Symbolic Algorithmic Verification of Generalized Noninterference

Conghua Zhou
School of Computer Science and Telecommunication Engineering, Jiangsu University
Zhenjiang, 212013, China
e-mail: chzhou@ujs.edu.cn

Abstract

In this paper we propose an algorithmic verification technique to check generalized noninterference. Our technique is based on the counterexamples search strategy mainly which generating counterexamples of minimal length. In order to make the verification procedure terminate as soon as possible we also discuss how to integrate the window induction proof strategy in our technique. We further show how to reduce counterexamples search and induction proof to quantified propositional satisfiability. This reduction enables us to use efficient quantified propositional decision procedures to perform generalized noninterference checking.

Keywords: Generalized noninterference; Quantified propositional satisfiability; Multilevel security

I. Introduction

One of the typical problems in computer security is that confidential data needs to be protected from undesired accesses. A well known approach to face this problem is the Multilevel Security , which is a policy for managing objects at various levels of secrecy. In multilevel secure systems every object and every user is bound to a secrecy level and the information flow can be directed only from low users to higher users. The system achieves this aim by implementing access control policies. As remarked in [1] this solution is still not satisfactory. Access control policies are defined to serve this task by specifying which accesses are allowed for which users. However, access control methods can only restrict direct information flow. For example, information leakage over covert channels[2], [3] is not controllable by access control methods.

In [4], Goguen and Meseguer first introduced the notion of noninterference as a means to control both direct as well as indirect information flow in a deterministic system. In practice, however there are much nondeterministic systems. Therefore, D. McCullough in his work [5] proposed a new security property, called generalized noninterference, to characterize the confidentiality on nondeterministic systems. After that, many more definitions based on generalized noninterference have been proposed in the literature, such as noninference [6], [7], separability [6], restrictiveness [5], the perfect security property[7].

Above security properties are global requirements. Henceforth, their verification is usually a complex task. At present, to the best of our knowledge there are only a sound approach[8] to checking generalized noninterference. However the approach is not complete. In this paper, based on a QBF[9] solver we focus on presenting a sound and complete approach to verifying generalized noninterference. The quantified boolean formula problem (QBF) is a generalization of the boolean satisfiability problem in which both existential quantifiers and universal quantifiers can be applied to each variable. Our work is motivated by that the verification methods based on QBF solvers[10], [11], [12], [13] have been shown to push the envelope of functional verification in terms of both capacity and efficiency, as reported in several academic and industrial case studies[14], [15], [16]. The successful application of QBF solvers in formal verification due to dramatic improvements in QBF solver technology over the past decade. At present, several powerful QBF solvers which can handle quantified propositional formulas with thousands of variables.

We present a symbolic algorithmic approach to the verification of generalized noninterference. The basic concept of our algorithmic approach consists of two aspects: one aspect is to search for a counterexample of generalized noninterference in executions whose length is bounded by some integer k, the search works by mapping the problem of the existence of some counterexample of length k to the quantified propositional satisfiability problem; another aspect is to use the widow induction technique[17] to verify generalized noninterference, and the induction hypothesis is checked by a QBF solver.

Our algorithmic approach shown in Fig. 1. consists of three basic steps:

1. Check the Bound: Determine whether the bound reaches the pre-computed threshold. If so, then claim the system satisfies noninterference.
2. Search for Counterexamples by a QBF Solver: Reduce the existence problem of counterexamples of some length to the quantified propositional satisfiability problem, i.e., there exists counterexamples of length k if and only if the quantified propositional formula $[M, GNI]_k$ is satisfiable. If $[M, GNI]_k$ is satisfiable, then claim that the system does not satisfy generalized noninterference, and return a counterexample.
3. Inductive Proof by a QBF Solver: Reduce the window induction proof to the quantified propositional satisfiability problem, i.e., the window induction proof of the size of window k succeeds if and only if $[M, GNI]_k^{IN}$ is satisfiable. If $[M, GNI]_k^{IN}$ is satisfiable, then claim that the system satisfies generalized noninterference, else let $k = k + 1$, return to Step 1.

The termination and completeness of our approach depends on the computation of the threshold. The threshold must satisfy that if the system does not satisfy generalized noninterference, then there must be a counterexample of length no more than the threshold. It is clear if the system does not satisfy generalized noninterference, then the minimal threshold is equal to the length of the shortest counterexamples. This implies that finding the smallest threshold is at least as hard as checking whether the system satisfies generalized noninterference. Consequently, we concentrate on computing an over-approximation to the smallest threshold based on graph-theoretic properties of the system. We discuss a bound, and show that the bound can be checked by a SAT solver[18].

A. Related Work

To the best of our knowledge, there are no any tools of verifying generalized noninterference based on states. Therefore, in this subsection we will compare our work with related work in theory. The traditional verification of generalized noninterference is called the unwinding approach[19] which reduces the global requirement to more local conditions that involves only individual transitions. The main problem of the unwinding approach is that it is not complete.That is if the individual transitions satisfy local conditions, we can conclude that the system satisfy generalized noninterference. However, if the local conditions are not satisfied, we can not declare that the system does not satisfy generalized interference.

