Computing and Informatics, Vol. 40, 2021, 1174–1196, doi: 10.31577/cai_2021_5_1174

VERIFYING COMPUTATION TREE LOGIC OF KNOWLEDGE VIA KNOWLEDGE-ORIENTED PETRI NETS AND ORDERED BINARY DECISION DIAGRAMS

Leifeng He, Guanjun Liu*

Department of Computer Science
Tongji University
201804 Shanghai, China
e-mail: {1710052, liuguanjun}@tongji.edu.cn

Abstract. Computation Tree Logic of Knowledge (CTLK) can specify many requirements of privacy and security of multi-agent systems (MAS). In our previous papers, we defined Knowledge-oriented Petri Net (KPN) to model MAS, proposed similar reachability graph to verify CTLK, gave their model checking algorithms and developed a related tool. In this paper, we use the technique of Ordered Binary Decision Diagrams (OBDD) to encode similar reachability graph in order to alleviate the state explosion problem, and verify more epistemic operators of CTLK. We design the corresponding symbolic model checking algorithms and improve our tool. We compare our model and method with MCMAS that is the state-of-the-art CTLK model checker, and experiments illustrate the advantages of our model and method. We also explain the reasons why our model and method can obtain better performances.

Keywords: Epistemic logic, model checking, Petri nets, multi-agent systems, OBDD, CTLK

1 INTRODUCTION

Errors in some privacy/security-critical systems such as privacy protocols can result in drastic consequences. Therefore, it is necessary for designers to use some logically

* Corresponding author

precise approaches to verify the correctness of these systems. *Model checking* [1, 2, 16] is an automated and practically successful approach of formally verifying these systems.

In the paradigm of model checking, a system S is described by a transition system [1], a Petri net [3, 4, 13, 22], or a program written in a dedicated modelling language such as reactive module [5], NUSMV (New Symbolic Model Verifier) [8] and ISPL (Interpreted Systems Programming Language) [21], called M_S. A requirement (or property) P of the system is specified by a logical formula ϕ_P. Verifying whether system S satisfies requirement P is encoded as the problem of checking whether model M_S satisfies formula ϕ_P, formally written as $M_S \models \phi_P$. Some requirements such as deadlock-freeness, safety, liveness and fairness, can be encoded by discrete temporal logics such as *Linear Temporal Logic* (LTL) [2, 6] and *Computation Tree Logic* (CTL) [2, 15].

A big challenge in model checking is the *state explosion problem*, i.e., the state space of a system grows exponentially with the number of variables. Many techniques have been developed to deal with this problem such as *Ordered Binary Decision Diagrams* (OBDD) [9], abstraction [32] and partial order reduction [17].

A *Multi-Agent System* (MAS) [27] is a distributed system in which multiple agents interact or collaborate together to perform a set of common and private tasks. Both the correctness of interacting/collaborating behaviors and the privacy/security of agents are important for MAS. Temporal logic can only specify some requirements of interacting/collaborating behaviors, but cannot specify the requirements of privacy/security. Therefore, *epistemic logic* [23] is considered in model checking so that the requirements of privacy/security can be specified and verified.

Epistemic logic comes from the philosophy field, and has been used in the computer science field as a means of reasoning about the knowledge and belief of agents in MAS [21, 28]. It is a modal logic concerned with agent-related reasoning and offers a useful analysis of privacy/security. It has been used for checking *agreement protocols* [26], *security protocols* [18] and other knowledge-related MAS [21]. By adopting epistemic modalities as primitives, one can naturally express private and collective (common or distributed) knowledge of agents. *Computation Tree Logic of Knowledge* (CTLK) [28] is an extension of CTL with epistemic logic so that it can specify the requirements of both the interaction/collaboration and the privacy/security of agents. *Bit Transmission Protocol* [29] and *Dining Cryptographers Protocol* [14] can be viewed as an MAS and their temporal-epistemic requirements can be specified by CTLK. Algorithms and tools have been developed for the related model checking [7, 8, 21].

MCMAS [21] is a state-of-the-art model checker that can model MAS by ISPL and verify CTLK based on Kripke model. However, it usually has three weaknesses:

1. programs written by ISPL are usually non-intuitive and have poor readability, which is also pointed out in [8];
2. when ISPL models an MAS, it usually needs an environment agent to handle interaction/collaboration of all agents, which results in poor scalability;

3. it has a high time complexity when generating Kripke model to verify CTLK, especially when using OBDD to compress the state space.

Petri net is a natural and intuitive modelling language to describe concurrent or distributed system. There are some studies on Petri-nets-based model checking for MAS [3, 4, 13, 22], but they only pay attention to the correctness of interacting/collaborating behaviors but do not consider privacy/security. Therefore, in our previous papers [19, 20], we defined *Knowledge-oriented Petri Nets* (KPN) to model both the process of interaction/collaboration of multiple agents and their epistemic evolutions. These epistemic evolutions are closely related to privacy/security of MAS. We used CTLK [28] with two epistemic operators to specify the requirements related to both interaction/collaboration and privacy/security, and defined similar reachability graph to verify them. We gave the corresponding model checking algorithms and developed a related tool. In this paper, we consider the other two epistemic operators of CTLK, use OBDD [9, 10] as the symbolic representation of similar reachability graph to alleviate the state explosion problem give some rules to construct a fixed variable order of OBDD. Then we design their symbolic model checking algorithms and improve our tool. Our model and method can well overcome the above three weaknesses of MCMAS and experiments show their effectiveness.

The remainder of this paper is organized as follows. Section 2 introduces some basic concepts of Petri nets. Section 3 introduces KPN and similar reachability graph. Section 4 recalls OBDD and presents a symbolic approach to encode and produce the similar reachability graph of a given KPN. Section 5 introduces the syntax and semantics of CTLK. Section 6 proposes our model checking algorithms. Section 7 introduces our tool and experiments. Section 8 concludes this paper.

2 PETRI NETS

Petri nets are recalled in this section. For more details, one can refer to [24] and [25]. $\mathbb{N} = \{0, 1, 2, \dots\}$ is the set of all non-negative integers. Given $m \in \mathbb{N}$ and $m > 0$, we denote $\mathbb{N}_m = \{1, 2, \dots, m\}$.

A *net* is a 3-tuple $N = (P, T, F)$ where P is a finite set of *places*, T a finite set of *transitions*, $F \subseteq (P \times T) \cup (T \times P)$ a set of *arcs*, and $P \cap T = \emptyset$. A net may be seen as a directed bipartite graph. Generally, a transition is represented by a rectangle and a place by a circle in a net graph. Given a net $N = (P, T, F)$ and a node $x \in P \cup T$, ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ and $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$ represent the *pre-set* and *post-set* of x, respectively.

A *marking* of $N = (P, T, F)$ is a mapping $M\colon P \to \mathbb{N}$ where $M(p)$ is the number of tokens in place p. Place p is *marked* at M if $M(p) > 0$. Notice that a marking is denoted as a set of marked places in this paper. For example, if the marking M satisfies that place p_1 has one token, place p_2 has 1 tokens and other places have no tokens, then it is written as $M = p_1 + p_2$ or $M = \{p_1, p_2\}$.

A net N with an *initial marking* M_0 is a *Petri net* or *net system* and denoted as (N, M_0). Transition t is *enabled* at M if $\forall p \in {}^\bullet t\colon M(p) > 0$, which is denoted as $M[t\rangle$.

Firing an enabled transition t yields a new marking M' and is denoted as $M[t\rangle M'$ where M' satisfies that $M'(p) = M(p)-1$ if $p \in^\bullet t \backslash t^\bullet$; $M'(p) = M(p)+1$ if $p \in t^\bullet \backslash {}^\bullet t$; and $M'(p) = M(p)$ otherwise. We call M as a *predecessor* of M'. A marking M_k is *reachable* from another marking M if $M_k = M$ or there exists a non-empty transition sequence $\sigma = t_1 t_2 \dots t_k$ such that $M[t_1\rangle M_1[t_2\rangle \dots \rangle M_{k-1}[t_k\rangle M_k$. $M[\sigma\rangle M_k$ represents that M reaches M_k after firing σ. The set of all markings reachable from M in a net N is denoted as $R(N, M)$.

The *reachability graph* of a Petri net is a 3-tuple $\Delta = (\mathbb{M}, T, \mathbb{F})$ where $\mathbb{M} = R(N, M_0)$ is the set of all reachable markings and $\mathbb{F} \subseteq \mathbb{M} \times T \times \mathbb{M}$ is the set of all directed edges such that $(M, t, M') \in \mathbb{F}$ iff $M[t\rangle M'$. In this paper, we omit transition names on all directed edges because they are not related to our model checking. Then $(M, M') \in \mathbb{F}$ iff $\exists t \in T$ such that $M[t\rangle M'$.

A Petri net is *safe* if each place has at most one token at each reachable marking. A transition t is *dead* at a marking M if $\forall M' \in R(N, M)$: $\neg M'[t\rangle$. A marking M is a *deadlock* if $\forall t \in T$, t is dead at M.

