Hindawi
Mobile Information Systems
Volume 2017, Article ID 5731678, 10 pages
https://doi.org/10.1155/2017/5731678

Research Article

A Formal Approach to Verify Parameterized Protocols in Mobile Cyber-Physical Systems

Long Zhang,[1] Wenyan Hu,[2] Wanxia Qu,[1] Yang Guo,[1] and Sikun Li[1]

[1]*College of Computer, National University of Defense Technology, Changsha, China*
[2]*Carnegie Mellon University, Pittsburgh, PA, USA*

Correspondence should be addressed to Yang Guo; guoyang@nudt.edu.cn

Received 16 February 2017; Accepted 12 April 2017; Published 10 May 2017

Academic Editor: Jun Cheng

Mobile cyber-physical systems (CPSs) are very hard to verify, because of asynchronous communication and the arbitrary number of components. Verification via model checking typically becomes impracticable due to the state space explosion caused by the system parameters and concurrency. In this paper, we propose a formal approach to verify the safety properties of parameterized protocols in mobile CPS. By using counter abstraction, the protocol is modeled as a Petri net. Then, a novel algorithm, which uses IC3 (the state-of-the-art model checking algorithm) as the back-end engine, is presented to verify the Petri net model. The experimental results show that our new approach can greatly scale the verification capabilities compared favorably against several recently published approaches. In addition to solving the instances fast, our method is significant for its lower memory consumption.

1. Introduction

A cyber-physical system (CPS) [1] is an integration of computation and physical components. The improvement of contemporary mobile devices, such as smartphones and wearable electronics, enables the formation of mobile CPSs [2, 3]. A mobile CPS could be considered as a subcategory of CPSs with inherent mobile features [4, 5]. Different from traditional CPSs, mobile CPSs could be built on mobile devices that travel with their owners. For example, mobile social networking [6, 7] helps people communicate with each other on their daily commute to and from work, traveling along the same rotes at about the same time. Due to distributed interactions of cyber word, physical word, and human behaviors, the mobile CPS becomes more complex. Asynchronous communication and the arbitrary number of components make the mobile CPS appear similar to a parameterized system. It is difficult to guarantee the parameterized system's correctness for any natural number [8, 9].

Due to the tight market windows and safety-critical nature of their applications, it has become an urgent need to design error-free mobile CPSs and thus a significant amount of time is spent on ensuring the correctness of mobile CPS designs. The verification of the CPS designs becomes an important issue [10]. Formal methods, replacing the traditional testing methods for large mobile CPSs, have been successfully used for verifying software, hardware, and physical systems in the past decades [11]. Model checking [12, 13] is an automatic formal approach to verify if the specification satisfies the properties and has been used in finite and infinite state system verification successfully.

Abstraction [14, 15] is a good way to reduce the state space. By abstraction, each agent of the mobile CPS can be modeled as a finite state automaton in which local transitions model one of the following: an internal action, a broadcast, or a reception of a message. A mobile CPS is defined as the composition of a finite but arbitrary number of copies of the automaton running in parallel. A mobile CPS which combined with an arbitrary number of components is a parameterized system, which is a wide class infinite system, including cache coherence protocols and mutual exclusion protocols.

Parameterized systems arise naturally in the modeling of mutual exclusion algorithms, distributed protocols, or cache coherence protocols. Parameterized verification [8] is aimed at verifying families of transition systems for all values of the parameter. Counter abstraction [14] is natural to model

parameterized systems into Petri nets and their extensions. Petri net is a powerful mathematical tool and has been used widely for modeling and verifying CPSs [10, 16, 17].

In this paper, we propose a formal approach to verify the safety properties of parameterized protocols in mobile CPSs. By using counter abstraction, the protocols of mobile CPSs are described as Petri nets, and then the state-of-the-art model checker is used to check the safety properties.

The significant contributions of this paper are as follows:

(i) We propose a new method based on the SAT-based model checking algorithm to verify the parameterized protocols of mobile CPSs. By using counter abstraction, we describe the parameterized protocol as a Petri net and then translate it into a finite state machine (FSM), so that IC3 [18, 19], the state-of-the-art finite state model checking algorithm, can be used as the back-end engine.

(ii) A smart encoding technique is introduced to make the verification efficient. A bounded Petri net is transformed into a FSM and described as a general format for most model checkers.

(iii) To improve the scalability of parameterized protocols verification, an incremental algorithm is proposed to make IC3 perform more efficiently.

The rest of this paper is organized as follows. In Section 2, we review the related work. Section 3 presents necessary preliminaries used in this paper. In Section 4, we propose our new method based on the model checking algorithm and give more details of the implementation and optimization. Section 5 shows the experimental evaluation on parameterized protocols. Section 6 concludes this paper and discusses future works.

2. Related Works

As a successful application in traditional hardware and software verification, model checking has been frequently used in CPS verification, especially for safety-critical CPSs. Akella and McMillin [20] encoded the physical system into an event-based discretized system and modeled the associated CPS by Security Process Algebra. The model checker, CoPS, was used to check the confidentiality properties. A statistical model checker has been recently utilized to analyze some aspects of CPSs [21]. However, this method also suffers from the classical model checking problems, such as the state space explosion and the lack of ability to reason about mathematical relations. Bae et al. [22] combined model checking and Multirate PALS (physically asynchronous, logically synchronous) methodology for the first time to verify an airplane turning control system. More cases should be studied for the verification of distributed cyber-physical systems using Multirate PALS.

