A Cut-off Approach for Bounded Verification of Parameterized Systems

Qiusong Yang
qiuling_yang@itechs.iscas.ac.cn
Laboratory for Internet Software Technologies
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China

Mingshu Li
mingshu@iscas.ac.cn
2State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China

ABSTRACT
The features in multi-threaded programs, such as recursion, dynamic creation and communication, pose a great challenge to formal verification. A widely adopted strategy is to verify tentatively a system with a smaller size, by limiting the depth of recursion or the number of replicated processes, to find errors without ensuring the full correctness. The model checking of parameterized systems, a parametric infinite family of systems, is to decide if a property holds in every size instance. There has been a quest for finding cut-offs for the verification of parameterized systems. The basic idea is to find a cut-off on the number of replicated processes or on the maximum length of paths needed to prove a property, standing a chance of improving verification efficiency substantially if one can come up with small or modest cut-offs. In this paper, a novel approach, called Forward Bounded Reachability Analysis (FBRA), based upon the cut-off on the maximum length of paths needed is proposed for the verification of parameterized systems. Experimental results show that verification efficiency has been significantly improved as a result of the introduction of our new cut-offs.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—Model checking, Formal methods

General Terms
Verification, Algorithms

Keywords
Parameterized System, Bounded Model Checking, Cut-off

1. INTRODUCTION
In multi-threaded programs, such as those written in the Java and C programming languages, it may be necessary for numerous replicated processes (or threads) to be dynamically created during the program’s execution. This may be done, for example, to enable the concurrent manipulation of different requests or to support various kinds of background computations. The verification of such kinds of systems poses a great challenge because it is often difficult, if not impossible, to predict the precise number of replications that may be needed for any particular execution. This may cause the verification of such programs to be undecidable, especially when such features as concurrency, recursion, dynamic creation and communication are present.

A strategy that is often adopted is to verify another program that is derived from the original program, by limiting the depth of recursion or the number of replicated processes. While the verification of this derived program does not assure the verification of the original, it may find the errors in the derived program that then might be expected to also be present in the original program. Thus, for example, the Java program given in Fig. 1, is derived from the producer-consumer problem presented in [7] by the removal of the method modifier “synchronized” of the get and put methods. In [7], only \( k \) replications of a class \( C \), a limit provided by the analyst, are allowed to be created. This creates a bound on the state space and enables verification of the derived program. We note that even if this kind of reduced version of a program is decidable, the size of the state space needed for verification may still spiral out of control when the number of replicated processes grows. Of more concern, perhaps, is the possibility that some errors that depend on the number of replicated processes (e.g. those that arise from higher levels of concurrency) might not be detected when only derived versions having bounded numbers of replications are investigated. For example, a derived version of the producer-consumer system may not be sufficient to support detecting that new elements will still be put into the buffer even when it is full if the number of replicated processes is limited to less than the capacity of the buffer, \( BUF\_CAPACITY \), in the derived version.

Therefore there is considerable interest in determining whether it is possible to compute a cut-off on the number of process replications or on the maximum length of paths in a derived version that is sufficient to assure that a verification of the derived program also serves as a verification of the original program. In this paper, we explore this possibility, focusing on parameterized systems, a subset of concurrent systems that involve multiple heterogeneous classes of finite-state processes and depend on parameters that define the...
number of replications of each class’s process template [11].
An advantage of parameterized systems is that the goal of verification of such systems is to decide if a property holds for every system size. Thus, the verification of a property of a parameterized system will hold irrespective of the system size, and thus without any concern for determining a bound on the number of process replications. Unfortunately, the verification of general parameterized systems is undecidable [2]. On the other hand, verification of a class of commonly used systems consisting of many identical and finite-state processes is decidable [19, 17, 1, 15, 13, 14, 8, 9].

Problem Statement. There has been a quest for finding cut-offs, bounds on the number of replications or the maximum length of paths, that nevertheless assure the possibility of the verification of parameterized systems. The basic problem is to show that, for a given property $S$ and a parameterized system $P(n_1,n_2,\ldots,n_k) = C_1 \times \cdots \times C_i \times (U_1,U_2,\ldots,U_k)^{(n_1,n_2,\ldots,n_k)}$, denoting an infinite family of concurrent systems in which each system is composed of $C_i$ through $C_1$ and $n_1$ replications of $U_1$ through $n_k$ replications of $U_k$ running in parallel synchronously, there is a cut-off $C(P,S)$, which is invariant with respect to $n_1,n_2,\ldots,n_k$, on the number of replications of each process template or on the maximum length of paths needed to prove the property $S$. An advantage of cut-off approaches is that the verification of an infinite family of systems can be turned into a verification problem having a finite state space. If one can come up with a cut-off that is small or moderate in size, verification efficiency might be substantial.