3 KNOWLEDGE-ORIENTED PETRI NETS

3.1 KPN

Definition 1 (KPN)**.** A *KPN* is a 7-tuples $\Sigma = (P_S, P_K, T, F, M_0, \mathcal{A}, L)$ where

1. $(P_S \cup P_K, T, F, M_0)$ is a safe Petri net;
2. $P_S \cap P_K = \emptyset$;
3. $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ is a set of names of agents;
4. $L : P_K \to 2^{\mathcal{A}}$ is a labeling function.

A KPN is a special Petri net in which the epistemic evolution of each agent is considered. For a KPN, P_S is called as a set of state places and the underlying Petri net w.r.t. P_S models the state transitions and interactions of multiple agents; P_K is called as a set of basic knowledge places, each $p \in P_K$ represents a basic knowledge obtained by one/multiple agents when it is marked and $L(p)$ means those agents who obtain the basic knowledge. Therefore, KPN can model both the interaction/collaboration process of multiple agents and their epistemic evolutions.

In this paper, we use a modified version of *bit transmission protocol* [19] to illustrate KPN and similar reachability graph. The KPN in Figure 1 models this protocol where the hollow circles are local state places, the solid circles are basic knowledge places, and agents a_1, a_2 and a_3 represent Senders 1, 2 and a Receiver, respectively. First, two Senders first compete to deliver a bit (i.e., 0 or 1) to a Receiver ($t_{1,1}$ or $t_{2,1}$). Second, the Receiver returns an acknowledgement once receiving a bit ($t_{3,1}$ and $t_{3,2}$). Finally, the acknowledgement is received by the Sender who has delivered a bit ($t_{1,2}$ or $t_{2,2}$). Places $p_{1,3}$, $p_{1,4}$, $p_{2,3}$, $p_{2,4}$, $p_{3,3}$ and $p_{3,5}$ are basic knowledge places. A token in $p_{1,3}$ (respectively $p_{2,3}$) means that Sender 1 (respectively Sender 2) has sent a bit.

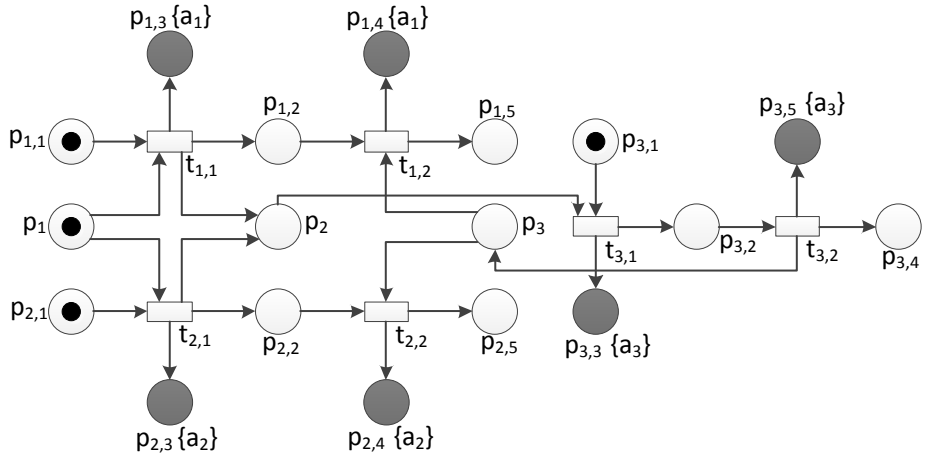


Figure 1. KPN modeling *bit transmission protocol*

A token in $p_{1,4}$ (respectively $p_{2,4}$) means that Sender 1 (respectively Sender 2) has received an acknowledgement. A token in $p_{3,3}$ (respectively $p_{3,5}$) means that the receiver has received a bit (respectively sent an acknowledgement).

3.2 Similar Reachability Graph

For marking M and a set of places $P \subseteq P_S \cup P_K$ in a KPN, $M \mid P$ denotes the projection of M onto P, i.e., $M \mid P = \{p \in P \mid M(p) > 0\}$. For agent a, we use P_a to represent those basic knowledge places w.r.t. agent a, i.e., $P_a = \{p \in P_K \mid a \in L(p)\}$. Obviously, $P_K = \bigcup_{i=1}^{m} P_{a_i}$. Then $M \mid P_K$ denotes the *basic knowledge* owned by all agents at M and $M \mid P_a$ denotes the *basic knowledge* owned by agent a at M.

Definition 2 (Similar reachability graph)**.** Given a KPN $\Sigma = (P_S, P_K, T, F, M_0, \mathcal{A}, L)$ where $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$, its *similar reachability graph* $\Delta = (\mathbb{M}, \mathbb{F}, \sim_{a_1}, \sim_{a_2}, \ldots, \sim_{a_m})$ is defined as follows:

1. $(\mathbb{M}, T, \mathbb{F})$ is the reachability graph of Petri net $(P_S \cup P_K, T, F, M_0)$; and
2. $\forall a \in \mathcal{A}$, $\sim_a \subseteq \mathbb{M} \times \mathbb{M}$ is a similar relation w.r.t. agent a such that $\forall M, M' \in \mathbb{M}$, $M \sim_a M'$ iff $M \mid P_a = M' \mid P_a$.

For each agent, a similar relation is constructed. If an agent owns the same basic knowledge at two markings, then the two markings are similar from its epistemic perspective. Obviously, a similar relation is reflexive, symmetric and transitive, i.e., it is an equivalence relation. Figures 2 a), 2 b) and 2 c) show the similar reachability graph of the KPN in Figure 1 w.r.t. agents a_1, a_2 and a_3, respectively. Here, those markings similar w.r.t. one agent are represented in the same line width. For example, there are 3 sets of markings similar w.r.t. agent a_1 in Figure 2 a) and one is

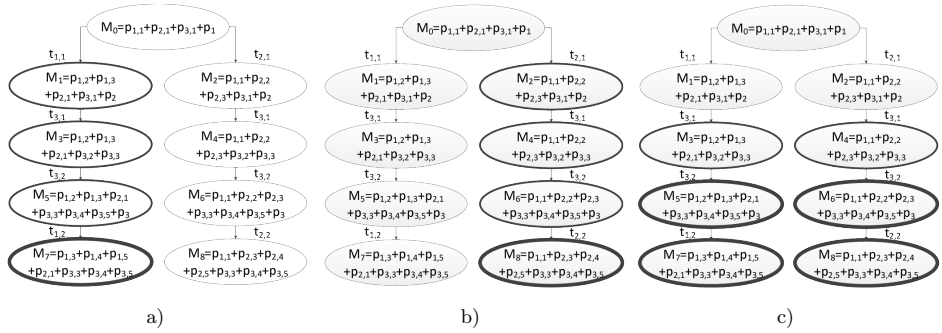


Figure 2. The similar reachability graph of the KPN in Figure 1

$\{M_0, M_2, M_4, M_6, M_8\}$, which means that agent a_1 has no basic knowledge at these markings. For agents a_2 and a_3, their similar relations are similar.

Given two equivalence relations $\sim_a$ and $\sim_b$, we give $\sim_a \cup \sim_b$, $\sim_a \cap \sim_b$ and $(\sim_a \cup \sim_b)^+$, i.e., $M(\sim_a \cup \sim_b)M'$ iff $M \sim_a M'$ or $M \sim_b M'$; $M(\sim_a \cap \sim_b)M'$ iff $M \sim_a M'$ and $M \sim_b M'$; $M(\sim_a \cup \sim_b)^+M'$ iff $\exists M_1, M_2, \ldots, M_x \in \mathbb{M}$ such that $M(\sim_a \cup \sim_b)M_1$, $M_1(\sim_a \cup \sim_b)M_2$, $\ldots$, $M_x(\sim_a \cup \sim_b)M'$. Consequently, $\sim_a \cap \sim_b$ and $(\sim_a \cup \sim_b)^+$ are still equivalence relations; but $\sim_a \cup \sim_b$ is not necessarily an equivalence relation because it is reflexive and symmetric but not necessarily transitive. These operations of binary relation are important for the definitions of epistemic operators in CTLK.

4 SYMBOLIC APPROACH OF PRODUCING SIMILAR REACHABILITY GRAPH USING OBDD

For (similar) reachability graph, there is the state explosion problem. In this paper, we use OBDD to handle this problem. This section first recalls OBDD [9, 10] and then presents a symbolic approach to encode and produce the similar reachability graph of a given KPN using OBDD. The markings, state transitions and similar relations in a similar reachability graph can be symbolically modelled by Boolean functions and then these functions are represented by OBDD. Therefore, we can significantly reduce the storage space if OBDD has a good compressing effect. In this paper, all Boolean functions are represented by OBDD.

4.1 Ordered Binary Decision Diagrams

OBDD is recalled in this section. For more details, one may refer to [9]. Here, we only review some of its definitions for readability.