Compared with the unwinding approach, our's has three advantages: first, our approach is not only sound, but also complete; second our approach can be implemented by a quantified boolean decision procedure which makes us verify large systems; third our approach combines the counterexample search strategy and induction proof technique. The counterexample search strategy makes us find counterexamples quickly.

The paper is organized as follows. In Section 2, we describe the symbolic representation for the nondeterministic security system model. In Section 3, we present our counterexample search based algorithmic verification technique to check generalized noninterference. In Section 4, we show how to reduce the verification to the satisfiability problem of a quantified propositional formula. In Section 5, our experimental results are presented. Some conclusions and ideas for future research are presented in Section 6.

II. Symbolic Representation for Security System Model

A. State-Observed Model

We consider only the state-observed modeling. The system are input-enabled, in the sense that any action can be taken at any time. Most of the literature restricts attention to two users: low level user L and high level user H, and the security policy L $\leq$H. This policy permits information to flow from L to H, but not from H to L. We also make this restriction here. We use a type of state transition graph called a Nondeterministic Security Labeled Kripke Structure(NSLKS) to describe the behavior of a security system.

Definition 1 A NSLKS M is a 8-tuple $(S, s_{in}, \Sigma, \Sigma_L, \Sigma_H, R, AP, O_L)$ where

- S is a finite non-empty set of states.

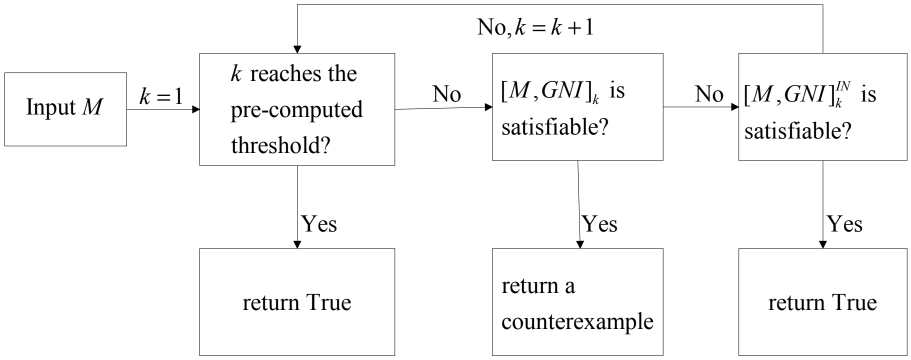

Fig. 1: A Framework for Checking Noninterference Using a SAT Solver

- $s_{in} \in S$ is an initial state.
- Σ is a finite set of actions with $\Sigma = \Sigma_L \cup \Sigma_H$.
- $\Sigma_L \subset \Sigma$ is a finite set of actions of L.
- $\Sigma_H \subset \Sigma$ is a finite set of actions of H.
- $R : S \times \Sigma \rightarrow 2^S \setminus \{\emptyset\}$ is a transition function.
- AP is a finite set of propositions.
- $O_L : S \rightarrow 2^{AP}$ is a labeling function valuations.

In a NSLKS M, a path $\pi = s_0, \sigma_0, s_1, \sigma_1, \ldots, \sigma_k, s_k$ of M is an alternating sequence of states and events subject to the following: for each $k \geq i \geq 0, s_i \in S, \sigma_i \in \Sigma$ and $s_{i+1} \in R(s_i, \sigma_i)$ holds. Given a action sequence $\sigma = \sigma_0 \cdots \sigma_k$, define $s_0 \bullet \sigma = \{s_k | s_0, \sigma_0, s_1, \sigma_1, \ldots, \sigma_k, s_k$ is a path$\}$ to represent a set of sates after executing σ.

B. Symbolic Representation of NSLKS

This subsection describes how a NSLKS can be represented symbolically. To represent this structure we must describe the set S, the set Σ, the transition relation R, and the labeling function O_L. Without loss of generality, we suppose that there are 2^m states for some $m > 0$, 2^n high user actions for some $n > 0$, 2^n low user actions, $AP = \{p_1, \cdots, p_k\}$ for some $k > 0$.