Petri nets are well-known tools for modeling and verification of distributed systems and CPSs. Xu and Deng [16] proposed a Petri nets-based method for architectural modeling of mobile agent systems. Chen et al. [17] investigated the use of Petri nets for modeling coordinated cyber-physical attacks on the smart grid. A novel hierarchical method was proposed to construct large Petri nets from a number of smaller Petri nets that can be created separately by different domain experts. Vita [23, 24], which is a novel mobile cyber-physical system for crowdsensing applications, introduced Petri nets to design a high level service state synchronization mechanism to address the possible unavailable situations of mobile devices in mobile CPSs. In order to define the functionality of traveler information systems (TIS) and integrate new functions and technologies based on cloud computing and mobile communications, Nemtanu et al. [25] presented a Petri nets-based model of this system. Zhang et al. [26] proposed a mechanism to model fault tolerated mobile agents by using colored Petri nets.

There is a large amount of related works on automating the parameterized verification problem [27–29]. The theorem prover PVS, for example, has been successfully applied to verify Small Aircraft Transportation System (SATS) [30]. By using the Model Checker Modulo Theories, Johnson and Mitra presented a model checking method for SATS [31]. Guo et al. [32, 33] proposed a new method to reduce the state space of parameterized systems by two-dimensional abstraction (TDA). Asynchronous composition was the key part of TDA but suffered from higher memory consumption.

3. Preliminaries

Petri nets have been popular models for various types of asynchronous or concurrent processes. A Petri net is a directed graph consisting of places (drawn as circles), transitions (typically boxes), and directed arcs. Input places point to a transition, and a transition points to output places. A number of tokens move around the net from place to place, and the distribution of tokens among the places (called the *marking*) represents the dynamic state of the entire modeled system. The formal definition of Petri nets is as follows.

Definition 1 (Petri net). A Petri net (PN) is a triple $\mathcal{PN} = (P, T, F)$, where P is a finite set of places, T is a finite set of transitions disjoint from P, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is flow relations for the set of arcs.

The configuration of $\mathcal{PN}$ is *markings*, which can be seen as the multisets of places. The semantics of $\mathcal{PN}$ is given by *markings*. A *marking* is a function $m : P \to \mathbb{N}$, which describes the number of tokens $m(p)$ in place $p \in P$. In the sequel, if places are ordered by $P = \{p_1, p_2, \ldots, p_n\}$, we often identify m and the vector $\langle m(p_1), m(p_2), \ldots, m(p_n)\rangle$.

Example 2. As shown in Figure 1, a simple example contains all components of a Petri net. There are two places and three transitions. Arcs have capacity 1 by default; if other than 1, the capacity is marked on the arc. Places have infinite capacity, and transitions have no capacity and cannot store tokens at all. The current marking is $\langle 1, 0\rangle$. If t_1 is fired, the next marking will be $\langle 2, 0\rangle$. If t_2 is fired, the next marking will be $\langle 0, 1\rangle$. The transition t_3 cannot be fired now.

In particular, it has been shown that certain communication procedures, which are common when programming

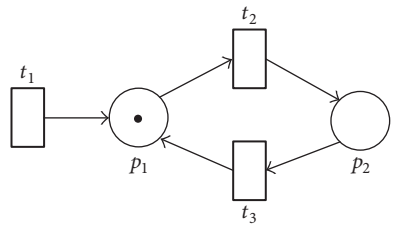

Figure 1: A Petri net with two places and three transitions.

distributed systems, can easily be modeled by plain Petri nets, but other communication procedures such as broadcast are not easily captured by plain Petri nets. For that reason, we address two extensions of the model in this paper, following the definition in [34].

(i) Petri nets with transfer arcs, in which a transition can also consume all tokens present in one place and move them to another

(ii) Petri nets with reset arcs, in which a transition can delete all tokens present in one place

This significance of these extensions in terms of modeling power has been demonstrated, for instance, in the modeling and verification of parameterized protocols in [35].

A common way to model a parameterized system by a Petri net is to apply the idea of the counter abstraction [8]. This principle consists of mapping each single process to a token and representing each state of each type of process by a place. In this case, the presence of a token in a given place $p \in P$ indicates that, in the current global state of the system, there is a process that corresponds to p in its local states. The transition of the Petri net then consumes and produces tokens to move the associated process from one state to another. This formalization has the drawback of abstracting away the actual identities of the processes. Still, some interesting properties, for instance, safety properties, can be verified at this level of abstraction.

In this paper, we focus on the safety property, which is equal to the coverability of well-structured transition systems (WSTSs) when expressing the safety property as the upward-closed set. Petri nets are WSTSs (with respect to $\leq$) [36]. We present the related notions of WSTSs and then define the PN safety problem.

Definition 3 (well-quasi-ordering). A well-quasi-ordering (wqo) is a reflexive and transitive binary relation $\preceq$ over set X, and for every infinite sequence $x_0, x_1, x_2, \ldots$ of elements from X, there exists $i < j$ such that $x_i \preceq x_j$. For $Y \subseteq X$, the upward-closure of Y is the set $\uparrow Y = \{x \mid \exists y \in Y,\ y \preceq x\}$. A set U is said to be $\preceq$-upward-closed (or simply upward-closed if $\preceq$ is clear from the context) if $U = \uparrow U$.

In Figure 1, the safety property is defined whether the tokens number of place p_2 is greater than or equal to 2; then we can use the upward-closed set $p_2 \geq 2$ to express the property.