Prior Work. The prior work on cut-offs can be classified into two categories: a) Cut-offs on the number of replications. Several pioneering attempts, such as those based on process closures [5], bisimulation [6], process invariants [20], and network invariants [22], have succeeded in reducing the verification of parameterized systems to the verification of a system with a finite number of processes. However, these approaches are semi-automatic and the closures or invariants needed to support these verifications have to be supplied by the analyst. More recently papers by [11], [4], and [3] have demonstrated the existence of cut-offs for various kinds of parameterized systems and specification logic pairs. Unfortunately, most cut-offs that have been demonstrated do not lead to efficient decision procedures or effective means for reductions, and small cut-offs have only been demonstrated for some relatively simple types of systems. b) Cut-offs on the maximum length of paths. A cut-off phrased in terms of the maximum length of paths would seem to offer computational complexity advantages, as a system whose bound is on the number of nodes in its longest path would seem to have a far smaller search space than a system whose bound is the same number of replicated processes, as the number of global states in the latter may grow exponentially. Moreover, note that in bounding the longest path, a cut-off on the number of replicated processes is inferred (by assuming that each node in a path is executed by a different process replication). In [17] and [12], the authors show that a cut-off exists for the verification of parameterized systems composed of a control process and an arbitrary number of user processes against indexed LTL properties. The proof of the existence of such cut-offs is useful in proving the decidability of the decision procedures for such systems. However this previous work has not shown how to compute the value of a cut-off, and thus is unable to help in the verification of actual systems of this kind.

Contributions. In this paper, an FBRA approach based upon the cut-off on the maximum length of paths is proposed for the verification of parameterized systems with only rendezvous actions against a property automaton, a finite state machine specifying a set of undesirable behavior. At the same time, the approach is sound and complete.

A Novel Approach to Determine Cut-offs: Given a parameterized system and a property automaton, at most $\exists i < \mathbb{R}$ increasingly refined abstractions, $\{A_0, A_1, \ldots, A_i, \ldots, A_{\exists - 1}\}$, are generated, where $\mathbb{R}$ is the total number of local states of all class templates. Let $C_i$ be a cut-off on the maximum length of paths needed to decide if the property holds in $A_i$. We first show that it is straightforward to calculate $C_0$. Second, we prove that $C_i (1 \leq i \leq \exists)$ can be reduced to the sum of $g_i$ and $C_{i-1}$, where $g_i$ is a number inherent to $A_i$. Finally, $C_\exists$ will be the cut-off that we expect.

Premature Termination: If some $C_i (i < \exists)$ is equal to zero, no undesired paths are feasible in $A_i$. The construction of $A_i$ ensures that $C_i$ is also equal to zero, meaning
that no undesired paths will be feasible in the original system. Then we know that the property does hold and the decision procedure can terminate. This makes the calculation of the final cut-off $C_0$ unnecessary. On the other hand, the decision procedure can also terminate with a conclusion that the property does not hold if a counter-example found in $A$, is also a feasible path of the original system.

**Guided Forward Reachability Analysis:** Typical forward reachability analyses of parameterized systems, such as [19] or its variant [13], use $\omega$-configurations as abstractions and fixed-point algorithms to construct a finite reachability graph. However, the problem with those approaches is that the loss of information during the generation of $\omega$-reachability graphs makes it difficult to construct a counter-example when a property does not hold. A cut-off for forward reachability analyses allows for a direct search over a non-compacted infinite reachability graph.

**Experiment Results:** The results show that our cut-off technique can help with the verification of real systems against properties of real interest. Experimental results show that verification efficiency has been greatly improved as a result of the introduction of our new cut-offs.

The rest of this paper is organized as follows: Preliminary definitions of parameterized systems are presented in Section 2. Section 3 gives an overview of our approach and those technical staff laying a foundation for every step of the approach is presented in Section 4. Experimental results are given in Section 5, and related work is included in Section 6. Section 7 concludes this paper.

## 2. PARAMETERIZED SYSTEMS

As for the definition of each component of a parameterized system, we will follow those notions presented in [10] except that the broadcast actions are excluded. To describe guarded transitions of a parameterized system, we first need the following definition:

**Boolean Guards/Actions [10]** Let $B = \{b_1, \ldots, b_n\}$ be a set of global boolean variables, and let $B'$ be their primed version. A boolean guard $\varphi_b$ is either the formula true or false or the conjunction of literals $l_1 \land \cdots \land l_p$, where $p \leq n$, such that $l_i$ is either $b_i$ or $\neg b_i$ from some $b \in B$. A boolean action $\varphi_a$ is a formula $b'_1 = v_1 \land \cdots \land b'_n = v_n$, where $v_i$ is one of true, false, $b_i$ for $1 \leq i \leq n$.

**Process** A process is a tuple $\langle S, \Sigma, \delta \rangle$, where:

- $S$: a finite set of states
- $\Sigma$: a set of actions used for constructing guarded transitions. Let $\varphi$ be $\varphi_b \land \varphi_a$. An action can be either of the following:
  - **Internal Action**, denoted as $\ell: \varphi$.
  - **Rendezvous Action**, including $\ell(\varphi)$ (output) and $\ell(\varphi)$ (input).

The action $\ell$ or $\ell(\varphi)$ is enabled if $\varphi_a$ is evaluated to be true and the global boolean variables will be updated according to $\varphi_b$, if the action is executed. The input action $\ell(\varphi)$ is enabled as long as there is an enabled rendezvous action $\ell$.  
- $\delta$: a subset of $S \times \Sigma \times S$, regulating a set of guarded transitions.