Let $\phi : S \leftrightarrow \{0,1\}^m$ be a bijection function that maps each state of S to a boolean vector of length m. The initial state s_{in} can be represented by a boolean vector $\phi^{-1}(s_{in})$, denoted by $I(s_{in})$. $\psi : \Sigma \leftrightarrow \{0,1\}^{n+1}$ be a bijection function satisfying $\psi : \Sigma_L \leftrightarrow \{0\} \times \{0,1\}^n$, and $\psi : \Sigma_H \leftrightarrow \{1\} \times \{0,1\}^n$. ψ maps each action of Σ to a boolean vector of length $n+1$. Let $\phi^{-1}(s) = (b_1, \ldots, b_m)$. Then, the state s can be characterized by a boolean formula as follows: $\bigwedge_{b_i=1}^{1\leq i\leq k} b_i' \wedge \bigwedge_{b_i=0}^{1\leq i\leq k} \neg b_i'$, where b_i' is an atomic proposition. For simplicity, we use $\phi^{-1}(s)$ instead of the above formula. The transition relation $s' \in R(s, \sigma)$ can be characterized by a boolean formula as follows: $\phi^{-1}(s) \wedge \psi^{-1}(\sigma) \wedge \phi^{-1}(s')$. The labeling function $O_L(s)$ can be represented as follows: $\widehat{O_L(s)} = \phi^{-1}(s) \wedge \bigwedge_{p \in O_L(s)} p \wedge \bigwedge_{p \notin O_L(s)} \neg p$. $O_L(s) \neq O_L(s')$ can be represented as follows: $\widehat{O_L(s)} \wedge \widehat{O_L(s')} \wedge \neg((\bigwedge_{p \in O_L(s)} p \wedge \bigwedge_{p \notin O_L(s)} \neg p) \leftrightarrow (\bigwedge_{p \in O_L(s')} p \wedge \bigwedge_{p \notin O_L(s')} \neg p))$.

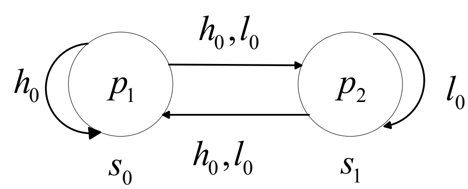

Fig. 2: Two state SLKS

In order to illustrate how to represent a NSLKS symbolically, we consider the two state structure shown in Fig 2. In this case, there are two states. We need one boolean variable v to encode states. We introduce one additional boolean variable v' to encode successor states. There are two actions including one low user action and one high user action. Since we need to distinguish these two kinds of actions, we need two boolean variables u_1, u_2 to encode actions. The aim introducing u_1 is to distinguish whether a action is a low user action or a high user action. Here, $u_1 = 1$ means the action encoded by (u_1, u_2) is a high user action, otherwise the action is a low user action. We use $(1,1)$ to encode h_0, $(0,1)$ to encode l_0. Thus we will represent the transition from state s_0 to state s_1 enabled by inputting l_0 by $\neg v \wedge \neg u_1 \wedge u_2 \wedge v'$. The boolean formula for the entire transition relation is given by $(\neg v \wedge \neg u_1 \wedge u_2 \wedge v') \vee (\neg v \wedge u_1 \wedge u_2 \wedge \neg v') \vee (v \wedge \neg u_1 \wedge u_2 \wedge v') \vee (v \wedge u_1 \wedge u_2 \wedge \neg v') \vee (\neg v \wedge u_1 \wedge u_2 \wedge v') \vee (v \wedge \neg u_1 \wedge u_2 \wedge \neg v')$. The labeling function is represented by $(\neg v \wedge p_1) \vee (v \wedge p_2)$.

III. VERIFYING GENERALIZED NONINTERFERENCE

A. Generalized Noninterference

Historically, one of the first information flow properties was noninterference, defined with respect to deterministic machines. With respect to the simple policy $L \leq H$, the definition of noninterference was formalized by saying that if one removes all the hidden inputs the observations in the view of low users remain unchanged. However, this is not as general as one would lick, since it is only meaningful for deterministic systems. A more general definition is to say that any possible set of observation is consistent with any possible sequence of hidden inputs. This is formalized as follows in the definition of generalized noninterference.

Definition 2 We call a NSLKS M satisfies generalized noninterference, denoted by $M \models GNI$, if and only if for each action sequence σ, each state s of $s_0 \bullet \sigma$, there exists a state $s' \in s_0 \bullet purge_L(\sigma)$ such that $O_L(s) = O_L(s')$, where $purge_L : \Sigma^* \rightarrow \Sigma_L^*$ restricts the sequence to the subsequence of actions of L.

B. Checking Generalized Noninterference by Searching for Counterexamples

Definition 3 (Counterexample for generalized noninterference) Let M be a NSLKS. A finite action sequence $\alpha \in \Sigma^*$ is called a counterexample of generalized noninterference iff there exists a action sequence σ and a state $s \in s_0 \bullet \sigma$ such that for each state $s' \in s_0 \bullet purge_L(\sigma)$, $O_L(s) \neq O_L(s')$.

It is easy to justify that a NSLKS M does not satisfy generalized noninterference iff there is a counterexample. That is we can check generalized noninterference if we consider all possible actions sequences. This leads to a straightforward generalized noninterference checking procedure. To check whether $M \models GNI$, the procedure checks all action sequences with length k for $k = 0, 1, 2, \cdots$. If a counterexample with length k is found, then the procedure proves that $M \not\models GNI$ and produces a counterexample of length k. If there are no counterexamples of length k, we have to increment the value of k indefinitely, and the procedure does not terminate. We now establish a bound on k, and have that for all k within the bound, if there are no counterexamples of length k, we can conclude that $M \models GNI$.