Definition 4 (well-structured transition systems). A well-structured transition system (WSTS) is a transition system equipped with a wqo on its states that satisfies the monotonicity property. A WSTS is a triple $(S, \rightarrow, \preceq)$ such that

(1) S is the (possibly infinite) state space,

(2) $\rightarrow \subseteq S \times S$ is transition relation,

(3) $\preceq$ is a wqo over S,

(4) for all $x, x', y \in S$, if $x \rightarrow x'$ and $x \preceq y$, there exists y' such that $y \rightarrow y'$ and $x' \preceq y'$.

The covering relation $\leq$ between Petri net markings is a wqo. A $\mathcal{PN} = (P, T, F)$ and the initial marking m_0 give rise to a WSTS $(S, I, \rightarrow, \leq)$, where S is the set of markings and I corresponding with m_0 is the initial states. The transition relation is defined as follows: there is an edge $m \rightarrow m'$ if and only if there is some transition $t \in T$ such that when transition t was fired, the marking m yields a new marking m'. The coverability problem for PN is defined as the coverability problem on this WSTS.

Definition 5 (PN safety problem). Given a Petri net $\mathcal{PN} = (P, T, F)$ and the initial marking m_0, we get a WSTS $(S, I, \rightarrow, \leq)$. Then given an $\leq$-upward-closed set $U \subseteq S$, does there exist a sequence $x_0 \rightarrow x_1 \rightarrow \cdots \rightarrow x_l$ such that $x_0 \in I$ and $x_l \in U$? We write safe$(\mathcal{PN}, I, U)$ if the answer is "no."

Example 6. As shown in Figure 1, if the initial marking is $m_0 = \langle 1, 0\rangle$, and the safety property is described by the upward-closed set $p_2 \geq 2$, there exists a sequence $\langle 1, 0\rangle \rightarrow \langle 2, 0\rangle \rightarrow \langle 1, 1\rangle \rightarrow \langle 0, 2\rangle$, which means the Petri net is not *safe*.

4. Incremental Bounded Model Checking Algorithm

This section describes how to bound a Petri net to an equivalent FSM and then verifies the safety properties by using SAT-based model checkers. An incremental method is proposed, which benefits more from the modern SAT solver by assumption.

4.1. Cut Off the Petri Net to FSM. In general, Petri nets are infinite state systems, as the number of tokens can be assigned with arbitrary $n \in \mathbb{N}$. In order to use the finite state model checking algorithms, we cut off the Petri net to FSM by a given boundary $B \in \mathbb{N}$. As the counter abstraction is used to model the parameterized systems, the boundary B can be defined as the current process number.

Definition 7 (finite state machine). A finite state machine is a quadruple $\mathcal{M} = \langle S, R, S_0, \text{In}\rangle$, where S is a finite set of states, $R \subseteq S \times S$ is the transition relations, $S_0 \subseteq S$ is the initial states, and In is a finite set of inputs.

To cut off the Petri net to FSM, we represent Petri nets as follows. Let $P = (p_1, p_2, \ldots, p_n)$ be the set of places. A marking m is represented as the tuple of natural numbers

$\langle m(p_1), m(p_2), \ldots, m(p_n)\rangle$. A transition t is represented as a pair $(\mathbf{g}, \mathbf{e}) \in \mathbb{N}^n \times \mathbb{N}^n$, where $\mathbf{g}$ is the guards and $\mathbf{e}$ is the effects. Formally, $\mathbf{g} = (F(p_1, t), F(p_2, t), \ldots, F(p_n, t))$, which represents the enabling condition, and $\mathbf{e} = (F(t, p_1), F(t, p_2), \ldots, F(t, p_n))$, which represents the yield marking m'.

The Petri net is a concurrent model, in which just one transition can be fired at one time. In the equivalent FSM, we introduce extra inputs to simulate and select one transition to be fired randomly.

Definition 8 (B-bounded Petri net, B-bounded upward-closed set). Given a Petri net $\mathscr{PN} = (P, T, F)$ and a boundary $B \in \mathbb{N}$, a B-bounded Petri net PN_B is a $\mathrm{FSM}\langle S, R, S_0, \mathrm{In}\rangle$, where S is the subset for all markings, so that, for a marking m, $m(p_i) \leq B$ for all $1 \leq i \leq n$; R is the subset of T, so that, for $t = (\mathbf{g}, \mathbf{e}) \in T$, the guard $\mathbf{g}$ can be fired under the boundary B; S_0 is the bounded initial marking m_0; In is the set of extra inputs for selecting which rule to be fired. Given an upward-closed set U, U_B is the B-bounded upward-closed set, which cuts off the infinite U to finite U_B by the boundary B.

B is the total token's boundary, so the summary of each place's token number should be less than or equal to B. Here an adder is used to count the total tokens, which will be discussed in Section 4.3.

The bad states are presented with an upward-closed set U in Petri nets. We bound U with the boundary B, by cutting off the upward-closed set with B. For instance, for an upward-closed set $p_2 \geq 2$, and the boundary 10, the bounded upward-closed set is $2 \leq p_2 \leq 10$.

Here, we introduce a function $cut\text{-}off(^*, B)$ to map the Petri net PN or upward-closed set U to B-bounded Petri net or B-bounded upward-closed set. That is to say, $\mathrm{PN}_B = cut\text{-}off(\mathrm{PN}, B)$, and $U_B = cut\text{-}off(U, B)$.

4.2. SAT-Based Model Checking Algorithms for Petri Nets. The Boolean Satisfiability (SAT) problem is a well-known NP-complete constraint satisfaction problem. With the introduction of bounded model checking (BMC) [37], it becomes clear that SAT solvers can be used for model checking [12]. There has been significant progress on SAT-based model checking techniques in the past two decades, including BMC, interpolation [38], and IC3.

