_t), S^{\bar{k}_t})$ Increase $\bar{k}_t$ in a breadth-first manner
end while
end for
return $\bar{k}_t$;

5 Proof of Soundness

We proceed by introducing definitions and propositions that will be used to prove the soundness of Procedure CutOff.

Definition 8 (Projection on processes). *Given a multi-parameterized system* $sys(\bar{k}_t) = (S, S_I, T,$ *Topo*$)$ *with* t *different types of processes and a set* $R \subseteq \{i_p \mid i \in [1, k_p] \ \wedge \ p \in [1, t]\}$*, the projected behavior w.r.t.* R *is denoted by* $sys(\bar{k}_t)\downarrow R = (S, S_I, T\downarrow R,$ *Topo*$)$*, such that labels of all transitions that do not directly involve moves of process* $i'_{p'} \in R$ *are renamed to* τ*. We will use* $\pi\downarrow R$ *to denote projection of a sequence of events on* R*.*

For the example of the RLDP protocol, in the projected system $sys(1_r, 1_l)\downarrow\{1_l\}$, both the transitions labeled as $\epsilon/\texttt{begin}$ and the transition labeled as $\texttt{begin/askl2}$ remain the same while all other transitions in the Figure 2 are substituted with τ transitions.

Proposition 1. *For any multi-parameterized system* $sys(\bar{k}_t)$ *with* t *different types of processes, the following holds for all properties* φ *(in the logic of* $\texttt{LTL}\setminus \texttt{X}$*) defined over states of processes whose indices belong to* $R = \{i_p \mid i \in [1, k_p] \wedge p \in [1, t]\}$*:* $sys(\bar{k}_t) \models \varphi \ \Leftrightarrow \ sys(\bar{k}_t)\downarrow R \models \varphi$*.*

Proposition 2. *Let* Φ *be the set of all properties (in the logic of* $\texttt{LTL}\setminus \texttt{X}$*) defined over states of processes whose indices belong to* $R = \{i_p \mid i \in [1, k_p] \wedge p \in [1, t]\}$*. The following holds for any two instances of multi-parameterized systems,* $sys(\bar{k}_t)$ *and* $sys(\bar{k}'_t)$*.*

$$\forall \varphi \in \Phi : \big(sys(\bar{k}_t) \models \varphi \ \Leftrightarrow \ sys(\bar{k}'_t) \models \varphi\big) \Rightarrow \\ \text{PATH}(sys(\bar{k}_t)\downarrow R, S_I^{\bar{k}_t}) = \text{PATH}(sys(\bar{k}'_t)\downarrow R, S_I^{\bar{k}'_t})$$

In the above, $S_I^{\bar{k}_t}$ *and* $S_I^{\bar{k}'_t}$ *are the initial state-sets of* $sys(\bar{k}_t)$ *and* $sys(\bar{k}'_t)$*, respectively.*

Proof. From Proposition 1, we conclude

$$\forall \varphi \in \Phi : \big(sys(\bar{k}_t) \models \varphi \ \Leftrightarrow \ sys(\bar{k}'_t) \models \varphi\big) \Rightarrow \big(sys(\bar{k}_t)\downarrow R \models \varphi \ \Leftrightarrow \ sys(\bar{k}'_t)\downarrow R \models \varphi\big)$$

If π denotes a path in a system over sequence of input/output actions, we denote the corresponding sequence of states in the path by $\texttt{seq}(\pi)$. Therefore,

$$\forall \varphi \in \Phi : \big(sys(\bar{k}_t)\downarrow R \models \varphi \;\Leftrightarrow\; sys(\bar{k}'_t)\downarrow R \models \varphi\big) \;\Rightarrow$$
$$\forall \pi \in \textsc{Path}(sys(\bar{k}_t)\downarrow R, S_I^{\bar{k}_t}) : \exists \pi' \in \textsc{Path}(sys(\bar{k}'_t)\downarrow R, S_I^{\bar{k}'_t}) : \mathtt{seq}(\pi) = \mathtt{seq}(\pi')$$
$$\bigwedge$$
$$\forall \pi' \in \textsc{Path}(sys(\bar{k}'_t)\downarrow R, S_I^{\bar{k}'_t}) : \exists \pi \in \textsc{Path}(sys(\bar{k}_t)\downarrow R, S_I^{\bar{k}_t}) : \mathtt{seq}(\pi') = \mathtt{seq}(\pi)$$
$$\Rightarrow\; \textsc{Path}(sys(\bar{k}_t)\downarrow R, S_I^{\bar{k}_t}) = \textsc{Path}(sys(\bar{k}'_t)\downarrow R, S_I^{\bar{k}'_t})$$