**Parameterized System** A parameterized system is a tuple $(B, C_1, \ldots, C_n, \{U_1, n_1\}, \ldots, \{U_k, n_k\})$, denoted as $P(t_1, \ldots, n_a)$, where $C_i = (\Sigma C_i, \Sigma C_i, \delta C_i)$ ($1 \leq i \leq l$) and $U_j = \{\Sigma U_j, \delta U_j\}$ ($1 \leq j \leq k$). Following the terminology of [17], $C_i$ through $C_l$ are called control processes, which can only be created once during each execution of the system. The templates $U_1$ through $U_k$, also called user processes, can have an arbitrary number of replications, as indicated by $n_1$ through $n_k$. All the control and user processes run in parallel synchronously and share information through the global boolean variables of $B$.

**Global State** A global state of a parameterized system has the form $G = \langle \rho, g \rangle$, where $\rho = \langle v_1, \ldots, v_n \rangle$ with $v_i$ $(1 \leq i \leq n) \in \{\text{true}, \text{false}\}$ is an evaluation of those boolean variables in $B = \{b_1, \ldots, b_n\}$, $g_i = \langle s_i, \ldots, s_i \rangle$ with $s_i = S_{C_i}$ $(1 \leq i \leq l)$ collects the set of states of control processes, and $g_2 = \langle s_1, \ldots, s_1, \ldots, s_k, \ldots, s_k \rangle$ with a dimension $s = n_1 + \cdots + n_k$ and $s_i \in S_{U_i}$ $(1 \leq i \leq k, 1 \leq j \leq n_i)$ represents the local states of replicated processes through $U_i$ through $U_k$.

Then the operational semantics of a parameterized system is defined as:

**Operational Semantics** Let $G = \langle \rho, s_1, \ldots, s_k \rangle$, $G' = \langle \rho', s_1', \ldots, s_k' \rangle$, and $\gamma = \rho \cup \rho'$. Then, $G \xrightarrow{\ell} G'$, denoting that the system transits from $G$ to $G'$ through a transition $\ell$, if and only if one of the following conditions holds:

- if there exists $i$ and $u$ such that $s_i \xrightarrow{\ell} s_i^,$ $u$ and $\gamma(\varphi) = \text{true}$, then $s_i' = s_i$ and $s_i^,$ = $s_i$ for all $j \neq i$.
- if there exists $i, j, u$ and $v$ such that $s_i \xrightarrow{\ell} s_i^,$ $u$, $s_j \xrightarrow{\ell} v$, and $\gamma(\varphi) = \text{true}$, then $s_i' = s_i, s_j' = s_j$ and $s_m = s_m$ for all $m \neq i, j$.

In this paper, properties of parameterized systems are expressed in property automata, whose accepting languages (paths leading to trap states) prescribe undesirable behavior of verified systems. If there exists a path accepted by a property automaton, which is also feasible in a parameterized system, we would say that the property does not hold in the system, otherwise it does.

**Property Automata** A property automaton $\phi$ is an FSM $\langle S_\phi, \Sigma_\phi, \delta_\phi, F, S_0, T \subseteq S_\phi \rangle$, where $S_0$ is a finite set of states, $F$ is a set of accepting states, $T$ is a set of trap states and $\delta_\phi \subseteq S_\phi \times \Sigma_\phi \times S_\phi$ is a set of transitions. A label in $\Sigma_\phi$ has one of the following forms:

- **Unsubscribed Event**, $\ell, \varphi, \phi_b$ where $\ell$ is an internal action or a communication event between $\ell$ and $\ell'$.
- **Subscribed Event**, $\ell(\varphi)$, where $I$ has a form of $\{i, j, \ldots, l, m, \ldots, m_j, \ldots, m_i\}$ with $i$ and $j$ being indexes to control processes and $I$, and $m_i$ being indexes to the ith replication of $U_i$ and the jth replication of $U_m$, respectively. This event refers to a transition in which one of the indexed processes or replications participates. If a user process’ replication is referred, we also call it a regulated replication.

Using predicate abstraction techniques of [18], representing each boolean predicate over concrete variables with a boolean variable, the producer-consumer problem presented in Fig. 1 can be abstracted as a finite set of automata. As shown in
3. OVERALL APPROACH

In this section, an overall view of our approach is presented. To start with, we first need the following definitions:

Configuration A configuration is a vector of the form $c = \langle s, a \rangle$, where $s$ is a state vector and $a$ is a counter vector.

A configuration can be thought of as the abstraction of a global state of a parameterized system. Instead of enumerating the state of each replication of user processes, the counter vector only records the number of replications in each local state. As a result, two global states can not be differentiated if they have the same number of replications in each local state.

Fig. 2, the two automata modeling the locks $LockFull$ and $LockEmpty$ correspond to the control processes of the parameterized system, while the other two automata, named $Producer$ and $Consumer$, are the user processes. The set of global boolean variables of the system consists of $bFull$ and $bEmpty$. The property automaton, stating that a $Producer$ can not put extra data into the buffer when it is full and a $Consumer$ can not withdraw data from the buffer when it is empty, is presented in Fig. 3, in which the diamond node represents a trap state.

**i-bounded Configurations** A vector $v \in \mathbb{Z}^n$ is said to be $i$-bounded if $v(j) \geq 0$ for all $1 \leq j \leq i$, where $v(j)$ represents the $j$th component of $v$. A configuration is said to be $i$-bounded if its counter vector is $i$-bounded.