IC3, known also as property directed reachability (PDR), is a recently proposed SAT-based model checking technique for the analysis of sequential circuits. IC3 maintains a list of trace: $[R_0, R_1, \ldots, R_N]$. The first element R_0 is special; it is simply identified with the initial states. For $k > 0$, R_k is a set of clauses that represents an overapproximation of the states reachable from the initial states in k steps or less. Together with the trace, the IC3 algorithm consists of a set of proof-obligations, which consists of a frame number k and a cube s. By manipulating the trace and the set of proof-obligations, IC3 gets new facts and adds them into the trace until it either (1) produced an inductive invariant proving the property or (2) added a proof-obligation at frame 0 with a cube that intersects the initial states, which is a counterexample. Without unrolling the model, IC3 performs better than most SAT-based model checking algorithms, especially in memory consumption.

Given a Petri net model PN, the safety property U, and the boundary $B \in \mathbb{N}$, U is expressed as an upward-closed set, and B is the current process number of the parameterized system to be verified. By Definition 8, we create the bounded Petri net model and bounded upward-closed set by the function $cut\text{-}off(*, B)$. PN_B is a FSM and can be translated into propositional logic directly. U_B represents the safety property for PN_B. An SAT-based model checker is used to check the property. There are multiple choices of model checkers. Here the state-of-the-art model checker IC3 was used as the back-end engine, because of its lower memory consumption. If the model checker returns UNSAT, it means PN is safe for the current boundary B; otherwise, a counterexample will be found.

We note this method as B-bounded model checking algorithm. Because of the use of finite state model checking algorithms, the B-bounded model checking algorithm will terminate when we find a counterexample or prove the safety.

For parameterized verification problem, we want to verify the system on an arbitrary parameter. If the maximum process number is M, we need M times single runs for $1 \leq B \leq M$. For each single run, the FSM is encoded into a new propositional formula to use an SAT solver, separately.

Fortunately, the modern SAT solver supports the incremental mechanism, so that the new SAT problem can be solved based on the previous solve result. The back-end SAT engine used in this paper is MINISAT [39], which supports the incremental mechanism by assumption. There is a vector to store the assumption variables which will be assigned to *True*. Hence, we introduce some extra variables, named active literals, to control the boundary of the bounded Petri net model.

Definition 9 (Inc-bounded Petri net, Inc-bounded upward–closed set). Given a Petri net $\mathscr{PN} = (P, T, F)$, a boundary $n \in \mathbb{N}$, and a maximum boundary $m \in \mathbb{N}$, an Inc-bounded Petri net $\mathrm{PN}_{n\to m}$ is a m-bounded Petri net PN_m equipped with the active literals vector $\mathbf{v}$ for controlling the boundary incrementally. An Inc-bounded upward-closed set $U_{n\to m}$ is a m-bounded upward-closet set U_m equipped with $\mathbf{v}$.

If $\mathbf{v} = \{a_{n+1}, a_{n+2}, \ldots, a_m\}$, the current boundary is n. If a_{n+1} is popped out, the boundary increases to $n + 1$. If $\mathbf{v} = \emptyset$, the boundary reaches the maximum boundary m. A new function *inc-cut-off(*,n,m)* is introduced to map the Petri net PN or upward-closed set U to Inc-bounded Petri net or Inc-bounded upward-closed set. We write that $\mathrm{PN}_{n\to m} = inc\text{-}cut\text{-}off(\mathrm{PN}, n, m)$, and $U_{n\to m} = inc\text{-}cut\text{-}off(U, n, m)$.

Algorithm 1 shows our new algorithm to verify the bounded Petri net from boundary n to m incrementally. The inputs are a Petri net model PN, safety property U, a base boundary n, and a maximum boundary m. U is expressed as an upward-closed set. The algorithm returns *unsafe* if it finds a counterexample; otherwise, it returns *safe* and proves the system is safe from parameters n to m.

Lines 1–5. Generate the active literals and push them into the assumption vector $\mathbf{v}$. Then, set the current boundary as n.

Input:
PN: a Petri net model to describe the parameterized protocol
U: an upward-closed set to describe the safety property
n: base boundary
m: maximum boundary
Output:
safe or *unsafe*
(1) initial assumption vector $\mathbf{v} \coloneqq \varnothing$ // push all active literals into assumption vector
(2) **for** $i \coloneqq m;\ i > n;\ i \coloneqq i - 1$ **do**
(3) $\mathbf{v}$.push(a_i)
(4) **end for**
(5) $i \coloneqq n$ // the initial boundary is n
(6) $\mathrm{PN}_{n\to m} \coloneqq$ *inc-cut-off*(PN, n, m) // create the incremental bounded Petri net
(7) $U_{n\to m} \coloneqq$ *inc-cut-off*(U, n, m)
(8) **while** $\mathbf{v} \neq \varnothing$ **do**
(9) // use an SAT-based model checker to verify the property at the boundary i
(10) **if** $\mathrm{PN}_{n\to m} \models U_{n\to m}$ **then**
(11) print CEX
(12) RETURN *unsafe*
(13) **else**
(14) print "PN is safe for current boundary i"
(15) $\mathbf{v}$.pop() // the boundary is increased by 1
(16) $i \coloneqq i + 1$
(17) **end if**
(18) **end while**
(19) RETURN *safe*

Algorithm 1: Incremental bounded model checking algorithm.

Lines 6-7. Create the incremental bounded Petri net model by using the assumption vector $\mathbf{v}$.