Proposition 3. *For any parameterized system with t types of processes,*

$$\forall \bar{n}_t \geq \bar{k}_t : \textsc{Path}(sys(\bar{k}_t), S_I^{\bar{k}_t}) \subseteq \textsc{Path}(sys(\bar{n}_t), S_I^{\bar{n}_t})$$
$$\forall p \in [1,t] : \textsc{Path}(1\mathrm{E}_p) \subseteq \textsc{Path}(sys(\bar{k}_t), S^{\bar{k}_t}) \Rightarrow\; \textsc{Path}(1\mathrm{E}_p) \subseteq \textsc{Path}(sys(\bar{n}_t), S^{\bar{n}_t})$$

where $S^{\bar{k}_t}$, $S_I^{\bar{k}_t}$ and $S^{\bar{n}_t}$, $S_I^{\bar{n}_t}$ are the sets of states and initial states of $sys(\bar{k}_t)$ and $sys(\bar{n}_t)$ respectively.

Theorem 2. *Given a parameterized system with t different types of processes each defined using a set of behavioral automata* Prot*, the following holds for all Type I and II properties φ in the logic of $\mathtt{LTL}\backslash \mathtt{X}$*

$$\forall p \in [1,t] : \textsc{Path}(1\mathrm{E}_p) \subseteq \textsc{Path}(sys(\bar{k}_t), S^{\bar{k}_t}) \Rightarrow\; \big(sys(\bar{k}_t) \models \varphi \Leftrightarrow sys(\bar{n}_t) \models \varphi\big)$$

where $\bar{n}_t = n_1, n_2, \ldots, n_t$, $\bar{k}_t = k_1, k_2, \ldots, k_t$, $S^{\bar{k}_t}$ is the set of states in $sys(\bar{k}_t)$, and $S_I^{\bar{k}_t}$ and $S_I^{\bar{n}_t}$ are initial state-sets of $sys(\bar{k}_t)$ and $sys(\bar{n}_t)$ respectively.

Proof. Due to space constraints, we provide a proof sketch for the theorem. The full proof is available at `http://www.cs.iastate.edu/~slede/golok/`.

Using Propositions 1, 2 and 3, it is required to prove that $\forall p \in [1,t], \forall i \leq n_p, \exists j \leq k_p$, $\textsc{Path}(sys(\bar{n}_t)\downarrow\{i_p\}, S_I^{\bar{n}_t}) = \textsc{Path}(sys(\bar{k}_t)\downarrow\{j_p\}, S_I^{\bar{k}_t})$.

Assume that there exists a sequence π in $\textsc{Path}(sys(\bar{n}_t)\downarrow\{i_p\}, S_I^{\bar{n}_t})$ that is not present in $\textsc{Path}(sys(\bar{k}_t)\downarrow\{j_p\}, S_I^{\bar{k}_t})$. This implies that for every path π' in $\textsc{Path}(sys(\bar{k}_t)\downarrow\{j_p\}, S_I^{\bar{k}_t})$, there exists an input/output event (e_1/e_0) such that e_1/e_0 is present in π and absent in π'. If e_1 is equal to ϵ, then it can be immediately shown that $\textsc{Path}(sys(\bar{k}_t)\downarrow\{j_p\}, S_I^{\bar{k}_t}) \not\subseteq \textsc{Path}(1\mathrm{E}_p)$ as ϵ/e_0 is an autonomous move. This forms the base case of the proof (contradiction of our assumption above). If e_1 is not equal to ϵ then there must be some event e_2/e_1 preceding e_1/e_0 in the path π and such ordering of events is absent in π'. In this case, it can be shown that π' diverges from π on event e_2/e_1. The proof of the theorem (i.e., contradiction of our assumption) can be realized by proceeding inductively (on the length of the diverging point between π and π') and eventually reaching the base case.