Definition 4(Deterministic System Construction)Let $M = (S, s_{in}, \Sigma, \Sigma_L, \Sigma_H, R, AP, O_L)$ be a security system, define $M^D = (S^D, s_{in}^D, \Sigma, \Sigma_L, \Sigma_H, R^D, AP)$ to be a system as follows:

- $S^D = 2^S$.
- $s_{in}^D = \{s_{in}\}$.
- $R^D : S^D \times \Sigma \rightarrow S^D$ is a transition function given by $S_1^D = R^D(S^D, \sigma)$ if and only if $S_1^D = \bigcup\limits_{s \in S^D} R(s, \sigma)$.

Definition 4 shows from a NSLKS N how to deduce a deterministic system which has same behaviors with N. This deduction can reduce the verification of generalized noninterference to the verification of noninterference over a deterministic system. The following Definition 5 further shows how to reduce the verification of noninterference to a reachability checking problem.

Definition 5.(Double Construction)Let $M^D = (S^D, s_{in}^D, \Sigma, \Sigma_L, \Sigma_H, R^D, AP)$ be a system, define $M^{D^2} = (S^{D^2}, s_{in}^{D^2}, \Sigma, \Sigma_L, \Sigma_H, R^{D^2}, AP)$ to be the system, where

- $S^{D^2} = S^D \times S^D$.
- $s_{in}^{D^2} = (s_{in}^D, s_{in}^D)$.
- $R^{D^2} : S^{D^2} \times \Sigma \rightarrow S^{D^2}$ is a transition function given by $R^{D^2}((s_1^{D^2}, s_2^{D^2}), a) = (R(s_1^D, a), R(s_2^D, a))$ for $a \in \Sigma_L$, and $R^2((s_1^D, s_2^D), a) = (R(s_1, a), s_2^D)$ for $a \in \Sigma_H$.

In Definition 5, we note that in every transition, $a \in \Sigma_H$ is applied only on the left part of each state pair. An easy induction shows that for every sequence of actions $\alpha \in \Sigma^*$, if $s_{in}^{D^2} \bullet \alpha = (s^D, t^D)$ in M^{D^2}, then in M^D we have $s^D = s_{in}^D \bullet \alpha$ and $t^D = s_{in}^D \bullet purge_L(\alpha)$. We therefore obtain the following lemma:

lemma 6 Let M be a security system model, we have $M \models GNI$ iff in M^{D^2}, for all states (s^D, t^D) reachable from $s_{in}^{D^2}$, we have that for each state $s \in S^D$, there exists a state $s' \in t^D$ such that $O_L(s) = O_L(s')$.

Let $|M^{D^2}|$ be the number of states in M^{D^2}. Then $|M^{D^2}| = |M^D|^2 = |2^{|M|}|^2 = 2^{2|M|}$. Since in M^{D^2}, every reachable state is reachable from the initial state within $|M^{D^2}|$ steps. Henceforth, we have the following theorem.

Theorem 7 Let M be a security system model, we have $M \models GNI$ iff there does not exist counterexamples of length no more than $2^{2|M|}$.

Theorem 7 says that when checking whether $M \models GNI$, we only need to check whether there are counterexamples of length no more than $2^{2|M|}$. However, it is unsatisfactory when one considers the necessary number of iterations before it terminates. For a system satisfying GNI, the number of iterations required is $2^{2|M|}$. But this could easily be far too many iterations! In theory we should consider only shortest paths between pairs of states. However this implies that finding the shortest path is at least as hard as checking whether $M \models GNI$. Consequently, we concentrate on computing an over-approximation to the shortest path based on graph-theoretic properties of M^{D^2}.

Definition 8. In a double construction M^{D^2}, we call a finite path $s_0^{D^2}, \sigma_0, \cdots, \sigma_{k-1}, s_k^{D^2}$ of M^{D^2} is a loop-free path if and only if for any $0 \leq i < j \leq k, s_i^{D^2} \neq s_j^{D^2}$.

Definition 9. (Recurrence Diameter)The recurrence diameter of a M^{D^2}, denoted by $rd(M^{D^2})$ is the longest loop-free path (defined by the number of its edges)in M^{D^2} between the initial state and any reachable state.

From the above definition, it is easy to justify that for the double construction M^{D^2} of M^D , any reachable states are reachable from the initial state within $rd(M^{D^2})$ steps. Henceforth, we have the following theorem.

Theorem 10. Let M be a security system model, we have $M \models GNI$ iff there does not exist counterexamples of length no more than $rd(M^{D^2})$.

The solution checking generalized noninterference based on counterexample search is given in pseudo-code below (Algorithm 1).