Lines 8–19. The while-loop is the main routine to verify the model incrementally. If the condition is satisfied at line 10, the model checker finds a counterexample and returns *unsafe*. If the model bounded by i is safe, then the algorithm increases the boundary at lines 15 and 16 and calls the SAT-based model checker again to solve the new model with higher boundary. When vector $\mathbf{v}$ is empty, the algorithm proves that the input PN is safe from boundaries n to m.

Algorithm 1 reuses the context from the previous solving results. The main routine in Algorithm 1 is based on an SAT-based model checker. Hence, the algorithm terminates when it finds a counterexample at line 12 or proves safety at line 19.

4.3. Implementation and Optimization. In this section, we introduce key points which make a great contribution in improving the performance.

The Petri Net Format. The input Petri net is encoded in the MIST format (https://github.com/pierreganty/mist/). Each place is mapped to a variable, and each transition corresponds to a rule. For each rule, there are guards and effects, as described in Definition 8. The guards are the conditions under which this rule can be fired, and the effects describe how tokens transfer from places. The target is the safety property to be verified, which is expressed as an upward-closed set.

The FSM Format. AIGER (http://fmv.jku.at/aiger/) is a format, library, and set of utilities for And-Inverter Graphs (AIGs). The hardware model checking competition (HWMCC) uses AIGER as input format, and most modern model checkers support AIGER as the input model. AIGER is a good way to describe FSM and can be translated into a propositional logic for an SAT solver. The bounded Petri net is encoded as an AIGER model, where each place corresponds to state variables in Boolean value. Extra input variables are introduced to select which rule to be fired and then update the state variables to set up the transition relations equally.

Encoding: Binary versus Unary. Encoding is important for SAT solvers. In this paper, both binary and unary encodings

Table 1: Encoding: binary versus unary.

n	Binary	One-hot
0	000	00000001
1	001	00000010
2	010	00000100
3	011	00001000
4	100	00010000
5	101	00100000
6	110	01000000
7	111	10000000

were used to encode the places in the Petri net model. As shown in Table 1, binary and unary encodings are used to encode the natural numbers. One-hot encoding is one possible unary encoding, where just one bit is "1" and the others are all "0." The binary encoding needs $\lceil \log_2[N] \rceil$ bits, and the unary encoding needs N bits to encode the natural number $0 \sim N$.

Full Adder. A full adder is designed to count the total token numbers to bound the Petri net. As binary and unary encodings are used to represent the token numbers for each place, we present how to design an n-bit binary and unary full adder, respectively.

A n-bit binary full adder is just combing n single 1-bit binary adders together. The 1-bit binary adder can be described using the following logics. The logics AND, OR, and XOR are represented as $\wedge$, $\vee$, and $\oplus$, respectively.

$$\begin{aligned} S &= A \oplus B \oplus C_{\text{in}} \\ C_{\text{out}} &= (A \wedge B) \vee \left(C_{\text{in}} \wedge (A \oplus B)\right) \end{aligned} \tag{1}$$

There are many different ways to design unary adder, but a simple n-bit unary adder is presented as follow:

$$S_i = \bigwedge_{j=0}^{i} \left(A[j] \wedge B[i-j]\right), \quad 0 \le i < n. \tag{2}$$

Structural Information. When using unary encoding to represent the token's number of places, it is important to give this structural information to the SAT solver. Hence, we add some extra logics as a constraint to the AIGER circuits. Figure 2 shows the logics to check if the n-bit vector $\mathbf{x}$ is encoded in one-hot. The total number of gates is $3 * n + 2$.

5. Experimental Evaluation

We have implemented the incremental bounded model checking algorithm in a tool named PNPV. PNPV is implemented with C++ and uses MINISAT as the back-end SAT solver. All input instances are encoded in the MIST format.

To measure PNPV's performance, we compare with TDA [32], which was used to verify parameterized cache coherence protocols. We also compare with the MIST toolkit (https://github.com/pierreganty/mist) with the classical backward [40] and EEC [41] algorithm for WSTS coverability on Petri nets instances.

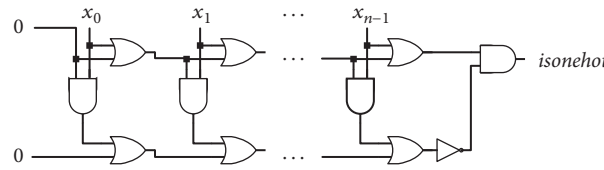

Figure 2: One-hot encoding checking logics.

All experiments are performed on a machine, with Intel 2.60 GHz CPU and 16 GB main memory, running CentOS 6.5 in 64-bit. The running time is limited to 600 seconds and memory to 2 GB.

5.1. Benchmarks. Memory coherence is very important for both the multicore processors and mobile CPSs. We verify several classical parameterized cache coherence protocols as described in [35]. We collected 12 Petri nets from the MIST repository, where six bounded Petri nets are all safe and six plain Petri nets are all unsafe. Some of those Petri nets are used to model the communication protocols in mobile CPSs.

5.2. Evaluation. As shown in Table 2, the unary encoding performs about 4~19 times better than binary. By using unary encoding, we can solve 160 processes for Illinois and Firefly protocols, but just 22 and 18 when using binary. For Berkeley, PNPV solves 146 and 18 processes by unary and binary, respectively. German protocol is an industry-like cache coherence protocol, for which 97 and 5 processes are solved by using unary and binary encoding, respectively. The CSM-broad and Dragon protocols both have about 90 processes solved by using the unary encoding but just 9 and 24 processes by using the binary encoding, respectively. The unary encoding is more competitive than binary on all protocols. We identify two reasons for unary's better performance.