**i-r-bounded Configurations** A vector $v \in \mathbb{Z}^n$ is said to be $i$-$r$-bounded if $0 \leq v(j) < r$ or $v(j) = \omega$ for all $1 \leq j \leq i$, where $\omega$ is greater than any integer. A configuration is said to be $i$-$r$-bounded if its counter vector is $i$-$r$-bounded.

A non-$i$-$r$-bounded configuration is an $i$-bounded but not $i$-$r$-bounded configuration.

By showing how the producer-consumer program is verified against the property given in Fig. 3, we’ll demonstrate how our approach works. The verification procedure can be summarized as follows:

**Construction of Extended Reachability Graph (ERG):** An ERG is actually the synchronous product of the property automaton and the parameterized system. It models the execution of control processes and regulated replications of a parameterized system and their execution’s impact on the property automaton, while the states of other replications of user processes are collected through counter vectors. Because two replications of a user process exhibit the same behavior as long as they are in the same local state, it still ensures that the original verification problem for a parameterized system holds if and only if all violation states are not reachable in the generated ERG.

Instead of directly using the complete ERG corresponding to a verification problem, a set of increasingly refined abstractions of it, $\{A_0, \cdots, A_i, \cdots, A_{\omega-1}\}$, is sequentially generated. In the abstraction $A_i$, each configuration is only required to be $i$-bounded. All graphs start from the same initial configuration $(s, a)$, in which all global boolean variables are initialized, all control processes and regulated replications are in their initial states, and the counter vector’s elements corresponding to initial states of user processes are set to $\omega$. The introduction of $\omega$ is to ensure that a conclusion will hold irrespective the size of a parameterized system. Then, new configurations reachable from the initial configuration are added. The procedure continues until all reachable $i$-bounded configurations have been enumerated.
However, the abstract reachability graph obtained is normally an infinite reachability graph because of the existence of $\omega$ elements in counter vectors and it prevents a direct forward reachability analysis. We will prove that $A_1$ only needs to include $i$-$BC_{<\omega}$-bounded configurations to determine the cut-off $C_i$, where $\theta$ is equal to 1 or 2 (because the parameterized systems considered in this paper contain no broadcast actions and an element in the counter vector will be added or decreased at most by 2 during a transition). If a non-$i$-$BC_{<\omega}$ bounded configuration is encountered during the construction of $A_i$, we just simply discard it.

**Inductive Calculation of Cut-offs:** The induction basis $C_0$ is obtained by investigating the coarsest abstraction $A_0$. Let $C_0$ be the length of the longest shortest paths in $A_0$ starting from one configuration to another in which the property automaton is in a trap state if such paths exist; otherwise, $C_0 = 0$. As for the induction of $C_i$ from $C_{i-1}$, $C_i$ will be set to the length of the longest shortest paths in $A_i$ starting from one configuration to another in which the property automaton is in a trap state; otherwise, $C_i$ will be set to the sum of $C_{i-1}$ and $g_i$, where $g_i$ is the length of the longest $i$-$BC_{<\omega}$-bounded path of $A_i$ if a non-$i$-$BC_{<\omega}$ bounded configuration has been discarded during the construction of $A_i$ and, otherwise, $C_i$ is equal to zero.

**Termination Criteria:** At most 3 inductions are needed, 3 being the total number of local states of user processes, and $C_3$ will be the cut-off on the maximum length of paths needed to decide if the property automaton holds in the verified parameterized system. The verification procedure can also terminate prematurely; a) an intermediate cut-off $C_i (i < 3)$ is equal to zero, indicating that no undesired behavior exists in the upward abstraction of the original parameterized system and thus the property does hold; b) a counter-example found in an abstract graph $A_i (i < 3)$ is not spurious and it is also feasible in the original parameterized system, indicating that the property does not hold.

An example abstract ERG $A_4$ for the producer-consumer program is presented in Fig. 4. The elements of each state vector, in left to right order, respectively correspond to the current state of the property automaton, the global boolean variables, $bFull$ and $bEmpty$, and the current states of the two control processes, $LockFull$ and $LockEmpty$. For the simplicity of notations, we use the notation $(\ldots, k_{ci(j)}, \ldots, n_{cj(j)}, \ldots)$ to represent a vector in which the element corresponding to the $i$th local state of Consumer (Producer) is equal to $k$ and the element to the $j$th local state is equal to $n$, while all other elements are zeros. As the elements $\omega_p$ and $\omega_c$ will be staying unchanged in the whole reachability graph, they are not explicitly specified other than in the initial configuration.

It should be noted that a different abstract reachability graph might be generated if the mapping from elements of counter vectors to local states of user processes is changed. For example, the configuration $c_3$ will not be 4-bounded if the state $s_3$ of Consumer is mapped on an element whose position index is less than 5, while it is under our current default mapping in which the first four elements are mapped on the local states $s_2, s_1, s_7$, and $s_8$ of Producer from left to right. In this paper, a randomly selected mapping is used and the research on heuristically selecting an optimal mapping for a verification problem is out of the scope of this paper. In addition, we can conclude the cut-off $C_4$ of the producer-consumer program is 2, as there is a path with a length of 2 leading to a trap state.