A Binary Decision Diagram (BDD) is a rooted, directed, and acyclic graph with two sink nodes labelled by 0 and 1 that represent Boolean functions **0** and **1**, respectively. Each non-sink node is labelled with a Boolean variable v and has two

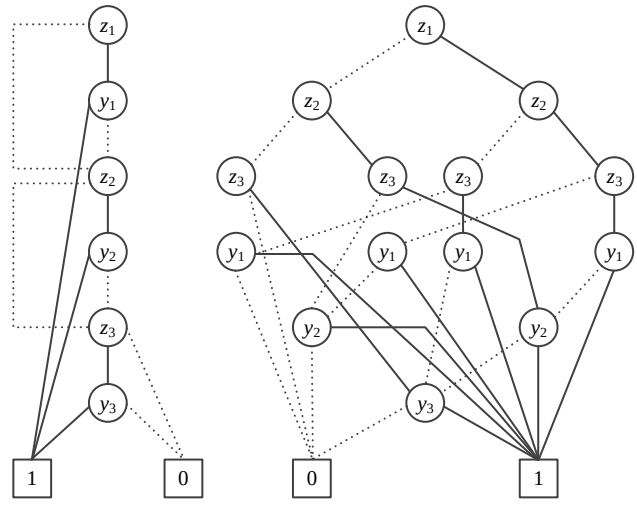


Figure 3. a) OBDD for function $f_3 = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$ in the variable order $z_1 < y_1 < z_2 < y_2 < z_3 < y_3$; b) OBDD for function $f_3 = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$ in the variable order $z_1 < z_2 < z_3 < y_1 < y_2 < y_3$

out-edges labelled by 1 (that represents *then*) and 0 (that represents *else*). Each non-sink node represents a Boolean function corresponding to its 1-edge if $v = 1$ or a Boolean function corresponding to its 0-edge if $v = 0$.

An OBDD is a BDD where all variables are totally ordered and each path from source node to a sink node visits these variables in the ascending order. A Reduced OBDD (ROBDD) is an OBDD where each node represents a distinct Boolean function and no variable node has identical 1-edge or 0-edge. ROBDD has some important properties. It provides compact representations of Boolean functions and there are efficient algorithms for performing all kinds of logical operations on ROBDD. They are all based on the crucial fact that an ROBDD has a canonical representation of a Boolean function: given a fixed variable order, there is exactly one ROBDD representing it for any Boolean function. Notice that we use the ROBDD technique in this paper, but for readability we still call it as OBDD in the next content.

OBDD can encode large sets of states with small data structures and can enable efficient manipulation of those sets. However, it is known that the size of an OBDD for a Boolean function seriously depends on the chosen variable order [11] and an improper variable order can still result in the node explosion problem, i.e., the number of nodes in an OBDD grows exponentially with the number of variables. To illustrate it, we consider a simple Boolean function $f_m = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee \cdots \vee (z_m \wedge y_m)$ [1]. Figures 3 a) and 3 b) show the OBDD in a linear-size variable order and the OBDD in an exponential-size variable order for $m = 3$, respectively. In Figure 3, solid lines

mean that the values of the corresponding variables are 1 and dotted lines mean that the values of the corresponding variables are 0.

To find an optimal variable order is an NP-hard problem [31], and thus a policy of dynamically reordering variables is often taken. For example, MCMAS takes such a policy but it often has a very high time complexity since it frequently looks for a good compromise between continuous variable reordering and efficiency of reducing memory consumption. In this paper, we use a fixed variable order instead of dynamically reordering variables. Therefore, our method of producing a similar reachability graph encoded by OBDD can save lots of time and space. In this paper, we use the OBDD-package in the CUDD library [10] developed by Fabio Somenzi at Colorado University.

4.2 Symbolic Approach of Producing Similar Reachability Graph

We now present our symbolic approach to encode and produce the similar reachability graph of a given KPN using OBDD. Our approach improves the corresponding one for bounded Petri nets [12] but we also need to encode similar relations. We use OBDD and some of its operations to encode and produce all reachable markings, state transitions and similar relations in a similar reachability graph. These OBDD operations include $\cdot$, $+$, $-$ and $\leftarrow$. Operations $\cdot$, $+$ and $-$ can be viewed as the intersection, union and subtraction of two sets, respectively. Boolean function $f(X) \mid \{x \leftarrow x'\}$ means this Boolean function where variable x is replaced by its primed copy x'. Later, our model checking algorithms also use another OBDD operation $\restriction$. Boolean function $f(X) \restriction Y$ means this Boolean function where each variable $x \in X \backslash Y$ are removed. For example, given $f(x_1, x_2, x_3) = x_1\, x_2\, x_3 + \overline{x_1}\, \overline{x_2}\, \overline{x_3}$, $f(x_1, x_2, x_3) \restriction \{x_1, x_2\} = x_1\, x_2 + \overline{x_1}\, \overline{x_2}$.

Given a KPN where $P_S \cup P_K = \{p_1, p_2, \ldots, p_n\}$, since it is safe, we can use a place p to denote a Boolean variable and then a marking M is encoded by a Boolean function composed of variables p_1, p_2, $\ldots$, p_n. For a Boolean function, if a place variable does not occur then this function represents such all markings that the place is marked or unmarked. Then **true** represents all possible markings composed of all marked or unmarked places, p represents those possible markings where p is marked and $\overline{p}$ represents those possible markings where p is unmarked. Then a marking M is encoded by operation $\cdot$ of all these p or $\overline{p}$. If $M(p) = 1$ then we choose p; otherwise, we choose $\overline{p}$. For example, if there is a KPN where $P = \{p_1, p_2, p_3, p_4\}$, then marking $M = \{p_1, p_3\}$ can be represented by $p_1 \cdot \overline{p_2} \cdot p_3 \cdot \overline{p_4}$ (or $p_1\, \overline{p_2}\, p_3\, \overline{p_4}$ for short) which is true only if $p_1 = p_3 = 1 \wedge p_2 = p_4 = 0$. Similarly, a set of markings can be encoded by operation $+$ of the corresponding Boolean functions. For example, if $M_1 = p_1\, \overline{p_2}\, p_3\, \overline{p_4}$ and $M_2 = p_1\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}$, then $\{M_1, M_2\} = M_1 + M_2 = p_1\, \overline{p_2}\, p_3\, \overline{p_4} + p_1\, \overline{p_2}\, \overline{p_3}\, \overline{p_4} = p_1\, \overline{p_2}\, \overline{p_4}$ which is true only if $p_1 = 1 \wedge p_2 = p_4 = 0 \wedge (p_3 = 1 \vee p_3 = 0)$. Similarly, variables p_1, p_2, $\ldots$, p_n are used to encode the current marking and their primed copies p'_1, p'_2, $\ldots$, p'_n are used to encode the next marking; and then a relation between two markings (e.g. state transition or similar relation) is encoded by operation $\cdot$ of the corresponding Boolean functions.

Based on OBDD, transitions can be fired at a set of markings rather than using a traditional marking-per-marking basis. Given a subset $\mathbb{M}_x \subseteq \mathbb{M}$ and a transition $t \in T$, we give two functions: $Enable(t, \mathbb{M}_x) = \{M \in \mathbb{M}_x \mid M[t\rangle\}$ and $Img(t, \mathbb{M}_x) = \{M \in \mathbb{M} \mid \exists M' \in \mathbb{M}_x : M'[t\rangle M\}$. Based on the rules of enabling transitions in Petri nets, we can calculate $Enable(t, \mathbb{M}_x)$, i.e., $Enable(t, \mathbb{M}_x) = \mathbb{M}_x \cdot \prod_{p \in {}^\bullet t} p$. In this paper, $\prod$ represents successive $\cdot$ operation. For example, $p_1 \cdot p_2 \cdot p_3 = \prod_{i=1}^{3} p_i$. Then we can calculate $Img(t, \mathbb{M}_x)$ based on the rules of firing transitions in Petri nets. If $Enable(t, \mathbb{M}_x) = \emptyset$, then $Img(t, \mathbb{M}_x) = \emptyset$. Otherwise, we modify the values of variables in $Enable(t, \mathbb{M}_x)$, i.e., $p = \overline{p}$ for each $p \in {}^\bullet t \setminus t^\bullet$, $p = \overline{p}$ for each $p \in t^\bullet \setminus {}^\bullet t$ (i.e., $\overline{p}$ becomes p due to $\overline{\overline{p}} = p$), and p is unchanged otherwise. Finally, the modified result is $Img(t, \mathbb{M}_x)$.

Algorithm 1 Produce similar reachability graph encoded by OBDD

Input: KPN $\Sigma = (P_S \cup P_K, T, F, M_0, \mathcal{A}, L)$
Output: Similar reachability graph $\Delta = SRG(\Sigma)$ encoded by OBDD

$M_0 =$ **true**;
for (each $p_i \in P_S \cup P_K$) **do**
  if $(M_0(p) > 0)$ **then** $M_0 = M_0 \cdot p_i$;
  else $M_0 = M_0 \cdot \overline{p_i}$;
$Reached = From = M_0$;
repeat
  for (each $t \in T$) **do**
    $From = From + Img(t, From)$;
  $New = From - Reached$; $From = New$; $Reached = Reached + New$;
until $(New == 0)$;
$\mathbb{M} = Reached$; $\mathbb{M}' = Reached\,|_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\}$;
$\mathbb{F} = \mathbb{M} \cdot \mathbb{M}' \cdot \sum_{t \in T} \Big(\prod_{p_i \in {}^\bullet t} p_i \cdot \prod_{p_j \in t^\bullet} p_j' \cdot \prod_{p_k \notin {}^\bullet t \cup t^\bullet} (p_k \equiv p_k') \Big)$;
for (each $a \in \mathcal{A}$) **do**
  $\sim_a = \mathbb{M} \cdot \mathbb{M}' \cdot \prod_{p_i \in P_a} (p_i \equiv p_i')$;
return $(\mathbb{M}, \mathbb{F}, \sim_{a_1}, \sim_{a_2}, \ldots, \sim_{a_m})$;

Based on $Img(t, \mathbb{M}_x)$, we give Algorithm 1 of producing the similar reachability graph of a given KPN encoded by OBDD, where $p \equiv p'$ is true if and only if the values of p and p' are the same. First, $\mathbb{M}$ is efficiently calculated by using $Img(t, \mathbb{M}_x)$ in a symbolic traversal algorithm. Second, we get $\mathbb{M}'$ by renaming each variable $p \in P_S \cup P_K$ into its primed copy p' in $\mathbb{M}$. Finally, we use $\mathbb{M}$ and $\mathbb{M}'$ to encode state transition $\mathbb{F}$ and similar relation $\sim$ for each agent based on their definitions. For each $M \in \mathbb{M}$ and $M' \in \mathbb{M}'$, $(M, M') \in \mathbb{F}$ iff there is a transition $\exists t \in T$ such that p_i is marked in M for each $p_i \in {}^\bullet t$; p_j' is marked in M' for each $p_j \in t^\bullet$, and the tokens of other places are the same for M and M'; and $(M, M') \in \sim_a$ if and only if the tokens of places in P_a are the same for M and M'.