Algorithm 1. Checking Generalized Noninterference based on Counterexample Search

{
$k = 1$
While $k \leq rd(M^{D^2})$ do
if there exists a counterexample of length k, return False
$k = k + 1$
End While
return True
}

IV. Reducing Verification to QBF

A. Quantified Boolean Formula

A Quantified Boolean Formula (QBF) is a generalized form of a Boolean formula that contains quantifiers. Quantifiers are of two types: universal or existential. For example, $\forall x \exists y \exists z((x \vee y \vee z) \wedge (\neg x \vee \neg y \neg z))$ is a QBF. Formally, the set of quantified boolean formulas(QBF) is defined inductively as follows:

Definition 11 (Quantified Boolean Formula)

1. If f is a propositional formula, it is also a quantified boolean formula.
2. If f is a quantified boolean formula, and x is a Boolean variable, then both $\exists x f$ and $\forall x f$ are quantified boolean formulas;
3. If f and g are quantified Boolean formulas, then $\neg f$, $f \wedge g$, $f \vee g$, and $f \rightarrow g$ are quantified boolean formulas;

As shown in [20], each quantified boolean formula can be written in the following prenex form: $\Phi = Q_1 x_1 \ldots Q_n x_n \phi$ with $Q_i \in \{\exists, \forall\}$ and x_i a propositional variable for $1 \leq i \leq n$, i.e. they consist of a sequence of quantifiers, the $prefix$, followed by a quantifier free propositional formula, the so-called $matrix$ of the formula. The semantics of a QBF Φ can be defined recursively as follows. If the prefix is empty, then the satisfiability of Φ is defined according to the truth tables of propositional logic. If Φ is $\exists x \phi$ (resp. $\forall x \phi$), Φ is satisfiable if and only if Φ_x or (resp. and) Φ_{-x} are satisfiable. Here Φ_x is the QBF obtained from by substituting x with $True$, Φ_{-x} is the QBF obtained from by substituting x with $False$. For example, the formula $\forall x \exists y(x \leftrightarrow y)$ is True. Given a QBF where all of its variables are quantified, the question of determining whether the formula evaluates to true or false is called a QBF satisfiability problem, sometimes called QBF problem.

B. Reducing Counterexample Search to QBF

In the previous section, we have showed generalized noninterference can be checked by searching for counterexamples. We now reduce counterexamples search to quantified propositional satisfiability. This reduction enables us to use efficient quantified propositional decision procedures to perform generalized noninterference checking.

Given a NSLKS structure M, and a bound k, we will construct a quantified boolean formula $[M, GNI]_k$. The variables $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ in $[M, GNI]_k$ denote a alternating finite sequence of states and actions on a path. The formula $[M, GNI]_k$ essentially represents constraints on $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ such that $[M, GNI]_k$ is satisfiable iff there exists a counterexample of length k.

To construct $[M, GNI]_k$, we first define a formula $[M]_k$ that constrains $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ to be a valid path in M. Second, we give the translation of a counterexample of length k to a quantified boolean formula.

Definition 12. (Unfolding the Transition Relation). For a NSLKS M, a positive integer k, $[M]_k = \bigwedge_{i=0}^{k-1} R(s_i, \sigma_i, s_{i+1})$

We recall that generalized noninterference says that the purged H actions are not allowed to lead to any effects observable to L. Henceforth, for the action sequence $\sigma_0, \cdots, \sigma_k$, we need to compute $purge_L(\alpha)$. Suppose that there are i low user actions in $\sigma_0, \cdots, \sigma_k$, define the following $[H]_k^i$ to encode the distributing of these low user actions in $\sigma_0, \cdots, \sigma_k$.

$$[H]_k^i = 0 \le l_1 < k \wedge 0 \le l_i < k \wedge \bigwedge_{j=1}^{i-1}(l_j < l_{j+1}) \wedge \bigwedge_{j=1}^{i}(\sigma_{l_j} \in \Sigma_L) \wedge \bigwedge_{0 \le j \le k-1}^{j \notin \{l_1, \cdots, l_i\}} (\sigma_j \in \Sigma_H)$$

Then we define $[M]_L^i$ to encode the execution of the system after inputting $purge_L(\alpha)$.

$$[M]_L^i = I(s'_{l_1}) \wedge \bigwedge_{j=1}^{i} R(s'_{l_j}, \sigma_{l_j}, s'_{l_{j+1}})$$

Combining all components, the encoding of a counterexample of length k is defined as follows.

Definition 13. (General Translation) For a NSLKS M, a positive integer k,

$$[M, NI]_k = \exists s_0, \sigma_0, ..., \sigma_{k-1}, s_k (I(s_0) \wedge [M]_k \wedge \bigvee_{i=1}^{k-1} \forall s'_{l_1}, ..., s'_{l_{i+1}} ([H]_k^i \wedge [M]_L^i \rightarrow O_L(s_k) \neq O_L(s'_{l_i})))$$

Theorem 14. For a NSLKS M, a positive integer k, $[M, GNI]_k$ is satisfiable if and only if for generalized noninterference there exists a counterexample of length k.

Theorem 14 says that we can check whether there exists a counterexample of length k by a QBF solver. Thus, in Algorithm 1, we can use a quantified propositional decision procedure instead of counterexample checking. We now consider how to use a propositional formula to encode a loop-free path. Directly from the definition of a loop-free path we have the following definition.