**4. DETAILED METHODOLOGIES**

In this section, the procedure to construct an ERG from a parameterized system is given. More importantly, several theorems supporting every step of our approach and their sketch proofs are provided. Before doing that, we first present two simple concepts. A path is said to be $i$-bounded if each configuration in it is $i$-bounded. An $i$-bounded path is said to be $i$-looped ($i$-loop-free) if there are (no) two configurations in it such that their state vectors are the same and the first $i$ elements of their counter vectors are componentwise equal. Similar concepts can also be defined for $i$-$r$-bounded configurations and paths.

**Construction of ERG:** Given a parameterized system $P = (B, C_1, \ldots, C_j, U_1, n_1, \ldots, U_k, n_k)$ and a property automaton $\phi = (S_0, \Sigma, b_0, F, T)$ characterizing the correctness of the parameterized system. We assume that the automaton refers to a set of regulated replications that are

---

1Detailed proofs, further details about the experiment systems and raw data given in the next section can be seen from http://itechs.iscas.ac.cn/qiusongyang/fbra.htm
sequentially generated from user processes \( \{U_r, \cdots, U_{rh}\} \).

An \( i \)-bounded (or \( i-r \)-bounded) ERG, is a directed graph \((V, E)\) such that:

- \( V \): a set of configurations. Each configuration \( c \in V \) has the form of \( \langle s, a \rangle \), where \( s \in S_0 \times B \times S_{C_1} \times \cdots \times S_{C_l} \times \cdots \times S_{U_{rh}} \), collects the current states of the property automaton, control processes and regulated replications and stores the evaluation of all global boolean variables, and the counter vector \( a \) collects states of non-regulated replications of user processes.
- \( E \subseteq V \times \Sigma_0 \times V \). Let \( C = \langle \langle s_0, \rho_0, s_1, \cdots, s_{i+1} \rangle, a \rangle \) be an \( i \)-bounded (\( i-r \)-bounded) configuration which has already been added to the graph, \( C' = \langle \langle \delta_0(s_0, \rho'_0), s'_1, \cdots, s'_{i+1}, a' \rangle, \gamma \rangle \), and \( \gamma = \rho \circ \rho' \). \( C \xrightarrow{} C' \), denoting that the combination of the property automaton and the parameter system transits from \( C \) to \( C' \) by executing a transition \( \ell \), if and only if the appending of \( C' \) does not form an \( i \)-looped path, and:

- \( \ell \cdot \varphi \): an internal action executed by a control process or a regulated replication and there exists \( i \) and \( \gamma \) such that \( s_i \xrightarrow{\ell} u \) and \( \gamma(\varphi) = true \), then \( s'_i = u \), \( a' = a \), \( s'_{i+1} = s_{i+1} \), \( s''_i = \delta_0(s_0, \ell) \). Assume \( \varphi \) is a total automaton.
- \( \ell \cdot \varphi \): an internal action executed by an unregulated replication and there exists \( s_m \xrightarrow{\ell} s_n \in \delta_\ell u \) where \( U_{\ell}(1 \leq \ell \leq k) \) is a user process, \( \gamma(\varphi) = true \), and the union of \( a = a = \langle -1_{s_m}, 1_{s_n} \rangle \) is \( i \)-bounded (\( i-r \)-bounded), then \( s'_i = s_i \), for all \( 1 \leq i \leq l + h \), \( a' = a + \langle -1_{s_m}, 1_{s_n} \rangle \), and \( s''_i = \delta_0(s_0, \ell) \).
- \( \ell \cdot \varphi \): the communication between \( s_i \xrightarrow{\ell} u \) and \( s_j \xrightarrow{\ell} v \), which are executed by a control process and a regulated replication, respectively, and \( \gamma(\varphi) = true \), then \( s'_i = u \), \( s'_j = v \), \( a' = a \), \( s''_i = \delta_0(s_0, i) \) and \( s''_j = s''_m \) for all \( m \neq i, j \).
- \( \ell \cdot \varphi \): the communication between \( s_i \xrightarrow{\ell} u \) and \( s_j \xrightarrow{\ell} v \), a transition executed by a control process or a regulated replication, and \( s_m \xrightarrow{\ell} s_n \), a transition executed by a user process \( U_{\ell} \), the sum of \( a \) and \( \langle -1_{s_m}, 1_{s_n} \rangle \) is \( i \)-bounded (\( i-r \)-bounded), and \( \gamma(\varphi) = true \), then \( s'_i = u \), \( a' = a + \langle -1_{s_m}, 1_{s_n} \rangle \), \( s''_i = \delta_0(s_0, l) \), and \( s''_j = s''_j \) for all \( j \neq i \).
- \( \ell \cdot \varphi \): the communication between \( s_m \xrightarrow{\ell} s_n \), a transition executed by a user process \( U_{\ell} \), and \( s_m \xrightarrow{\ell} s_n \), a transition executed by a user process \( U_{\ell} \), the vector \( a = \langle -1_{s_m}, 1_{s_n} \rangle + \langle -1_{s_m}, 1_{s_n} \rangle \) is \( i \)-bounded (\( i-r \)-bounded), and \( \gamma(\varphi) = true \), then \( s'_i = s_i \), for all \( 1 \leq i \leq l + h \), \( a' = a + \langle -1_{s_m}, 1_{s_n} \rangle + \langle -1_{s_m}, 1_{s_n} \rangle \), and \( s''_i = \delta_0(s_0, l) \).