The first reason is the difference in expressing the transition in FSM. Though there are more variables used to encode the places, the transition relations can be generated by shifting the variables. The logics for transitions are simpler than binary, and the SAT problem is easy to solve.

The second reason is the use of the IC3 algorithm. IC3 is an incremental inductive algorithm, which builds the overapproximation from the last SAT results incrementally. The unary encoding is more efficient for learning clauses.

Figure 3 shows a comparison of the B-bounded model checking algorithm with the incremental bounded model checking algorithm on five parameterized protocols. The incremental bounded model checking algorithm is competitive on all instances, especially for Firefly, Illinois, and Berkeley. The main reason is that the B-bounded algorithm cannot

Table 2: The maximum process number solved by different encodings. As presented in Section 4.3, one-hot encoding is used for unary. Six parameterized protocols are used to test the performance, and the maximum process number is represented in the table. All results are collected by running Algorithm 1.

Protocols	Unary	Binary
Illinois	160	22
Firefly	160	18
Berkeley	146	18
German	97	5
CSMbroad	93	9
Dragon	90	24

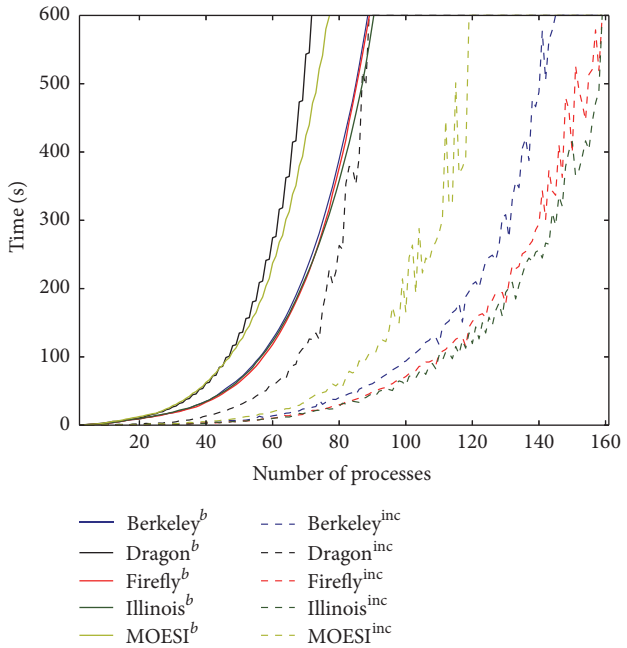

Figure 3: The performance comparison of B-bounded with incremental bounded model checking algorithm. The superscripts b and inc represent B-bounded model checking algorithm and incremental bounded model checking algorithm, respectively.

reuse the previous result to prove the current verification problem. Algorithm 1 gives us a good solution with little expenses when adding active literals for the generated FSM.

TDA [32, 33] uses X-abstract and Y-abstract to reduce the state space of parameterized systems and speed up the verification performance. Asynchronous composition is the key part of TDA, but it suffers from higher memory consumption. As shown in Table 3, PNPV performs better than TDA on all parameterized protocols. For Berkeley and Dragon protocols, TDA solves 14 processes, but PNPV solves 146 and 90 processes, respectively. PNPV handles 160 processes for Firefly and Illinois, but just 22 processes are solved by TDA. Table 3 shows that TDA suffers from high memory consumption, as most instances hit the memory limit before hitting the time limit. PNPV is also competitive with respect to speed when comparing with TDA, especially for large process numbers. As the results are all zero when the process number $n < 10$, we present the data from $n = 10$.

MIST is a tool to check safety properties against Petri net-like models. To compare with PNPV, we select the classical backward and EEC algorithms to run 12 Petri net benchmarks. Six out of 12 instances are bounded Petri nets. The total token numbers of the bounded Petri nets are limited. All of the six bounded Petri nets are safe. To test PNPV's ability of bug finding, six unsafe instances were collected from the MIST toolkit to evaluate the performance.

As shown in Table 4, PNPV performs better than both backward and EEC algorithms for all unsafe instances. Both in time and memory usage, PNPV is competitive. For six bounded Petri nets, PNPV wins on four out of six instances.

6. Conclusion and Future Works

We introduced an incremental bounded model checking algorithm to verify the safety properties of parameterized protocols in mobile CPS. By using counter abstraction, the protocol is modeled as a Petri net. Then the state-of-the-art SAT-based model checking algorithm is used to verify the safety properties. The algorithm can be used to verify parameterized systems, including cache coherence protocols, mutual exclusion communication protocols, and common concurrency primitives in mobile CPSs. The results show that our new approach can greatly scale the verification capabilities compared favorably against several recently published approaches. Due to using IC3 as the back-end model checking algorithm, our method is significant for its lower memory consumption.

There are two directions to extend the current work in the future. The first one is to study the property to be verified. Liveness would be an interesting direction, as the liveness property can be converted into safety. Security problems and run time verification would also be a good direction in the future. The second one is to model more complex systems. The ideas we have presented are naturally applicable to other concurrency systems modeled by Petri net or its extension. It is natural to shift SAT solver to SMT solver, and it would be a good way to improve the scalability.

Table 3: Comparison of running time and memory consumption between PNPV and TDA. *n* is the number of processes, which is the parameter of the parameterized protocols. *Max* stands for the maximum number of solved processes. Memory consumption is in MB and running time in seconds.