Theorem 3 (Soundness). *If Procedure* CutOff *terminates, the return $\bar{k}_t$ is the cut-off as per the Definition 7.*

Proof. Follows from Theorem 2.

```
1  process left-diner { ... }
2  process right-diner { ... }
3
4  topology {
5
6   connectivity {
7    left-diner 0 -- right-diner 0
8   }
9
10  additionrule add-two {
11   create: left-diner x
12   create: right-diner y
13   require: right-diner z -- left-diner 0
14   remove: var z -- left-diner 0
15   add: var z -- var x
16   add: var x -- var y
17   add: var y -- left-diner 0
18  }
19  msgs {
20   (left-diner, begin, self)
21   (left-diner, askl, rpeer)
22   (right-diner, askr2, lpeer) ...
23  }
24 }
25
26 initialconfig {  }
```

Fig. 4. Input file for the RLDP protocol

6 Golok: A Tool to Find Cut-off

We have implemented our technique in a tool, *Golok*[1]. It is written in Scheme [20] in ∼4K lines of code. We now describe the input language to Golok using the RLDP example and describe the several optimizations we implemented in our tool.

6.1 Front End: Input Language of Golok

The input file containing the specification for the RLDP protocol is displayed in Figure 4. The specification has three main components: (a) the process specification, (b) the topology specification and (c) the initial configuration specification.

Process Specification. The process specifications (lines 1, 2) contain the behavioral automata for every process type as described in Section 3 (Figures 1(a) and (b)).

Topology Specification. The topology specification serves to restrict communication patterns between processes. It is defined using the keyword **`topology`** (lines 4 - 25) and is composed of three parts. The first part of the topology specification (lines 6-8) specifies the topology of the initial system instance. In RLDP, the initial system instance has one philosopher of each type (processes are zero-indexed). The second part of the topology specification (`additionrule` lines 10-18) is the addition rules that ensure that newly generated system instances follow the communication topology of the protocol. For RLDP, the addition rule `add-two` (lines 10-18) states that any new system instances will create two new philosophers of different types (lines 11-12) and that they linked to other processes to preserve that neighbors are of different types (lines 13-17). The final part of the topology specification (lines 19-23) is responsible for specifying the direction of the flow of events between processes, where every tuple (d, e, s) describes the event to be received e, the type of the recipient process d and the index of the sender process s. There are four choices for s: `self` (message sent and received by the same process), `rpeer` (message sent by the right neighbor of d) `lpeer` (message sent by the left neighbor of d) and `peer` (message sent by a neighbor of d)[2].

Initial Configuration Specification. The initial configuration is explicitly specified if any process needs to start from a different automaton other than the first automaton in its process specification.

[1] A cutting tool typically used in Indonesia and the Philippines.
[2] `lpeer`, `rpeer` used for ring topology, `peer` for other topologies.

6.2 System Instance Generator/Checker

The System Instance Generator/Checker (SIGC) is the main module of Golok. The goal of SIGC is to construct a system instance $sys(\bar{k}_t)$ (see Procedure CutOff) and check whether for all $p \in [1, t]$, $\text{PATH}(1\text{E}_p) \subseteq \text{PATH}(sys(\bar{k}_t), S^{\bar{k}_t})$. The main challenge in implementing SIGC is to reduce the computational cost involved in checking for path inclusion by considering all possible paths from all states. We describe several optimizations we have implemented in Golok to help reduce the computational cost.

Simulation-base cut-off computation. Simulation relation [21] identifies pairs of states in a transition system such that one element of the pair simulates all possible behavior (in terms of sequence and branching of transitions) of the other. It is a stronger relation than language inclusion. Furthermore, computing simulation relation is linear to the state-space of the transition system as opposed to computing language inclusion which is exponential to the state-space. As a result, it is computationally efficient to use simulation rather than language inclusion. The results of Theorem 3 still holds. Given a protocol specification Prot and a multi-parameterized system $sys(\bar{k}_t)$ with t different types of processes, a state r in $sys(\bar{k}_t)$ is said to simulate a state s in 1E_p, denoted as $s \prec r$, if the following holds:

$$\forall e/e', s' : s \xrightarrow{\tau^* e/e'} s' \in 1\text{E}_p \Rightarrow \exists r' : r \xrightarrow{\tau^* e/e'} r' \in sys(\bar{k}_t) \wedge s' \prec r'$$

In the above, $\tau^* e/e'$ represents zero or more τ transitions followed by an e/e' transition. We say that 1E_p is simulated by $sys(\bar{k}_t)$ if and only if there exists a state r in $sys(\bar{k}_t)$ such that for all start states s in 1E_p, $s \prec r$.