During the construction of an ERG, it might be the case that there is a self-loop transition in some user process such that \( s_m \) and \( s_n \) in a transition \( s_m \xrightarrow{\ell} s_n \) refer to the same local state. As demonstrated in [17], we can insert intermediate configurations to address the issue. For an \( i \)-bounded ERG, a path of it is called a path with a trap state if the path is ended with a configuration in which the property automaton is in a trap state.

Then, we have the following theorems for the constructed ERGs:

**Theorem 1.** A property automaton holds in a parameterized system if and only if there are no paths with a trap state in the corresponding \( 3 \)-bounded ERG.

The theorem holds because an \( 3 \)-bounded ERG precisely models the behavior of the property automaton, control processes and regulated replications. It’s unnecessary to differentiate one non-regulated replication from another as they exhibit the same behavior if they are in the same local state and thus their states can be collected using counter vectors.

Let \( G \) be a configuration of \( A_i \) and \( \min(G, i) \) denote the length of the shortest \( i \)-bounded path with a trap state in \( A_i \) starting from \( G \), if at least one such path exists; otherwise, define \( \min(G, i) = 0 \). \( C_i = \max(\min(G, i) | G \in A_i) \) is defined as the length of the longest shortest \( i \)-bounded paths with a trap state starting from configurations of \( A_i \) if such paths exist, otherwise define \( C_i = 0 \). The next theorem states how \( C_i \) is calculated through inductions. Its proof is similar to the one given in [21] to prove the decidability of covering problems for vector addition systems.

**Theorem 2.** \( C_{i+1} \leq R \cdot (2C_i)^{i+1} + C_i \) for \( 0 \leq i \leq 3 \), where \( R = |S_0| \cdot |S_{C_1}| \cdot |S_{C_2}| \cdot |S_{U_{rh}}| \) is the product of the size of the individual automata.

**Proof.** Assume that there is an \( (i + 1) \)-bounded path with a trap state in the \((i + 1)\)-bounded ERG (otherwise, \( C_{i+1} = 0 \)).

Case 1: There is an \((i + 1)-2C_i\)-bounded path with a trap state. Then there must be an \((i + 1)-2C_i\)-bounded path with a trap state and the path is also \((i + 1)\)-loop-free. It is obvious that the path’s length must be \( \leq R \cdot (2C_i)^{i+1} \).

Case 2: Otherwise. Then there is an \((i + 1)\)-bounded path with a trap state which is not \((i + 1)-2C_i\)-bounded. The path can be re-presented as \( p_1 p_2 \) such that \( p_1 \) is \((i + 1)\)-\(2C_i\)-bounded and \( p_2 \) begins with a configuration \( c = \langle s, a \rangle \) which is not \((i + 1)-2C_i\)-bounded. Without loss of generality, we assume that \( a(i + 1) \geq 2C_i \). From case 1, we can choose \( p_1 \) to be of length \( \leq R \cdot (2C_i)^{i+1} \).

Since \( p_2 \) is an \( i \)-bounded path with a trap state, we know that there exists a path \( p_2 \) of length \( \leq C_i \) which is also an \( i \)-bounded path with a trap state and starting with \( s \). Note that all the entries in a state’s counter vector will be increased or decreased less than or equal to 2 in each transition. We can conclude that \( p_2 \) is an \((i + 1)\)-bounded path with a trap state. Hence, \( p_2 p_1 \) is an \((i + 1)\)-bounded path with a trap state of length \( \leq R \cdot (2C_i)^{i+1} + C_i \).

Based on Theorem 2, it is rather straightforward to construct an algorithm for a cut-off based reachability analysis of parameterized systems. We only need to check those paths with a length \( \leq C_3 \) to decide if a property holds. However, \( C_3 \) might be an astronomical figure if we simply use Theorem 2. For the example verification problem presented in Fig. 2, we will get \( C_0 = 2 \), \( C_1 \leq 66 \), \( C_2 \leq 278848 \), \( C_3 \leq 2.775 \times 10^{18} \). To further improve verification efficiency, the following lemma holds:

**Lemma 1.** If there are no paths with a trap state in the \((i + 1)-2C_i\)-bounded ERG and no \((i + 1)\)-bounded but non-\((i + 1)-2C_i\)-bounded configurations have been discarded during the construction, then \( C_{i+1} = 0 \).

**Lemma 2.** If non-\((i + 1)-2C_i\)-bounded configurations are discarded during the construction of the \((i + 1)-2C_i\)-bounded
Table 1: List of Example Problems Used in the Experiments.

| Index | Systems | φ     | | | | | |
|-------|---------|-------|---|---|---|---|
| 𝑃1   | Bin Example | Inverse Dependency (F) | 24 | Control | 6 | User | 9 | 4 |
| 𝑃2   | Loop Example | Looping Dependency (F) | 32 | Control | 4 | User | 7 | 4 |
| 𝑃3   | Simple Protocol | Mutual Exclusive (F) | 36 | Server | 9 | Client | 10 | 5 |
| 𝑃4   | No Orphan Packets (T) | | | | | | |
| 𝑃5   | Producer&Consumer | Pull Put (F) | 15 | | | | |
| 𝑃6   | Empty Get (F) | 18 | LockFull | 4 | Producer | 17 | |
| 𝑃7   | Produce First(T) | 18 | LockEmpty | 4 | Consumer | 17 | 16 |
| 𝑃8   | Empty Get and Full Put (F) | 34 | | | | | |
| 𝑃9   | Gas Station | Start Pumping First (T) | 19 | Pump1 | 12 | Client | 23 | 11 |
| 𝑃10  | Start Pumping First (T) | 62 | Pump2 | 12 | | | |

ERG, ℐ_{i+1}, will be less than or equal to the length of the longest shortest 𝑖-bounded paths with a trap state if such paths exist; otherwise, ℐ_{i+1} ≤ ℐ_{i} + 2g_{i+1}, where 2g_{i+1} the length of the longest path of the (𝑖 + 1)-2ℐ-bounded ERG.