Definition 13. $loopfree(s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2}) = \bigwedge_{i=0}^{k-1} R^2(s_i^{D^2}, \sigma_i, s_{i+1}^{D^2}) \wedge \bigwedge_{0 \le i < j \le k} (s_i^{D^2} \neq s_j^{D^2})$.

The solution checking generalized noninterference based on QBF is given in pseudo-code below (Algorithm 2).

Algorithm 2. Checking Generalized Noninterference based QBF

{
$k = 1$
While $I(s_0^{D^2}) \wedge loopfree(s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2})$ is satisfiable do
if $[M, GNI]_k$ is satisfiable return the counterexample $\sigma_0 ... \sigma_{k-1}$
$k = k + 1$
End While
return True
}

C. Combining Induction

In Algorithm 1,2, if $M \models GNI$, then the program must iterate $rd(M^{D^2})$ times. This is not feasible. In this subsection we will discuss how to combine the induction technique and the above counterexample search technique such that the program terminates earlier. In addition, the successful usage of the induction makes us be able to handle larger models since the induction step has to consider only paths of length 1.

We first consider the classical induction. An induction proof consists of proving the following two subgoals:

- For all states $s_0^{D^2}$, if $I(s_0^{D^2})$ holds, then for each state s of $s_0^{D^2}(1)$, there exists a state $s' \in s_0^{D^2}(2)$ such that $O_L(s) = O_L(s')$, where $s_0^{D^2}(i)$ represents the ith element of $s_0^{D^2}$. The subgoal can be encoded as a quantified boolean formula: $\forall s_0^{D^2}(I(s_0^{D^2}) \rightarrow \forall s \in s_0^{D^2}(1) \exists s' \in s_0^{D^2}(2)(O_L(s) = O_L(s')))$.
- For all paths $s_0^{D^2}, \sigma_0, s_1^{D^2}$, if for each state s of $s_0^{D^2}(1)$, there exists a state $s' \in s_0^{D^2}(2)$ such that $O_L(s) = O_L(s')$, then for each state s of $s_1^{D^2}(1)$, there exists a state $s' \in s_1^{D^2}(2)$ such that $O_L(s) = O_L(s')$. The subgoal can be encoded as a quantified formula: $\forall s_0^{D^2} \forall \sigma_0 \forall s_0^{D^2} (R^{D^2}(s_0^{D^2}, \sigma_0) = s_1^{D^2} \wedge \forall s \in s_0^{D^2}(1) \exists s' \in s_0^{D^2}(2)(O_L(s) = O_L(s'))) \rightarrow \forall s_1 \in s_1^{D^2}(1) \exists s'_1 \in s_1^{D^2}(2)(O_L(s_1) = O_L(s'_1))$.

Consider the following tiny example: the system M_1 consists two isolated components: A and B, where A satisfies generalized noninterference, B does not satisfies generalized noninterference. The initial state of A is also the initial state of M. Thus, it is easy to justify M also satisfies generalized noninterference. In B the length of minimal counterexample is 1. However, in this case the classical induction technique can not be used to prove $M \models GNI$ successfully. The reason is that the classical induction technique attempt to prove that $B \models GNI$ while this is impossible. Therefore the application of the classical induction has much limitation.

Windowed induction is a modified induction technique which has been used to prove a hardware system design[17]. The advantage of windowed induction over classical induction is that it provides the user with a way of strengthening the induction hypothesis: lengthening the window k. Mathematically, for noninterference windowed induction with window size $k \geq 0$ consists of the following two steps:

- Prove that for all paths $s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2}$, if $I(s_0^{D^2})$ holds, then for each state s_i of $s_i^{D^2}(1)$, there exists a state $s_i' \in s_i^{D^2}(2)$ such that $O_L(s_i) = O_L(s_i')$ for all $0 \leq i \leq k$.
- Prove that for all paths $s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2}, \sigma_k, s_{k+1}^{D^2}$, if for each state s_i of $s_i^{D^2}(1)$, there exists a state $s_i' \in s_i^{D^2}(2)$ such that $O_L(s_i) = O_L(s_i')$ for all $0 \leq i \leq k$, then for each state s_{k+1} of $s_{k+1}^{D^2}(1)$, there exists a state $s_{k+1}' \in s_{k+1}^{D^2}(2)$ such that $O_L(s_{k+1}) = O_L(s_{k+1}')$.

The first step can be completed by checking whether $[M, GNI]_k$ is satisfiable. The second step can be completed by checking whether a corresponding quantified formula is satisfiable. We recall the definition of $[M]_k$. Then we have that $[M^{D^2}]_k = \bigwedge_{i=0}^{k-1} R^2(s_i^{D^2}, \sigma_i, s_{i+1}^{D^2})$. Let $[M, GNI]_k^{In} = \forall s_0^{D^2} \forall \sigma_0 ... \forall \sigma_{k-1} \forall s_k^{D^2} \forall \sigma_k \forall s_{k+1}^{D^2} (([M^2]_{k+1} \wedge (\bigwedge_{i=0}^{k} (\forall s_i \in s_i^{D^2}(1) \exists s_i' \in s_i^{D^2}(2)(O_L(s_i) = O_L(s_i')) \rightarrow \forall s_{k+1} \in s_{k+1}^{D^2}(1) \exists s_{k+1}' \in s_{k+1}^{D^2}(2)(O_L(s_{k+1}^1) = O_L(s_{k+1}^{'}))))$. It is easy to justify that $[M, GNI]_k^{In}$ is satisfiable if and only if the conclusion we must prove in the second step of windowed induction is correct. The solution checking generalized noninterference based on induction is given in pseudo-code below (Algorithm 3).