	Berkeley				Dragon				Firefly				Illinois			
n	TDA		PNPV		TDA		PNPV		TDA		PNPV		TDA		PNPV	
	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.
10	0.97	16.00	0.62	0.00	1.03	15.50	0.60	0.00	0.59	0.00	0.37	0.00	0.04	0.00	0.49	0.00
11	1.69	46.20	0.66	0.00	0.75	46.20	0.72	0.00	0.61	0.00	0.41	0.00	0.06	0.00	0.54	0.00
12	3.03	130.8	0.71	0.00	3.11	135.1	0.90	6.10	0.64	0.00	0.47	0.00	0.09	0.00	0.60	0.00
13	6.03	416.0	0.78	0.00	8.07	431.4	1.08	6.80	0.69	0.00	0.52	0.00	0.14	0.00	0.67	0.00
14	19.55	1168	0.85	0.00	24.47	1210	0.37	7.10	0.77	0.00	0.60	0.00	0.22	0.00	0.75	0.00
15	Memout		0.94	0.00	Memout		0.64	7.60	0.95	7.90	0.67	0.00	0.42	7.40	0.84	0.00
16	Memout		1.06	0.00	Memout		1.07	8.10	1.24	19.10	0.79	0.00	0.74	18.6	0.96	0.00
17	Memout		0.20	5.00	Memout		0.48	8.70	1.14	40.60	0.89	0.00	1.59	40.60	1.10	5.20
18	Memout		0.37	5.30	Memout		1.13	9.80	1.56	76.20	1.04	5.20	2.25	78.20	0.25	5.20
19	Memout		0.56	5.60	Memout		0.72	10.20	3.68	161.1	0.18	5.40	3.38	161.0	0.42	5.40
20	Memout		0.79	5.90	Memout		1.67	10.90	6.77	311.8	0.39	5.80	6.81	315.4	0.64	5.80
21	Memout		1.04	6.10	Memout		1.52	11.50	13.75	638.2	0.57	5.90	14.33	644.9	0.89	6.00
22	Memout		0.35	6.50	Memout		1.49	12.60	29.61	1249	0.84	6.20	27.28	1224	1.17	6.20
23	Memout		0.70	6.80	Memout		1.98	13.20	Memout		1.09	6.60	Memout		0.49	6.80
Max	14		146		14		90		22		160		22		160	

Table 4: Comparison of running time and memory consumption for different algorithms on Petri net benchmarks. Memory consumption is in MB, and running time is in seconds.

Problem	PNPV		Backward		EEC	
Instance	Time	Mem.	Time	Mem.	Time	Mem.
			Safe instances			
kanban	7.39	6.60	427.62	54.90	**0.83**	**0.00**
lamport	**0.42**	0.00	0.63	0.00	0.84	0.00
newdekker	**0.55**	3.30	0.71	0.00	0.85	0.00
newrtp	**0.58**	0.00	0.72	0.00	0.86	0.00
peterson	**0.68**	0.00	0.75	0.00	0.86	0.00
read-write	2.31	4.50	**0.81**	0.00	0.91	0.00
			Unsafe instances			
kanban	**1.05**	**5.20**	459.92	54.90	Timeout	
leabasicapproach	**0.08**	0.00	0.93	0.00	0.08	0.00
pingpong 2	**0.10**	0.00	0.93	0.00	0.13	0.00
pingpong_wrong	**0.13**	0.00	0.94	0.00	0.20	0.00
pncsacover	**6.22**	**4.90**	Timeout		23.89	8.70
pncsasemiliv	**1.06**	**4.40**	1.40	6.00	23.42	8.70

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to acknowledge that this work was supported by the National Natural Science Foundation of China (Grant no. 61133007). The authors thank Carl Kwan for helpful and detailed comments and suggestions.

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference (DAC '10)*, pp. 731–736, New York, NY, USA, June 2010.

[2] M. Conti, S. K. Das, C. Bisdikian et al., "Looking ahead in pervasive computing: challenges and opportunities in the era of cyberphysical convergence," *Pervasive and Mobile Computing*, vol. 8, no. 1, pp. 2–21, 2012.

[3] X. Hu, K. Bai, J. Cheng et al., "MeDJ: multidimensional emotion-aware music delivery for adolescent," in *Proceedings of the World Wide Web*, pp. 793–794, Proceedings of the World Wide Web, 2017.

[4] J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, and D. C. Schmidt, "R&D challenges and solutions for mobile cyber-physical applications and supporting internet services," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 45–56, 2010.

[5] L. Zhou, X. Hu, E. C.-H. Ngai et al., "A dynamic graph-based scheduling and interference coordination approach in heterogeneous cellular networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3735–3748, 2016.

[6] X. Hu, T. H. S. Chu, V. C. M. Leung, E. C.-H. Ngai, P. Kruchten, and H. C. B. Chan, "A Survey on mobile social networks: applications, platforms, system architectures, and future research directions," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1557–1581, 2015.

[7] X. Hu, J. Zhao, B.-C. Seet, V. C. M. Leung, T. H. S. Chu, and H. Chan, "S-aframe: agent-based multilayer framework with context-aware semantic service for vehicular social networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 44–63, 2015.

[8] S. M. German and A. P. Sistla, "Reasoning about systems with many processes," *Journal of the Association for Computing Machinery*, vol. 39, no. 3, pp. 675–735, 1992.

[9] R. Bloem, S. Jacobs, A. Khalimov et al., "Decidability in parameterized verification," *Synthesis Lectures on Distributed Computing Theory*, vol. 6, no. 1, pp. 1–170, 2015.

[10] W. Hunt, "Modeling and verification of cyber-physical systems," in *Proceedings of the National Workshop on High-Confidence Automotive Cyber-Physical Systems*, 2008.