The correctness of the first lemma is rather obvious. In Theorem 2’s proof, the length of the longest (𝑖 + 1)-2ℐ-bounded path is bounded by R(2C_{𝑖})^{+1}, while Lemma 2 uses a rather smaller bound g_{𝑖+1} obtained through investigating the verification problem itself. From these two Lemmas, we have the following theorem:

Theorem 3. To inductively calculate ℐ_{𝑖+1} from ℐ_{𝑖}, only a finite (𝑖 + 1)-2ℐ-bounded ERG is needed.

We also have two theorems supporting the premature termination of the inductive process, making the calculation of the final cut-off C_{3} unnecessary:

Theorem 4. If ℐ_{𝑖} = 0 for any 1 ≤ 𝑖 < 3, then there will be no j-bounded paths with a trap state in the j-bounded ERG for any j such that i < j ≤ 3 and the property does hold.

Theorem 5. For a path with a trap state of the (𝑖 + 1)-2ℐ-bounded ERG, it will be a concrete counter-example in the original parameterized system if each configuration in the path is 3-bounded and the property does not hold.

In addition, we can construct an (𝑖 + 1)-ℐ-bounded ERG instead of an (𝑖 + 1)-2ℐ-bounded one if no complementary actions are simultaneously enabled in a state of user processes as it is impossible for an element in the counter vector to be decreased by 2 during a transition.

5. EXPERIMENTAL RESULTS

The systems used in the following experiments are listed in Table 1. For each example problem, the details, such as the control processes and user processes of a parameterized system, the property to be verified, and their sizes are presented. Here, the size of an FSA is the sum of the number of states and transitions. The indicator “F” or “F” shows whether or not the property actually holds in the system being verified. The last column, 3, gives the total number of local states of all user processes.

Our algorithm, called Forward Bounded Reachability Analysis (FBRA), and existing typical algorithms for verification of parameterized systems are implemented in JAVA. Each data sample collected during the experiments consists of execution time, size of memory space needed to store the global states reached during the verification and some peculiar data to each algorithm. The algorithms were run on a HP laptop with a Dual P8400/62.2G/1GB memory running Windows XP.

5.1 Results of FBRA

We tested the algorithm FBRA on the example problems listed in Table 1. For the reason of limited space, the inductions to calculate cut-offs for some typical example problems are given in Table 2 (complete results are presented in the web page given in footnote 1). The boolean variable Bi, is true if an i-bounded path with a trap state is encountered during the ith induction, while B2i is true when some i-bounded but not i-ℐ-bounded nodes are encountered during the ith induction. The cut-off for each induction is listed in Ci and the number of states visited during each induction is recorded in Si. FBRA’s average execution time on each example problem is given in the last column.

From Table 2, we can first observe that Theorem 4 can improve verification efficiency dramatically by avoiding useless inductions for certain problems. As for the problems 𝑃4, 𝑃5 and 𝑃10, the FBRA algorithm draws a conclusive conclusion that the property holds in the verified parameterized system after the first induction because C0 is equal to zero. In the problems 𝑃2 and 𝑃3, a concrete counter-example is found during an intermediate induction and the whole decision procedure also terminates prematurely as a result of Theorem 5.

Secondly, the variable B2i is remarkably similar to the evaluation function in branch and bound search algorithms. If an i-bounded but not i-ℐ-bounded node is encountered during the ith induction, then B2i being set to true, the branch rooted with the node will be discarded for further exploration to find an i-bounded path with a trap state. If B2i = true for some 1 ≤ i ≤ 3, it also indicates that the state space explored in the ith induction is reduced because of the introduction of Ci−1, as shown in P1, P2 and P3.

Thirdly, assume that the algorithm FBRA does not terminate after the 3rd induction, as shown in P1 and P3. The (3 + 1)th induction, in which the original parameterized system instead of one of its abstractions is used, is going to be executed to calculate C3. If C3 is equal to zero, we can claim that the property does hold. Otherwise, a counter-example whose length is not greater than C3 may exist in the parameterized system. If being lucky enough, we might encounter such one counter-example during the calculation of C3, in the case of P1, and the (3 + 2)th induction will be completely unnecessary. Otherwise, all the paths with
a length $\leq C_3$ are needed to be checked to decide if a real counter-example does exist, as shown in $P_6$.