In this paper, we use a fixed place order. We consider the structure of a KPN, and propose some rules to construct this order. In OBDD, a marking or a set of markings is encoded by a Boolean function composed of variables $p_1, p_2, \ldots, p_n$. Firing a transition t will change the assignments of these places (i.e., p becomes $\overline{p}$ or $\overline{p}$ becomes p) that belong to $({}^\bullet t \cup t^\bullet) \setminus ({}^\bullet t \wedge t^\bullet)$. It will need more time for OBDD to compute a new Boolean function if the distance between the two variables in a variable order is longer. Therefore, we can combine the structure of Petri nets and reasonably arrange the order of places.

Two places are dependent if there is a transition which affects them [30]. Then we can conclude that the shorter the average distance between all dependent places in a variable order is, the better effect of compacting the state space OBDD has. Therefore, we propose the following rules:

1. For a subnet modelling an agent, dependent places should be as close as possible;
2. For the common places via which these subnets are connected, they should be at the middle of places of these subnets; and
3. The unprimed variable p should be close to its primed variable p'.

Therefore, we can obtain a good variable order through these rules and our experiments will evidence this idea. But for MCMAS, it is not easy to do so since its modelling language ISPL does not have an obvious or direct structure representation.

5 CTLK

We use CTLK [28] as the specification language of temporal-epistemic requirements in MAS. In general, when a kind of modelling language is used to describe MAS, the syntax of CTLK is based on this language (e.g. ISPL and KPN) and its induced model (e.g. Kripke model and similar reachability graph) is used to explain the semantics of CTLK. In this paper, we use KPN as the model to describe MAS. Then for CTLK, its syntax is based on KPN and its semantics is based on similar reachability graph.

Definition 3 (Syntax of CTLK)**.** Given a KPN $\Sigma = (P_S, P_K, T, F, M_0, \mathcal{A}, L)$, the syntax of CTLK is defined by the following existential normal form (ENF) expressions:

$$\phi ::= \mathbf{true} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid EX\,\phi \mid EG\,\phi \mid E(\phi\,U\,\phi) \mid \mathcal{K}_a \mid \mathcal{E}_\Gamma\,\phi \mid \mathcal{D}_\Gamma\,\phi \mid \mathcal{C}_\Gamma\,\phi$$

where $p \in P_S \cup P_K$, $a \in \mathcal{A}$ and $\Gamma \subseteq \mathcal{A}$.

Other operators can be derived from the above ones: $deadlock \stackrel{def}{=} \neg EX\,\mathbf{true}$; $\phi_1 \vee \phi_2 \stackrel{def}{=} \neg(\neg\phi_1 \wedge \neg\phi_2)$; $\phi_1 \rightarrow \phi_2 \stackrel{def}{=} \neg\phi_1 \vee \phi_2$; $AX\,\phi \stackrel{def}{=} \neg EX\,\neg\phi$; $EF\,\phi \stackrel{def}{=} E(\mathbf{true}\,U\,\phi)$; $A(\phi_1\,U\,\phi_2) \stackrel{def}{=} \neg E(\neg\phi_2\,U\,(\neg\phi_1 \wedge \neg\phi_2)) \wedge \neg EG(\neg\phi_2)$; $AG\,\phi \stackrel{def}{=} \neg EF\,\neg\phi$; $AF\,\phi \stackrel{def}{=} A(\mathbf{true}\,U\,\phi)$.

Given a similar reachability graph $\Delta = (\mathbb{M}, \mathbb{F}, \sim_{a_1}, \sim_{a_2}, \ldots, \sim_{a_m})$, a *computation* starting from a marking $M \in \mathbb{M}$ is a maximal sequence of markings, i.e., $\pi = (M^0, M^1, \ldots)$ such that $M^0 = M$ and $\forall i \in \mathbb{N}$: $(M^i, M^{i+1}) \in \mathbb{F}$. For an infinite computation $\pi = (M^0, M^1, \ldots)$, we denote $\pi(i) = M^i$ for each $i \in \mathbb{N}$. For a finite computation $\pi = (M^0, M^1, \ldots, M^n)$, $\pi(i) = M^i$ for each $i \leq n$, and $\pi(i) = \emptyset$ for each $i > n$. $\Pi(M)$ denotes the set of all computations starting from M.

Since a KPN is safe, we can use a place p in $P_S \cup P_K$ to represent one atomic proposition in CTLK. A marked place p means the value **true** of the corresponding atomic proposition and otherwise the value **false**.

Definition 4 (Semantics of CTLK)**.** Given a similar reachability graph $\Delta = (\mathbb{M}, \mathbb{F}, \sim_{a_1}, \sim_{a_2}, \ldots, \sim_{a_m})$, a marking $M \in \mathbb{M}$ and a CTLK formula ϕ, $(\Delta, M) \models \phi$ denotes that ϕ is true at M in Δ. Δ can be omitted when no ambiguity takes place. The relation $\models$ is defined inductively as follows:

- $M \models \mathbf{true}$;
- $M \models p$ iff $M(p) = 1$;
- $M \models \neg\phi$ iff $M \not\models \phi$;
- $M \models \phi_1 \wedge \phi_2$ iff $M \models \phi_1$ and $M \models \phi_2$;
- $M \models EX\,\phi$ iff there exists $\pi \in \Pi(M)$ such that $\pi(1) \neq \emptyset \wedge \pi(1) \models \phi$;
- $M \models EG\,\phi$ iff there exists $\pi \in \Pi(M)$ such that for all $i \geq 0$, if $\pi(i) \neq \emptyset$, then $\pi(i) \models \phi$;
- $M \models E(\phi_1\, U\, \phi_2)$ iff there exists $\pi \in \Pi(M)$ and $n \in \mathbb{N}$ such that $\pi(n) \neq \emptyset \wedge \pi(n) \models \phi_2$ and $\pi(j) \models \phi_1$ for each $j \in \{0, 1, \ldots, n-1\}$;
- $M \models \mathcal{K}_a\,\phi$ iff $\forall M' \in \mathbb{M} : M' \sim_a M \Rightarrow M' \models \phi$;
- $M \models \mathcal{E}_\Gamma\,\phi$ iff $\forall M' \in \mathbb{M} : M' \left(\bigcup_{a\in\Gamma} \sim_a\right) M \Rightarrow M' \models \phi$;
- $M \models \mathcal{D}_\Gamma\,\phi$ iff $\forall M' \in \mathbb{M} : M' \left(\bigcap_{a\in\Gamma} \sim_a\right) M \Rightarrow M' \models \phi$;
- $M \models \mathcal{C}_\Gamma\,\phi$ iff $\forall M' \in \mathbb{M} : M' \left(\bigcup_{a\in\Gamma} \sim_a\right)^+ M \Rightarrow M' \models \phi$.

Definition 5 (Validity)**.** A CTLK formula ϕ is valid in KPN Σ (denoted $\Sigma \models \phi$) iff its similar reachability graph Δ satisfies $(\Delta, M_0) \models \phi$.

As mentioned above, CTLK is an extension of CTL with epistemic logic. From Definition 4 we can see that the semantics of temporal operators are the same with those of CTL [2, 15, 16]. Here, we only explain epistemic operators.

The semantics of $\mathcal{K}_a$ means that agent a can derive that ϕ is true at marking M if and only if ϕ is true at each marking that has the same knowledge with M w.r.t. agent a, i.e., from its epistemic perspective, there was no doubt that ϕ is true at M.

The semantics of $\mathcal{E}_\Gamma$ means that each agent in a group of agents Γ can derive that ϕ is true at marking M, i.e., $\forall a \in \Gamma : M \models \mathcal{K}_a\,\phi$. Obviously, $\mathcal{K}_a\phi \stackrel{def}{=} \mathcal{E}_{\{a\}}\phi$.

The semantics of $\mathcal{D}_\Gamma$ means that it is a distributed knowledge in a group of agents Γ that ϕ is true at marking M, i.e., $M \models \mathcal{K}_{a_\Gamma}\,\phi$ where a_Γ is viewed as a special agent which owns all basic knowledge of those agents in Γ.