Algorithm 3. Checking Generalized Noninterference based Counterexample Search and Induction

{
$k = 1$
While $I(s_0^{D^2}) \wedge loopfree(s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2})$ is satisfiable do
if $[M, GNI]_k$ is satisfiable return the counterexample $\sigma_0 ... \sigma_{k-1}$
if $[M, GNI]_k^{In}$ is satisfiable return True
$k = k + 1$
End While
return True
}

D. Example

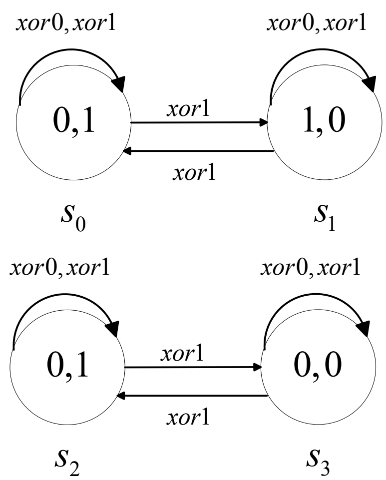

Fig. 3: An example: M

In this subsection we take an example from [21] to show our translation procedure. The example is deterministic system. So, we add some local transition relations to the system such that the system is nondeterministic. Consider a machine M with two bits of state information, H and L (for "high" and "low," respectively). The machine has two commands, $xor0$ and $xor1$. There are two users: Holly (who can read and modify high and low information) and Lucy (who can read only low information). The system keeps two bits of state (H, L). For this example, the operation affects both state bits regardless of whether Holly or Lucy executes the instruction. The state transition relation of M is given in Fig. 3. s_0 is the initial state of M.

In M, there are four states. We need two boolean variables v_1, v_2 to encode states, and v_2 is observable for the low user. We introduce two additional boolean variable $v_1^{'}, v_2^{'}$ to encode successor states. There are two actions including one low user action $xor0$ and one high user action $xor1$. We need two boolean variables u_1, u_2 to encode actions. The aim introducing u_1 is to illustrate whether a action is a low user action or a high user action. Here, $u_1 = 1$ means the action encoded by u_1, u_2 is a high user action, otherwise the action is a low user action. We use $(0,1)$ to encode $xor0$, use $(1,1)$ to encode $xor1$. Thus the boolean formula R for the entire transition relation is given by $R(v_1,v_2,u_1,u_2,v_1^{'},v_2^{'}) = (\neg v_1 \wedge v_2 \wedge \neg u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge v_2^{'}) \vee (\neg v_1 \wedge v_2 \wedge u_1 \wedge u_2 \wedge v_1^{'} \wedge \neg v_2^{'}) \vee (v_1 \wedge \neg v_2 \wedge \neg u_1 \wedge u_2 \wedge v_1^{'} \wedge \neg v_2^{'}) \vee (v_1 \wedge \neg v_2 \wedge u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge v_2^{'}) \vee (v_1 \wedge v_2 \wedge u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge \neg v_2^{'}) \vee (v_1 \wedge v_2 \wedge \neg u_1 \wedge u_2 \wedge v_1^{'} \wedge v_2^{'}) \vee (\neg v_1 \wedge \neg v_2 \wedge \neg u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge \neg v_2^{'}) \vee (\neg v_1 \wedge \neg v_2 \wedge u_1 \wedge u_2 \wedge v_1^{'} \wedge v_2^{'}) \vee (\neg v_1 \wedge v_2 \wedge u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge \neg v_2^{'}) \vee (v_1 \wedge \neg v_2 \wedge u_1 \wedge u_2 \wedge v_1^{'} \wedge \neg v_2^{'}) \vee (v_1 \wedge v_2 \wedge u_1 \wedge u_2 \wedge v_1^{'} \wedge v_2^{'}) \vee (\neg v_1 \wedge \neg v_2 \wedge u_1 \wedge u_2 \wedge \neg v_1^{'} \wedge \neg v_2^{'})$. Since only v_2 is observable for Lucy, we only need a boolean variable p to encode the observation of Lucy. For the state s, if $v_2 = 1$, let $L(s) = \{p\}$, else let $L(s) = \emptyset$. Thus the labeling function is represented by $\widehat{O_L(s)} = (\neg v_1 \wedge v_2 \wedge p) \vee (v_1 \wedge \neg v_2 \wedge \neg p) \vee (v_1 \wedge v_2 \wedge p) \vee (\neg v_1 \wedge \neg v_2 \wedge \neg p)$. The initial state can be encoded as $(\neg v_1 \wedge v_2)$.