### 5.2 FBRA vs. others

Although cut-off approaches are presented in [11], [4], [3], [17] and [12], those results highly depend on some unique characteristics of verification problems, such as ring or network topologies of processes, process symmetry, indexed logic without the next operator. As a different combination of parameterized systems and property specification is used in this paper, it prevents the direct comparison of our approach with existing cut-off approaches. To demonstrate the effectiveness of our approach, we made a comparison between the algorithm FBRA and typical existing algorithms for verification of parameterized systems, albeit they are not based on cut-offs. We implemented the Forward Reachability Analysis (FRAKM) algorithm proposed by R.M. Karp and R.E. Miller [19], a variant of FRAKM proposed by E.A. Emerson and K.S. Namjoshi [13], named FRAEN, and the Backward Reachability Analysis (BRA) algorithm proposed by P.A. Abdulla et al. [1] (FRAEN and BRA are also both used by J. Espana et al. in [14]). In the literature, a lot of algorithms have been derived from these three basic ones to verify parameterized systems or other systems with an infinite state space.

In these algorithms, the concept of quasi-order, a binary relationship (i.e. reflexive and transitive) on the set of global states, plays a critical part in constructing a finite reachability graph. A configuration $c = (s, a)$ is said to be the pre-order of another configuration $c' = (s', a')$, denoted as $c \preceq c'$, if $s = s'$ and $a \preceq a'$. Let $d$ and $d'$ be two configurations that have been added to a partially generated reachability graph. Let $e$ be a new configuration backward or forward reachable from $d$, but not from $d'$. The strategies adopted by those algorithms to decide if the new configuration is added to the existing partial reachability graph are different from one another. They are summarized in Table 3, where each symbol $\lor$ denotes that the algorithm in the same row uses the rule in the same column to compact the state space and each $\times$ symbol does not. In the algorithm BRA, the new configuration $e$ does not need to be explicitly represented if $d \preceq e$, $d' \preceq e$, $d = e$, or $d' = e$. The new configuration $e$ will be replaced by a corresponding $\omega$-configuration if $d \preceq e$ or $d = e$ in the algorithm FRAEN.

The algorithm FRAEN extends FRAKM by introducing an extra rule that the new configuration $e$ is discarded and a new edge from $e$'s predecessor to $d'$ is added if $e \preceq d'$.

### 6. RELATED WORK

To counter the state explosion problem in which the number of states of a reachability graph may grow exponentially with the number of processes, some approaches are presented in [5] [6] [22] [20] to reduce an infinite set of verification problems, a verification problem for a system with a certain size, to a single problem. Typically in [5], the reduction is based on the collapse theorem stating that, for all
Parameterized systems defined in this paper can also be modeled as Petri nets. To counter the explosion caused by the high dimensionality of practical Petri nets, those places (correspond to local states of user processes) which are thought of being not important are discarded to construct an abstraction in [16]. If the abstraction is too coarse, an automatic refinement is performed and a more precise abstraction is obtained. The process is iterated until the property is proved to be true or false. However, only partial space of each abstraction is explored in our method because of the cut-off $C_i$ and the iterative process can be terminated when $C_i = 0$ for some $0 \leq i \leq 3$.

7. CONCLUSION AND DISCUSSIONS

In this paper, a novel approach is proposed for the verification of parameterized systems based on cut-offs on the maximum length of paths needed to be explored. Given a parameterized system and a property automaton stating a set of undesirable behavior of the system, a cut-off on the maximum length of paths to decide if the property holds is determined through an inductive procedure. In each induction, only a finite abstract reachability graph is needed and thus the verification of a parameterized system with an infinite state space is turned into a series of verifications of finite state systems. The existence of a cut-off on the maximum length of paths allows for a direct forward reachability analysis of parameterized systems, otherwise a compacted finite reachability graph has to be constructed first, and it is rather straightforward to construct a counter-example when a property does not hold. Experimental results show that the algorithm outperforms typical forward and backward reachability analysis algorithms in many of the example problems.

In the definition of property automata, only behavior of indexed user processes is considered. Actually, they can be extended to safety properties, such as global deadlock free properties. One way is to use property automata to describe all possible "bad" sequences of resource acquire/release operations. An alternative one is to convert a deadlock free property as it is guaranteed that at least one of the state transitions is always enabled. As future work, our approach can be extended to liveness properties in Büchi Automata, although calculating $C_{i+1}$ from $C_i$ will become much more complicated.

In addition, there is an implicit ordering on the local states of user processes in this paper. As discussed previously, a
Acknowledgements

The work was partially done during the first author’s visit to the Laboratory for Advanced Software Engineering Research (LASER) in the Department of Computer Science at the University of Massachusetts Amherst. The work was inspired by a conjecture proposed by Prof. Leon J. Osterweil, stating that the number of instances of user processes is bounded for certain types of verification problems and the bound is a function of the attributes of the system and the property to be verified. Prof. Lori A. Clarke gave a lot of valuable comments on how to make the presentation more accurate and clearer. The discussions with Prof. George S. Avrunin were also greatly helpful. The authors wish to thank anonymous reviewers for their valuable contributions and comments.

The work was partially supported by the National Natural Science Foundation of China under grant Nos. 60903051 and 90718042, the Hi-Tech Research and Development Program of China under grant Nos. 60903051 and 2007AA010903, the 973 program under grant No. 2007CB310802, as well as the Knowledge Innovation Program of the Chinese Academy of Sciences under grant No. ISCAS2009-DR09.

8. REFERENCES