The semantics of $\mathcal{C}_\Gamma$ means that it is a common knowledge in a group of agents Γ that ϕ is true at marking M, i.e., $\forall a_{i_0}, a_{i_1}, a_{i_2}, \ldots, \in \Gamma$: $M \models \mathcal{K}_{a_{i_0}}\phi \wedge \mathcal{K}_{a_{i_1}}(\mathcal{K}_{a_{i_0}}\phi) \wedge \mathcal{K}_{a_{i_2}}(\mathcal{K}_{a_{i_1}}(\mathcal{K}_{a_{i_0}}\phi)) \wedge \ldots$, i.e., for each $j \in \mathbb{N}$: $M \models \mathcal{K}_{a_{i_j}}(\mathcal{K}_{a_{i_{j-1}}}(\ldots(\mathcal{K}_{a_{i_0}}\phi))) \Rightarrow M \models \mathcal{K}_{a_{i_{j+1}}}(\mathcal{K}_{a_{i_j}}(\mathcal{K}_{a_{i_{j-1}}}(\ldots(\mathcal{K}_{a_{i_0}}\phi))))$.

For a KPN, each agent can gain some basic knowledge when its basic knowledge places are marked. Therefore, for each $p \in P_a$, $M \models \mathcal{K}_a p$ and $M \models \mathcal{E}_{L(p)}p$ hold iff $M(p) = 1$. For complex knowledge, we use CTLK to specify it and verify it by model checking. For example in Figure 1, when Receiver has received a bit, he knows that Sender 1 or Sender 2 sent it but cannot know who sent it. CTLK formula $\phi_1 = AG(p_{3,3} \rightarrow (K_{a_3}(p_{1,3} \vee p_{2,3}) \wedge \neg K_{a_3} p_{1,3} \wedge \neg K_{a_3} p_{2,3})$ specifies this case. Later, our tool will check its validity.

6 MODEL CHECKING ALGORITHMS OF CTLK

Since every CTLK formula can be translated into its ENF expression [1], we only consider the algorithms of verifying those formulas in Definition 3.

Our algorithms extend those of CTL in [16]. Given a KPN Σ and a CTLK formula ϕ, the basic verifying procedure mainly includes three steps:

1. The similar reachability graph Δ of Σ is produced by Algorithm 1;
2. The markings in Δ satisfying ϕ (i.e., $Sat(\Delta, \phi)$) are computed recursively;
3. It follows that $\Sigma \models \phi$ if $M_0 \in Sat(\Delta, \phi)$.

Given a similar reachability graph Δ, Algorithm 2 describes a high-level structure of recursively computing $Sat(\Delta, \phi)$. The algorithms Sat_{EX}, Sat_{EG}, Sat_{EU}, $Sat_{\mathcal{K}}$, $Sat_{\mathcal{E}}$, $Sat_{\mathcal{D}}$ and $Sat_{\mathcal{C}}$ are described in Algorithms 3, 4, 5, 6, 7, 8 and 9, respectively.

Algorithms 3, 4 and 5 are similar to our conference paper [19] and a main difference is that this paper uses OBDD to compute the predecessors of markings in a given set of markings (say X). Obviously, $\mathbb{F} \cdot (X \mid_{p_i \in P_S \cup P_K} \{p_i \leftarrow p_i'\})$ means these state transitions $\{(M, M') \in \mathbb{F} \mid M' \in X\}$ so $(\mathbb{F} \cdot (X \mid_{p_i \in P_S \cup P_K} \{p_i \leftarrow p_i'\})) \upharpoonright P_S \cup P_K$ means these predecessors of markings in X. Note that $(\mathbb{F} \cdot (\mathbb{M} \mid_{p_i \in P_S \cup P_K} \{p_i \leftarrow p_i'\})) \upharpoonright P_S \cup P_K = (\mathbb{F} \cdot \mathbb{M}') \upharpoonright P_S \cup P_K = \mathbb{F} \upharpoonright P_S \cup P_K$ means these markings that are not deadlocks, so $\mathbb{M} - (\mathbb{F} \upharpoonright P_S \cup P_K)$ means these deadlocks.

Algorithms 6 and 9 are similar to our conference papers [19, 20] and a main difference is that this paper uses OBDD to compute the markings similar to at least one marking in a given set of markings (say X) w.r.t. an agent (say a). Obviously, $\sim_a \cdot (X \mid_{p_i \in P_S \cup P_K} \{p_i \leftarrow p_i'\})$ means these similar relations $\{(M, M') \in \sim_a \mid M' \in X\}$ so $(\sim_a \cdot (X \mid_{p_i \in P_S \cup P_K} \{p_i \leftarrow p_i'\})) \upharpoonright P_S \cup P_K$ means these markings similar to at least one marking in X w.r.t. a.

Algorithm 7 shows the process of computing $Sat_{\mathcal{E}}(\phi, \Gamma)$. Because it is not easy to directly compute this set, we choose to compute its complement set, i.e., $\neg Sat_{\mathcal{E}}(\phi, \Gamma)$. First, we compute $Sat(\Delta, \neg\phi)$ (say X). Second, we search for the markings that are

Algorithm 2 $Sat(\Delta, \phi)$

Input: Similar reachability graph Δ, CTLK formula ϕ
Output: $\{M \in \mathbb{M} \mid (\Delta, M) \models \phi\}$

if (ϕ is **true**) **then return** $\mathbb{M}$;
if (ϕ is a place) **then return** $\mathbb{M} \cdot \phi$;
if (ϕ is $\neg\phi_1$) **then return** $\mathbb{M} - Sat(\Delta, \phi_1)$;
if (ϕ is $\phi_1 \wedge \phi_2$) **then return** $Sat(\Delta, \phi_1) \cdot Sat(\Delta, \phi_2)$;
if (ϕ is $EX\,\phi_1$) **then return** $Sat_{EX}\,\phi_1$;
if (ϕ is $EG\,\phi_1$) **then return** $Sat_{EG}\,\phi_1$;
if (ϕ is $E(\phi_1\,U\,\phi_2)$) **then return** $Sat_{EU}(\phi_1, \phi_2)$;
if (ϕ is $\mathcal{K}_a\,\phi_1$) **then return** $Sat_{\mathcal{K}}(\phi_1, a)$;
if (ϕ is $\mathcal{E}_\Gamma\,\phi_1$) **then return** $Sat_{\mathcal{E}}(\phi_1, \Gamma)$;
if (ϕ is $\mathcal{D}_\Gamma\,\phi_1$) **then return** $Sat_{\mathcal{D}}(\phi_1, \Gamma)$;
if (ϕ is $\mathcal{C}_\Gamma\,\phi_1$) **then return** $Sat_{\mathcal{C}}(\phi_1, \Gamma)$;

Algorithm 3 $Sat_{EX}(\phi)$

$X = Sat(\Gamma, \phi)$; $Y = (\mathbb{F} \cdot (X\,|_{p_i \in P_S \cup P_K}\,\{p_i \leftarrow p_i'\})) \restriction P_S \cup P_K$;
return Y;

Algorithm 4 $Sat_{EG}(\phi)$

$X = Z = Sat(\Gamma, \phi)$; $Y = \mathbb{M}$;
while $(X \neq Y)$ **do**
  $Y = X$; $X = Z \cdot (\mathbb{F} \cdot (X\,|_{p_i \in P_S \cup P_K}\,\{p_i \leftarrow p_i'\})) \restriction P_S \cup P_K$;
$Z_1 = (\mathbb{M} - (\mathbb{F} \restriction P_S \cup P_K)) \cdot Z$; $Y_1 = \emptyset$;
while $(Y_1 \neq Z_1)$ **do**
  $Y_1 = Z_1$; $Z_1 = Z_1 + (Z \cdot (\mathbb{F} \cdot (Z_1\,|_{p_i \in P_S \cup P_K}\,\{p_i \leftarrow p_i'\})) \restriction P_S \cup P_K)$;
return $Y = Y + Y_1$;

Algorithm 5 $Sat_{EU}(\phi_1, \phi_2)$

$X = Sat(\Gamma, \phi_2)$; $Z = Sat(\Gamma, \phi_1)$; $Y = \emptyset$;
while $(X \neq Y)$ **do**
  $Y = X$; $X = X + (Z \cdot (\mathbb{F} \cdot (X\,|_{p_i \in P_S \cup P_K}\,\{p_i \leftarrow p_i'\})) \restriction P_S \cup P_K)$;
return Y;

Algorithm 6 $Sat_{\mathcal{K}}(\phi, a)$

$X = Sat(\Delta, \neg\phi)$; $Y = (\sim_a \cdot (X\,|_{p_i \in P_S \cup P_K}\,\{p_i \leftarrow p_i'\})) \restriction P_S \cup P_K$;
return $\mathbb{M} - Y$;

Algorithm 7 $Sat_{\mathcal{E}}(\phi, \Gamma)$

$X = Y = Sat(\Delta, \neg\phi)$; $X = X\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\}$;
for (each $a \in \Gamma$) **do**
  $Y = Y + (\sim_a \cdot X) \upharpoonright P_S \cup P_K$;
return $\mathbb{M} - Y$;

Algorithm 8 $Sat_{\mathcal{D}}(\phi, \Gamma)$

$Y = Sat(\Delta, \neg\phi)$; $Y = \prod_{a \in \Gamma} \sim_a \cdot (Y\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\}) \upharpoonright P_S \cup P_K$;
return $\mathbb{M} - Y$;

similar to at least one marking in X w.r.t. one agent $a \in \Gamma$, i.e., $(\sim_a \cdot (X\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\})) \upharpoonright P_S \cup P_K$. The union of these markings is $\neg Sat_{\mathcal{E}}(\phi, \Gamma)$. The complement of this set is $Sat_{\mathcal{E}}(\phi, \Gamma)$.