Simulation based cut-off computation may lead to additional challenges. For certain systems where there exists a cut-off that can be identified using path inclusion, a stronger requirement for cut-off based on simulation may fail to obtain such a cut-off. From our experimental results, we have realized that such a problem exists when in addition to parameterized components, the system also contains non-parameterized components (ones whose number is pre-specified and fixed). For instance, in the bounded-buffer protocol, there exists only one buffer for all instances of the systems. Similarly, for the singly-parameterized spin lock, there is only one object in any system instance. The problem of using simulation for such systems can be alleviated by projecting out any actions that result from the non-parameterized components.

Reducing the number of Simulation Checks. As the size of a system instance could be prohibitively large, performing a simulation check on every state to verify if it simulates 1E_p can still be expensive. To reduce the number of simulation checks, we construct the system instances on-the-fly (i.e. states are generated when needed), perform partial-order reduction ([22]) to ensure re-use of intermediate simulation checking results. Furthermore, for every system configuration s in $sys(\bar{k}_t)$, the following constant-time check is done before performing a simulation check. If the system configuration s does not have any process that is able to make an autonomous move (a move that does not require any external stimuli), this system configuration s is never expanded. The reason is that, since the first transition in any 1E_p must come from an autonomous move, then it is not possible that a system configuration s where no process is able to make an autonomous move is the configuration that simulates 1E_p for any type p.

Table 1. Experimental results of our tool *Golok* compared to existing techniques

Protocol	Topology	Process Types		Existing Work		Our Technique				
		# of types	# of Params	References	Known Cut-off	Computed Cut-off ($\bar{k}_t$)	Time (sec)	Explored States	States in $sys(\bar{k}_t)$	%gain
Dining Philosophers [11, 19]	Ring	1	1	[11]	4	**3**	0.54	33	510	93.53
		2	(r, l) 2	[11]	$2_r, 2_l$	$3_r, 3_l$	4.84	7,524	268,536	97.20
Bounded-Buffer [23]	Star	3	(p, c) 2	X†	X	$2_p, 1_c$	1.10	37	269	86.25
Spin Lock [24]	Star	2	(t) 1	[26]	3	$\mathbf{2}_t$	0.62	13	84	84.50
	Multi-star*	2	(t, o) 2	X	X	$2_t, 2_o$‡	0.52	15	243	93.80
DME [25]	Ring	1	(fc) 1	[27]	4	$\mathbf{2}_{fc}$	0.51	5	7	28.58
		2	(f, c) 2	X	X	$1_f, 1_c$	0.51	4	7	42.86

*Multi-star: All processes of different types are connected; †: To the best of our knowledge, no known results exist.
‡: Golok produced same cut-off value for different sizes of the buffer, displayed performance results are for the system with buffer of size 1.
r: right philosopher, l: left philosopher; p: producer, c: consumer; t: thread, o: object; fc: dme node, f: forward dme node, c: critical dme node.

7 Case Studies

Besides the RLDP protocol, we ran Golok on three other multi-parameterized systems with different communication topologies to validate our technique: the Bounded Buffer protocol [23], a variant of the Spin Lock protocol [24] and a variant of the Distributed Mutual Exclusion Protocol [25]. All examples along with the tool, Golok, are available at `http://www.cs.iastate.edu/~slede/golok/`. All experiments were run on a single core Pentium 4, 2.53 Ghz with 2 GB of RAM. Table 1 summarizes the experimental results. First four columns presents the parameterized systems and their various features: topology, number of process types in the system and number of types of the process that are parameterized. The rest of the table provides typical solutions obtained for some of the examples from the most relevant existing work and compared it with the results obtained from our tool, Golok. To the best of our knowledge, none of the existing techniques provide with a viable tool that can be used in practice. As a result, we only provide execution time information for our technique.

The table shows that while parameterized systems with different communication topology are handled by different techniques (developed primarily for the topology under consideration), our technique is applicable uniformly to all parameterized systems (both singly- and multi-parameterized) with different communication topologies. Note that, in some cases, Golok has identified a smaller cut-off value compared to the ones known in the existing work (shown in bold font). This can be attributed primarily to the fact that existing techniques for cut-off identification are independent of the system behavior (only topology dependent, e.g., [27]) or rely on abstractions that are sufficient but not necessary (e.g., [26]). As our technique is system dependent, Golok may compute different cut-off values for different systems with the same topology.

The table also shows impact of the optimizations in our technique. For instance, the last column shows the proportion of states that are not explored in the cut-off instance of the parameterized system while verifying that the instance simulates the 1E for all types of processes that are parameterized.