[11] M. U. Sanwal and O. Hasan, "Formal verification of cyber-physical systems: coping with continuous elements," in *Proceedings of the International Conference on Computational Science and Its Applications*, pp. 358–371, Springer, 2013.

[12] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.

[13] K. L. McMillan, "Symbolic model checking," in *Symbolic Model Checking*, pp. 25–60, Springer, 1993.

[14] A. Pnueli, J. Xu, and L. Zuck, "Liveness with $(0, 1, \infty)$-counter abstraction," in *Proceedings of the International Conference on Computer Aided Verification*, pp. 107–122, Springer, Berlin, Germany, 2002.

[15] E. Clarke, M. Talupur, and H. Veith, "Environment abstraction for parameterized verification," in *Proceedings of the International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 126–141, Springer, 2006.

[16] D. Xu and Y. Deng, "Modeling mobile agent systems with high level Petri nets," in *Proceedings of the 2000 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 3177–3182, October 2000.

[17] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, "Petri net modeling of cyber-physical attacks on smart grid," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 741–749, 2011.

[18] A. R. Bradley, "SAT-based model checking without unrolling," in *Proceedings of the International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 70–87, Springer.

[19] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proceedings of the Formal Methods in Computer-Aided Design FMCAD '11*, pp. 125–134, November 2011.

[20] R. Akella and B. M. McMillin, "Model-checking BNDC properties in Cyber-physical systems," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference COMPSAC '09*, pp. 660–663, July 2009.

[21] L. Bu, Q. Wang, X. Chen et al., "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *ACM SIGBED Review*, vol. 8, no. 2, pp. 7–10, 2011.

[22] K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky, "Designing and verifying distributed cyber-physical systems using Multirate PALS: an airplane turning control system case study," *Science of Computer Programming*, vol. 103, pp. 13–50, 2015.

[23] X. Hu, T. H. S. Chu, H. C. B. Chan, and V. C. M. Leung, "Vita: a crowdsensing-oriented mobile cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, pp. 148–165, 2013.

[24] X. Hu, X. Li, E. C.-H. Ngai, V. C. M. Leung, and P. Kruchten, "Multidimensional context-aware social network architecture for mobile crowdsensing," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 78–87, 2014.

[25] F. C. Nemtanu, I. M. Moise, M. G. Beldescu, and V. Iordache, "Model of cloudified traveller information system based on petri nets," in *Proceedings of the 36th International Spring Seminar on Electronics Technology Automotive Electronics*, 2013.

[26] S.-Z. Zhang, Z.-H. Ding, and J.-L. Hu, "Modeling fault tolerated mobile agents by colored petri nets," in *International Conference on Intelligent Computing*, pp. 607–617, Springer, 2015.

[27] M. C. Browne, E. M. Clarke, and O. Grumberg, "Reasoning about networks with many identical finite state processes," *Information and Computation*, vol. 81, no. 1, pp. 13–31, 1989.

[28] K. L. McMillan, "Parameterized verification of the flash cache coherence protocol by compositional model checking," in *Proceedings of the Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pp. 179–195, Springer, Berlin Heidelberg, Germany.

[29] J. Esparza, P. Ganty, and R. Majumdar, "Parameterized verification of asynchronous shared-memory systems," *Journal of the ACM*, vol. 63, no. 1, article 10, 2016.

[30] C. Munoz, V. Carreño, and G. Dowek, "Formal analysis of the operational concept for the small aircraft transportation system," in *Rigorous Development of Complex Fault-Tolerant Systems*, pp. 306–325, Springer Berlin Heidelberg, 2006.

[31] T. T. Johnson and S. Mitra, "Parametrized verification of distributed cyber-physical systems: an aircraft landing protocol case study," in *Proceedings of the IEEE/ACM 3rd International Conference on Cyber-Physical Systems ICCPS '12*, pp. 161–170, April 2012.

[32] Y. Guo, W. Qu, L. Zhang, and W. Xu, "State space reduction in modeling checking parameterized cache coherence protocol by two-dimensional abstraction," *Journal of Supercomputing*, vol. 62, no. 2, pp. 828–854, 2012.

[33] L. Zhang, W. Qu, Y. Guo, and S. Li, "Automatic abstraction for verification of parameterized systems," *Journal of Computer-Aided Design and Computer Graphics*, vol. 26, no. 6, pp. 991–998, 2014.

[34] G. Geeraerts, "On the expressive power of petri nets with transfer arcs vs. petri nets with reset arcs," Tech. Rep. 572, Université Libre de Bruxelles, 2007.

[35] G. Delzanno, "Automatic verification of parameterized cache coherence protocols," in *Proceedings of the International Conference on Computer Aided Verification*, pp. 53–68, 2000.

[36] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!," *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63–92, 2001.

[37] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *Proceedings of the International conference on tools and algorithms for the construction and analysis of systems*, vol. 1579, pp. 193–207, Springer.

[38] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proceedings of the International Conference on Computer Aided Verification*, vol. 2725, pp. 1–13, Springer.

[39] N. Sorensson and N. Een, "Minisat v1.13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, no. 53, 2005.

[40] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay, "General decidability theorems for infinite-state systems," in *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science LICS'96*, pp. 313–321, 1996.

[41] G. Geeraerts, J.-F. Raskin, and L. Van Begin, "Expand, enlarge and check: new algorithms for the coverability problem of WSTS," *Journal of Computer and System Sciences*, vol. 72, no. 1, pp. 180–203, 2006.