Algorithm 8 shows the process of computing $Sat_{\mathcal{D}}(\phi, \Gamma)$. Similar to Algorithm 7, we also compute its complement set, i.e., $\neg Sat_{\mathcal{D}}(\phi, \Gamma)$. First, we compute $Sat(\Delta, \neg\phi)$ (say X). Second, we search for the markings that are similar to at least one marking in X w.r.t. each agent $a \in \Gamma$. Obviously, $\prod_{a \in \Gamma} \sim_a \cdot (X\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\})$ means these similar relations $\{(M, M') \mid X \in M'$ and $(M, M') \in \sim_a$ for each $a \in \Gamma\}$ so $\prod_{a \in \Gamma} \sim_a \cdot (X\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\})) \upharpoonright P_S \cup P_K$ means these markings similar to at least one marking in X w.r.t. each agent $a \in \Gamma$, i.e., $\neg Sat_{\mathcal{D}}(\phi, \Gamma)$. The complement of this set is $Sat_{\mathcal{D}}(\phi, \Gamma)$.

The complexity of our model checking algorithms consists of two parts. First, a similar reachability graph needs to be generated, but the number of markings possibly grows exponentially even though KPN is safe. Therefore, we use OBDD to encode these states instead of explicitly representing them. Second, we verify a CTLK formula ϕ based on the similar reachability graph. Given an uncompressed similar reachability graph with n markings, k pairs of state transitions and w pairs of similar relations, our CTLK model-checking problem can be determined in time $\mathcal{O}((n + k + w) \cdot |\phi|)$ where $|\phi|$ is the number of atomic propositions/operators in ϕ. When we use OBDD to encode the similar reachability graph and verify ϕ, the complexity of the related algorithm depends on the size of OBDD, the operations of OBDD and $|\phi|$. Certainly, at the worst case the size of OBDD still grows exponentially [31] if the chosen variable order is

Algorithm 9 $Sat_{\mathcal{C}}(\phi, \Gamma)$

$X = Sat(\Delta, \neg\phi)$; $Y = \mathbb{M}$; $Z = \emptyset$;
while $(X \neq Y)$ **do**
  $Y = X$; $Z = X\ |_{p_i \in P_S \cup P_K}\ \{p_i \leftarrow p_i'\}$
  for (each $a \in \Gamma$) **do**
    $X = X + (\sim_a \cdot Z) \upharpoonright P_S \cup P_K$;
return $\mathbb{M} - Y$;

```
transition      preset      postset
t11             0,1.        3,5,6.
t21             1,2.        4,6,7.
t12             5,9.        8,11.
t22             7,9.        10,12.
t31             6,13.       14,15.
t32             15.         9,16,17.
@
place           name        token       label
0               p11         1           .
1               p1          1           .
2               p21         1           .
3               p13         0           a1.
4               p23         0           a2.
5               p12         0           .
6               p2          0           .
7               p22         0           .
8               p14         0           a1.
9               p3          0           .
10              p24         0           a2.
11              p15         0           .
12              p25         0           .
13              p31         1           .
14              p33         0           a3.
15              p32         0           .
16              p35         0           a3.
17              p34         0           .
@
```

a)

```
formula     operator    former_formula      latter_formula
1           p13         0                   0
2           p23         0                   0
3           or          1                   2
4           Ka3         0                   3
5           Ka3         0                   1
6           not         0                   5
7           Ka3         0                   2
8           not         0                   7
9           and         6                   8
10          and         4                   9
11          p33         0                   0
12          Rarrow      11                  10
13          AG          0                   12
@
```

b)

```
/cygdrive/d/kpner
heleifeng@DESKTOP-TLJ799U ~
$ cd /cygdrive/d/kpner

heleifeng@DESKTOP-TLJ799U /cygdrive/d/kpner
$ ./kpner btp.ppn
File is read!
All reachable markings are generated and execution time = 0 seconds
Here, the number of markings = 9
Here, The number of nodes of the max OBDD = 63
The number of markings satifying this formula = 9
All markings satisfy CTLK1 so CTLK1 is TRUE!
Execution time of verifying these formulas = 0.000000 seconds
Memory used by OBDD = 8.97122e+06 bytes

heleifeng@DESKTOP-TLJ799U /cygdrive/d/kpner
```

c)

Figure 4. a) The specification of the KPN in Figure 1; b) the specification of ϕ_1; c) the verification result

terrible, but lots of studies [12, 21] show that OBDD can work well at most of cases.

7 TOOL AND EXPERIMENTS

Based on our algorithms, we develop a model checker KPNer written in C++ programming language. After inputting a KPN and some CTLK formulas, our tool can output their verification results. This KPN and these formulas are stored in a .ppn file and our tool can read them. Figure 4 a) shows the specification of the KPN in Figure 1, (b) shows the specifications of formulas $\phi_1 = AG(p_{3,3} \rightarrow (K_{a_3}(p_{1,3} \vee p_{2,3}) \wedge \neg K_{a_3} p_{1,3} \wedge \neg K_{a_3} p_{2,3})$, and (c) shows the verification result.

In what follows, we use Dining Cryptographers Protocol [14] as the benchmark to show the performance of KPNer, and compare KPNer and MCMAS. In this protocol, n ($n \geq 3$) cryptographers share a meal around a circular table, and either one of them or their employer pays for the meal. They would like to discover whether

one of them paid money without revealing the identity of this cryptographer. To this end, there is a coin between any two cryptographers. The coin is randomly tossed and the result can only be seen by its left and right cryptographers. This protocol requires that each cryptographer must say "same" or "different" of his left and right coins. If a cryptographer paid money, he tells a lie. Otherwise, he tells the truth. If there are odd "different", then there must be one cryptographer paid for the meal because it is impossible that all cryptographers tell the truth but there are odd "different". On the other hand, if there are even "different", then the employer paid money. Therefore, after all cryptographers say, every cryptographer can know whether one of them paid, but they cannot know who paid in case one of them did pay. Of course, they know their employer paid in case none of them did pay.

We verify the following two epistemic requirements which were also considered in [7, 8, 21]:

1. when each cryptographer has said, everyone either knows that their employer paid (i.e., no cryptographer paid), or knows that one cryptographer paid but he (not payer) cannot know who paid; and
2. when each cryptographer has said and no cryptographer paid, it is a common knowledge in all cryptographers that their employer paid.

Due to symmetry, we only consider cryptographer 1 when verifying the first requirement. The two requirements can be formalised by the following formulas:

$$\phi_2 = AG\left(\left(\bigwedge_{i=1}^{n} c^i_{said} \wedge \neg c^1_{paid}\right) \rightarrow \left(\mathcal{K}_{c_1}\, e_{paid} \vee \left(\mathcal{K}_{c_1}\left(\bigvee_{i=2}^{n} c^i_{paid}\right) \wedge \bigwedge_{i=2}^{n} \neg \mathcal{K}_{c_1}\, c^i_{paid}\right)\right)\right)$$

$\phi_3 = AG\left(\left(\bigwedge_{i=1}^{n} c^i_{said} \wedge \bigwedge_{i=1}^{n} \neg c^i_{paid}\right) \rightarrow \mathcal{C}_{\mathcal{A}}\, e_{paid}\right)$ where n is the number of cryptographers, c^i_{said} represents that Cryptographer i said "same" or "different", c^i_{paid} (respectively e_{paid}) represents that Cryptographer i (respectively the employer) paid, c_1 represents Cryptographer 1 and $\mathcal{A}$ represents all cryptographers.

The KPN in [19] models this protocol but it has a high degree of concurrency and the state space explodes seriously with increasing the number of cryptographers. Therefore, in our experiments we modify the structure of this KPN as follows:

1. each cryptographer first sees his left and right coins, then pays or does not pay money, and finally says "same" or "different";
2. all cryptographers execute the above processes in turn.

In other words, we reduce unnecessary concurrent structures as much as possible. Obviously, the modified KPN also conforms to this protocol. The results show that ϕ_2 and ϕ_3 are both valid in this protocol. Here, we only show the performance of KPNer by increasing the number of cryptographers. Table 1 shows the experimental results.

Table 1 shows that the fixed variable order of OBDD generated by our rules has a satisfying performance. For example, when the number of cryptographers is 100, OBDD in this order only uses 17 390 nodes to encode 1.007×10^{33} markings so that it only uses 4.108×10^8 bytes in the whole model checking process. Notice that CUDD cannot count up the number of markings in an OBDD when the OBDD is composed of more than 1 024 Boolean variables. Therefore, when the number of cryptographers is more than 100 (the number of places $> 1\,024$), we only know the number of nodes of an OBDD but cannot know the number of markings in the OBDD. Here, we use INF (infinite) to represent these numbers and |OBDD| to represent the number of nodes in the OBDD encoding $\mathbb{M}$.