8 Summary and Conclusion

We have presented a technique for generating cut-off values for each of the parameters of a multi-parameterized system, proved its soundness, implemented the technique in a tool, and demonstrated its applicability to a number of canonical case studies. As Golok provides an automated realization of our method, the tool can be effectively used even for cases where the system is non-parameterized. For example, consider that the objective is to verify RLDP protocol with N pairs of "Left" and "Right" philosophers such that N is prohibitively large and as a result, standard model checking tools fail to provide any result due to state-space explosion. In such cases, a parameterized version of the system can be considered in Golok and if a cut-off is returned (e.g., $3_r, 3_l$ for RLDP protocol) then model checking the system instance with this cut-off is equivalent to model checking the much larger system instance contains N pairs of philosophers.

Future work includes extending the expressive power of behavioral automata representation, associated formalisms, techniques and Golok to allow for specification of parameterized system whose behavior is constrained by the valuations of messages being exchanged and to allow broadcast.

References

1. Manna, Z., Pnueli, A.: An exercise in the verification of multi-process programs. Beauty is our business: a birthday salute to Edsger W. Dijkstra, pp. 289–301 (1990)
2. Apt, K.R., Kozen, D.C.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. 22(6), 307–309 (1986)
3. Clarke, E.M., Talupur, M., Veith, H.: Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 33–47. Springer, Heidelberg (2008)
4. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000)
5. Baldan, P., Corradini, A., König, B.: A framework for the verification of infinite-state graph transformation systems. Inf. Comput. 206(7), 869–907 (2008)
6. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
7. Przydatek, B., Song, D., Perrig, A.: Sia: secure information aggregation in sensor networks. In: SenSys. (2003)
8. Byrd, G., Flynn, M.: Producer-consumer communication in distributed shared memory multi-processors. Proceedings of the IEEE 87(3), 456–466 (1999)
9. Marelly, R., Grumberg, O.: Gormel - grammar oriented model checker. Technical Report 697, The Technion (1992)
10. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.D.: Parameterized verification with automatically computed inductive assertions. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 221–234. Springer, Heidelberg (2001)
11. Emerson, E.A., Kahlon, V.: Model checking large-scale and parameterized resource allocation systems. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 251–265. Springer, Heidelberg (2002)
12. Hanna, Y., Basu, S., Rajan, H.: Behavioral automata composition for automatic topology independent verification of parameterized systems. In: ESEC/FSE 2009 (August 2009)

13. Emerson, E.A.: Temporal and modal logic, pp. 995–1072 (1990)
14. Yavuz-Kahveci, T., Bultan, T.: Verification of parameterized hierarchical state machines using action language verifier. In: MEMOCODE 2005, pp. 79–88 (2005)
15. Roychoudhury, A., Ramakrishnan, I.V.: Inductively verifying invariant properties of parameterized systems. Automated Software Engg. 11(2), 101–139 (2004)
16. Clarke, E.M., Grumberg, O., Jha, S.: Verifying parameterized networks using abstraction and regular languages. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 395–407. Springer, Heidelberg (1995)
17. Zuck, L.D., Pnueli, A.: Model checking and abstraction to the aid of parameterized systems (a survey). Computer Languages, Systems & Structures 30(3-4), 139–169 (2004)
18. Sun, J., Liu, Y., Roychoudhury, A., Liu, S., Dong, J.S.: Fair model checking with process counter abstraction. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 123–139. Springer, Heidelberg (2009)
19. Dijkstra, E.: Two starvation free solutions to a general exclusion problem. EWD 625, Plataanstraat 5, 5671 AL Neunen, The Netherlands
20. Abelson, H., et al.: Revised report on the algorithmic language scheme. Higher Order Symbol. Comput. 11(1), 7–105 (1998)
21. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1982)
22. Mazurkiewicz, A.W.: Basic notions of trace theory. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 285–363. Springer, Heidelberg (1989)
23. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating System Concepts. Wiley, Chichester (2004)
24. Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessors. IEEE Trans. Parallel Distrib. Syst. 1(1), 6–16 (1990)
25. Wolper, P., Lovinfosse, V.: Verifying properties of large sets of processes with network invariants. In: Workshop on Automatic Verification Methods for Finite State Systems, pp. 68–80 (1990)
26. Basu, S., Ramakrishnan, C.R.: Compositional analysis for verification of parameterized systems. Theor. Comput. Sci. 354(2), 211–229 (2006)
27. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: POPL, pp. 85–94 (1995)