From Table 1, we can see that the process of producing similar reachability graph spends much more time than the process of verifying CTLK formulas. For example, when the number of cryptographers is 160, it spends almost 1.1 hours to produce the similar reachability graph but it only spends 59.155 s to verify ϕ_2 and ϕ_3. Due to the good performance of our variable order, OBDD has a satisfying effect of compressing the state space. Therefore, we can efficiently produce similar reachability graph and thus KPNer achieves a good performance. Even when the number of cryptographers is 200, KPNer can still produce the similar reachability graph with about 10^{64}–10^{67} markings in 2.2 hours, and verify ϕ_2 and ϕ_3 in 100 s. Obviously, KPNer is able to handle larger number of cryptographers. Note that all experiments are conducted on a PC equipped with Intel(R) Core(TM) i5-2 400 CPU @ 3.10 GHz and RAM @ 4.00G.

We compare KPNer with the state-of-the-art CTLK model checker MCMAS. Table 2 shows the experimental results. Here, any run that exceeds 3 hours but cannot output a result is reported as a timeout. To present a fair comparison, the two model checkers both use the same CUDD version 2.5.1 [10]. The experimental results (i.e., time and space) of KPNer and MCMAS are for the whole model checking process, i.e. both generating the induced model (Kripke model or similar reachability graph) and verifying CTLK formulas. The results show that KPNer is much more efficient and needs less memory than MCMAS. MCMAS can only verify this protocol for the case of 36 cryptographers at most. Besides, the performance of MCMAS is unstable. For example, when the number of cryptographers is 24 or 25, MCMAS runs more than 3 hours but does not output any result. However, when the number is 26 or 27, MCMAS can output the results in 100 s. Fortunately, the performance of KPNer keeps stable. The time spent by KPNer slowly increases with increasing the number of cryptographers.

For MCMAS, it spends too much time to generate the induced model (i.e., Kripke model) from an ISPL program. We think that there are at least two reasons:

1. each state in Kripke model is global so that it needs to first generate the local state of each agent and then combine these local states to generate a global state according to its environment agent. Additionally, for each agent, when generating a new local state from a given local state, the agent considers not

No. (n) of Crypto-graphers	KPN				Similar Reachability Graph		Time to Verify CTLK (s)	Memory (bytes)
	$\|P\|$ $(10n+1)$	$\|T\|$ $(12n)$	$\|F\|$ $(87n-2)$	$\|\mathbb{M}\|$	$\|$OBDD$\|$ $(176n-210)$	Time (s)		
10	101	120	868	75 783	1 550	0.561	0.046	4.544×10^7
20	201	240	1 738	1.615×10^8	3 310	3.15	0.203	5.045×10^7
30	301	360	2 608	2.513×10^{11}	5 070	10.015	0.67	6.93×10^7
40	401	480	3 478	3.452×10^{14}	6 830	27.612	1.685	9.39×10^7
50	501	600	4 348	4.436×10^{17}	8 590	65.067	3.042	1.291×10^8
60	601	720	5 218	5.465×10^{20}	10 350	134.769	4.992	1.717×10^8
70	701	840	6 088	6.54×10^{23}	12 110	226.373	7.472	2.212×10^8
80	801	960	6 958	7.665×10^{26}	13 870	359.223	10.655	2.8×10^8
90	901	1 080	7 828	8.839×10^{29}	15 630	546.798	13.401	3.476×10^8
100	1 001	1 200	8 698	1.007×10^{33}	17 390	780.13	16.957	4.212×10^8
110	1 101	1 320	9 568	INF	19 150	1 140.383	23.742	5.03×10^8
120	1 201	1 440	10 428	INF	20 910	1 570.899	28.719	5.934×10^8
130	1 301	1 560	1 1308	INF	22 670	1 959.575	34.819	6.902×10^8
140	1 401	1 680	1 2178	INF	24 430	2 519.01	41.138	7.952×10^8
150	1 501	1 800	1 3048	INF	26 190	3 305.629	50.856	8.994×10^8
160	1 601	1 920	1 3918	INF	27 950	3 942.364	59.155	1.026×10^9
170	1 701	2 040	1 4788	INF	29 710	4 817.216	66.223	1.151×10^9
180	1 801	2 160	1 5658	INF	31 470	5 695.316	67.143	1.28×10^9
190	1 901	2 280	1 6528	INF	33 230	6 219.775	78.937	1.424×10^9
200	2 001	2 400	1 7398	INF	34 990	7 986.221	99.623	1.576×10^9

Table 1. Experimental results of KPNer for different numbers of cryptographers

No. (n) of Crypto-graphers	KPNer			MCMAS		
	No. of Markings	Memory (bytes)	Time (s)	No. of States	Memory (bytes)	Time (s)
10	75 783	4.544×10^7	0.608	45 056	1.729×10^7	3.803
11	167 943	4.911×10^7	0.764	98 304	1.676×10^7	1.599
12	368 647	5.166×10^7	0.967	212 992	1.803×10^7	0.81
13	802 823	5.369×10^7	1.217	458 752	2.032×10^7	1.462
14	1.737×10^6	4.912×10^7	1.418	983 040	2.306×10^7	2.347
15	3.736×10^6	4.985×10^7	1.715	2.097×10^6	2.36×10^7	2.348
16	7.995×10^6	5.015×10^7	1.95	4.456×10^6	2.302×10^7	11.351
17	1.704×10^7	5.176×10^7	2.262	9.437×10^6	1.405×10^8	23.538
18	3.618×10^7	5.467×10^7	2.59	1.992×10^7	1.023×10^9	11 971.3
19	7.655×10^7	4.776×10^7	2.933	4.194×10^7	2.482×10^8	420.914
20	1.615×10^8	5.045×10^7	3.369	8.808×10^7	8.83×10^7	18.494
21	3.397×10^8	5.322×10^7	3.822	1.845×10^8	2.724×10^8	1 137.51
22	7.13×10^8	5.824×10^7	4.398	3.859×10^8	6.909×10^8	3 321.47
23	1.493×10^9	5.926×10^7	4.913	8.053×10^8	1.998×10^8	273.433
24	3.121×10^9	6.053×10^7	5.537	Timeout		
25	6.51×10^9	6.056×10^7	6.395	Timeout		
26	1.356×10^{10}	6.085×10^7	7.004	7.248×10^9	9.038×10^7	50.229
27	2.819×10^{10}	6.392×10^7	7.612	1.503×10^{10}	1.26×10^8	96.39
28	5.852×10^{10}	6.172×10^7	8.376	3.114×10^{10}	4.177×10^8	5 695.47
29	1.213×10^{11}	6.833×10^7	9.579	6.442×10^{10}	6.213×10^8	15 707.7
30	2.513×10^{11}	6.93×10^7	10.716	Timeout		
31	5.197×10^{11}	6.924×10^7	12.105	2.749×10^{11}	3.365×10^8	6 366.01
32	1.074×10^{12}	7.127×10^7	13.431	5.669×10^{11}	1.785×10^8	199.838
33	2.216×10^{12}	7.372×10^7	14.959	1.168×10^{12}	6.545×10^8	37 996.9
34	4.57×10^{12}	7.672×10^7	15.864	Timeout		
35	9.415×10^{12}	7.853×10^7	18.579	Timeout		
36	1.938×10^{13}	8.176×10^7	19.858	1.017×10^{13}	5.584×10^8	27 315.8
37	3.986×10^{13}	8.431×10^7	22.167	Timeout		
38	8.191×10^{13}	8.72×10^7	24.476	Timeout		
39	1.683×10^{14}	9.06×10^7	26.848	Timeout		
40	3.452×10^{14}	9.39×10^7	30.607	Timeout		

Table 2. Experimental results comparing KPNer and MCMAS on dining cryptographers protocol

only its actions but also others' actions due to the synchronous semantics of ISPL. Therefore, it spends much time to generate a new state from a given state. Fortunately, KPNer only needs to check preset and postset of a transition when generating a new state from a given state, which can save much time;

2. MCMAS dynamically reorders variables when encoding and producing Kripke model via OBDD. It needs much time to find a satisfying variable order. This is also the reason why the performance of MCMAS is unstable. Fortunately, KPNer can utilize our rules to construct a fixed variable order so that much time can be saved too. These rules utilize the structure of KPN so its performance is also stable.

There are other two CTLK model checkers MCK [7] and MCTK [8]. As shown in [21], MCMAS has the best performance among them. Therefore, in this paper we only compare KPNer with MCMAS.

8 CONCLUSION

In the conference versions [19, 20] of this paper, we defined KPN and similar reachability graph, and used them to verify CTLK in which only two epistemic operators $\mathcal{K}$ and $\mathcal{C}$ were considered. In this paper, we consider the other two epistemic operators $\mathcal{E}$ and $\mathcal{D}$ in CTLK, use OBDD to encode a similar reachability graph and give some rules to construct a fixed variable order of OBDD so as to alleviate the state explosion problem well. As shown in our experiments, our model checking method is much more efficient than the state-of-the-art CTLK model checker MCMAS. Our future work includes:

1. we consider simplifying CTLK formulas and thus continually optimize our algorithms;
2. we explore more complex epistemic operators to verify more complex epistemic properties;
3. we plan to give a heuristic method of automatically generating a fixed variable order of OBDD; and
4. we extend KPN from safe Petri nets to bounded Petri nets to simulate MAS more precisely.

Acknowledgement

This paper is supported in part by the National Nature Science Foundation of China (Nos. 62032019 and 62172299), in part by the Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0100) and in part by the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Baier, C.—Katoen, J. P.: Principles of Model Checking. MIT Press, 2008.

[2] Wang, J.—Tepfenhart, W.: Formal Methods in Computer Science. Chapman and Hall/CRC Press, 2019.

[3] Deng, Y.—Wang, J.—Tsai, J. J. P.—Beznosov, K.: An Approach for Modeling and Analysis of Security System Architectures. IEEE Transactions on Knowledge and Data Engineering, Vol. 15, 2003, No. 5, pp. 1099–1119, doi: 10.1109/tkde.2003.1232267.

[4] Wang, J.—Li, D.: Resource Oriented Workflow Nets and Workflow Resource Requirement Analysis. International Journal of Software Engineering and Knowledge Engineering, Vol. 23, 2013, No. 5, pp. 677–693, doi: 10.1142/s0218194013400135.

[5] Alur, R.—Henzinger, T. A.—Mang, F. Y. C.—Qadeer, S.—Rajamani, S. K.—Tasiran, S.: MOCHA: Modularity in Model Checking. In: Hu, A. J., Vardi, M. Y. (Eds.): Computer Aided Verification (CAV 1998). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1427, 1998, pp. 521–525, doi: 10.1007/bfb0028774.

[6] Vardi, M. Y.: An Automata-Theoretic Approach to Linear Temporal Logic. In: Moller, F., Birtwistle, G. (Eds.): Logics for Concurrency. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1043, 1996, pp. 238–266, doi: 10.1007/3-540-60915-6_6.

[7] Gammie, P.—van der Meyden, R.: MCK: Model Checking the Logic of Knowledge. In: Alur, R., Peled, D. A. (Eds.): Computer Aided Verification (CAV 2004). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3114, 2004, pp. 479–483, doi: 10.1007/978-3-540-27813-9_41.

[8] Su, K.—Sattar, A.—Luo, X.: Model Checking Temporal Logics of Knowledge via OBDDs. The Computer Journal, Vol. 50, 2007, No. 4, pp. 403–420, doi: 10.1093/comjnl/bxm009.

[9] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, Vol. C-35, 1986, No. 8, pp. 677–691, doi: 10.1109/tc.1986.1676819.

[10] Somenzi, F.: CUDD: CU Decision Diagram Package – Release 2.5.1. Available at: `http://vlsi.colorado.edu/fabio/CUDD`, 2012.

[11] Rudell, R.: Dynamic Variable Ordering for Ordered Binary Decision Diagrams. Proceedings of 1993 International Conference on Computer Aided Design (ICCAD), 1993, pp. 42–47, doi: 10.1109/iccad.1993.580029.

[12] Pastor, E.—Cortadella, J.—Roig, O.: Symbolic Analysis of Bounded Petri Nets. IEEE Transactions on Computers, Vol. 50, 2001, No. 5, pp. 432–448, doi: 10.1109/12.926158.

[13] Celaya, J. R.—Desrochers, A. A.—Graves, R. J.: Modeling and Analysis of Multi-Agent Systems Using Petri Nets. Proceedings of 2007 IEEE International Conference on Systems, Man and Cybernetics, 2007, pp. 1439–1444, doi: 10.1109/icsmc.2007.4413960.

[14] Chaum, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. Journal of Cryptology, Vol. 1, 1988, No. 1, pp. 65–75, doi: 10.1007/bf00206326.

[15] Clarke, E. M.—Emerson, E. A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In: Kozen, D. (Ed.): Logics of Programs (Logics of Programs 1981). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 131, 1981, pp. 52–71, doi: 10.1007/bfb0025774.

[16] Clarke, E. M.—Grumberg, O.—Peled, D. A.: Model Checking. MIT Press, 1999.

[17] Clarke, E. M.—Grumberg, O.—Minea, M.—Peled, D.: State Space Reduction Using Partial Order Techniques. International Journal on Software Tools for Technology Transfer, Vol. 2, 1999, No. 3, pp. 279–287, doi: 10.1007/s100090050035.

[18] Halpern, J. Y.—Pucella, R.: Modeling Adversaries in a Logic for Security Protocol Analysis. In: Abdallah, A. E., Ryan, P., Schneider, S. (Eds.): Formal Aspects of Security (FASec 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2629, 2003, pp. 115–132, doi: 10.1007/978-3-540-40981-6_11.

[19] He, L.—Liu, G.: Model Checking CTLK Based on Knowledge-Oriented Petri Nets. Proceedings of the 21st International Conference on High Performance Computing and Communications (HPCC), 2019, pp. 1139–1146, doi: 10.1109/hpcc/smartcity/dss.2019.00161.

[20] He, L.—Liu, G.: Verifying Computation Tree Logic of Knowledge via the Similar Reachability Graphs of Knowledge-Oriented Petri Nets. Proceedings of the 2020 39th Chinese Control Conference (CCC), 2020, pp. 5026–5031, doi: 10.23919/ccc50068.2020.9188719.

[21] Lomuscio, A.—Qu, H.—Raimondi, F.: MCMAS: An Open-Source Model Checker for the Verification of Multi-Agent Systems. International Journal on Software Tools for Technology Transfer, Vol. 19, 2017, No. 1, pp. 9–30, doi: 10.1007/s10009-015-0378-x.

[22] Lopes, B.—Benevides, M.—Haeusler, E. H.: Reasoning about Multi-Agent Systems Using Stochastic Petri Nets. In: Bajo, J. et al. (Eds.): Trends in Practical Applications of Agents, Multi-Agent Systems and Sustainability. Springer, Cham, Advances in Intelligent Systems and Computing, Vol. 372, 2015, pp. 75–86, doi: 10.1007/978-3-319-19629-9_9.

[23] Meyer, J.-J. C.—van der Hoek, W.: Epistemic Logic for AI and Computer Science. Cambridge University Press, 2004.

[24] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol. 77, 1989, No. 4, pp. 541–580, doi: 10.1109/5.24143.

[25] Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, Berlin, Heidelberg, 2013, doi: 10.1007/978-3-642-33278-4.

[26] van Oorschot, P.: Extending Cryptographic Logics of Belief to Key Agreement Protocols. Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93), 1993, pp. 232–243, doi: 10.1145/168588.168617.

[27] Weiss, G. (Ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, 1999.

[28] Penczek, W.—Lomuscio, A.: Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), 2003, pp. 209–216, doi: 10.1145/860575.860609.

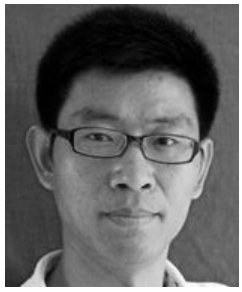
[29] Fagin, R.—Halpern, J. Y.—Moses, Y.—Vardi, M. Y.: Reasoning about Knowledge. MIT Press, 1995.

[30] Noack, A.: A ZBDD Package for Efficient Model Checking of Petri Nets. Ph.D. Thesis, BTU Cottbus, Department of Computer Science, 1999 (in German).

[31] Heiner, M.—Rohr, C.—Schwarick, M.: MARCIE – Model Checking and Reachability Analysis Done Efficiently. In: Colom, J. M., Desel, J. (Eds.): Applications and Theory of Petri Nets and Concurrency (PETRI NETS 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7927, 2013, pp. 389–399, doi: 10.1007/978-3-642-38697-8_21.

[32] Nouri, A.—Raman, B.—Bozga, M.—Legay, A.—Bensalem, S.: Faster Statistical Model Checking by Means of Abstraction and Learning. In: Bonakdarpour, B., Smolka, S. A. (Eds.): Runtime Verification (RV 2014). Springer, Cham, Lecture Notes in Computer Science, Vol. 8734, 2014, pp. 340–355, doi: 10.1007/978-3-319-11164-3_28.

Leifeng He received his B.Sc. degree from the University of Shanghai for Science and Technology, Shanghai, China, in 2015. He is currently working toward his Ph.D. degree in the Department of Computer Science and Technology, School of Electronics and Information Engineering, Tongji University, Shanghai, China. His research interests include model checking, Petri net, workflow, multi-agent systems, and epistemic logic.



Guanjun Liu received his Ph.D. degree in computer software and theory from the Tongji University, Shanghai, China, in 2011. He was Post-Doctoral Research Fellow with the Singapore University of Technology and Design, Singapore, from 2011 to 2013. He was Post-Doctoral Research Fellow with the Humboldt University of Berlin, Germany, from 2013 to 2014, funded by the Alexander von Humboldt Foundation. In 2013, he joined the Department of Computer Science of the Tongji University as Associate Professor, and now he serves there as Professor. He has (co-)authored over 100 papers including 23 ones in IEEE/ACM Transactions and two books. His research interests include Petri net theory, model checking, Web service, workflow, discrete event systems, machine learning and credit card fraud detection.