We consider whether there are counterexamples of length 3. Let $k = 3$. The variables $s_0, \sigma_0, s_1, \sigma_1, s_2, \sigma_2, s_3$ denote a alternating finite sequence of states and actions on a path. For simplicity, in the boolean variables encoding states, for $0 \leq i \leq k$ we use $s_i[1]$ to represent the first boolean variable, $s_i[2]$ to represent the second boolean variable. For $0 \leq i \leq k-1$ we use $\sigma_i[1]$ to represent the first boolean variable, $\sigma_i[2]$ to represent the second boolean variable. Thus $[M]_2 = (\neg s_0[1] \wedge s_0[2]) \wedge R(s_0[1], s_0[2], \sigma_0[1], \sigma_0[2], s_1[1], s_1[2]) \wedge R(s_1[1], s_1[2], \sigma_1[1], \sigma_1[2], s_2[1], s_2[2]) \wedge R(s_2[1], s_2[2], \sigma_2[1], \sigma_2[2], s_3[1], s_3[2])$.

For the action sequence $\sigma = \sigma_0\sigma_1\sigma_2$, $purge_L(\sigma) \in \{\varepsilon, \sigma_0, \sigma_1, \sigma_2, \sigma_0\sigma_1, \sigma_0\sigma_2, \sigma_1\sigma_2, \sigma_0\sigma_1\sigma_2\}$. In the following for each value of $purge_L(\sigma)$, we show how to encode counterexamples.

- For the case $purge_L(\sigma) = \varepsilon$, we need to represent $\sigma_0, \sigma_1, \sigma_2$ are high user actions, and $O_L(s_3) \neq O_L(s_0)$. Thus we have that $\eta_0 = \sigma_0[1] \wedge \sigma_1[1] \wedge \sigma_2[1] \wedge O_L(s_3) \neq O_L(s_0)$.
- For the case $purge_L(\sigma) = \sigma_0$, we need to represent that σ_1, σ_2 are high user actions, and for each valid path $s_0^{'}, \sigma_0, s_1^{'}$, $O_L(s_3) \neq O_L(s_1^{'})$. Thus we have that $\eta_1 = \forall s_0^{'} \forall s_1^{'}((\neg \sigma_0[1] \wedge \sigma_1[1] \wedge \sigma_2[1] \wedge I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_0[1], \sigma_0[2], s_1^{'}[1], s_1^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_1^{'}))$.
- For the case $purge_L(\sigma) = \sigma_1$, we need to represent that σ_0, σ_2 are high user actions, and for each valid path $s_0^{'}, \sigma_1, s_1^{'}$, $O_L(s_3) \neq O_L(s_1^{'})$. Thus we have that $\eta_2 = \sigma_0[1] \wedge \neg\sigma_1[1] \wedge \sigma_2[1] \wedge \forall s_0^{'} \forall s_1^{'}(I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_1[1], \sigma_1[2], s_1^{'}[1], s_1^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_1^{'}))$.
- For the case $purge_L(\sigma) = \sigma_2$, we need to represent that σ_0, σ_1 are high user actions, and for each valid path $s_0^{'}, \sigma_2, s_1^{'}$, $O_L(s_3) \neq O_L(s_1^{'})$. Thus we have that $\eta_3 = \sigma_0[1] \wedge \sigma_1[1] \wedge \neg\sigma_2[1] \wedge \forall s_0^{'} \forall s_1^{'}(I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_2[1], \sigma_2[2], s_1^{'}[1], s_1^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_1^{'}))$.
- For the case $purge_L(\sigma) = \sigma_0\sigma_1$, we need to represent that σ_2 are high user actions, and for each valid path $s_0^{'}, \sigma_0, s_1^{'}, \sigma_1, s_2^{'}$, $O_L(s_3) \neq O_L(s_2^{'})$. Thus we have that $\eta_4 = \neg\sigma_0[1] \wedge \neg\sigma_1[1] \wedge \sigma_2[1] \wedge \forall s_0^{'} \forall s_1^{'} \forall s_2^{'}(I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_0[1], \sigma_0[2], s_1^{'}[1], s_1^{'}[2]) \wedge R(s_1^{'}[1], s_1^{'}[2], \sigma_1[1], \sigma_1[2], s_2^{'}[1], s_2^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_2^{'}))$.
- For the case $purge_L(\sigma) = \sigma_0\sigma_2$, we need to represent that σ_1 are high user actions, and for each valid path $s_0^{'}, \sigma_0, s_1^{'}, \sigma_2, s_2^{'}$, $O_L(s_3) \neq O_L(s_2^{'})$. Thus we have that $\eta_5 = \neg\sigma_0[1] \wedge \sigma_1[1] \wedge \neg\sigma_2[1] \wedge \forall s_0^{'} \forall s_1^{'} \forall s_2^{'}(I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_0[1], \sigma_0[2], s_1^{'}[1], s_1^{'}[2]) \wedge R(s_1^{'}[1], s_1^{'}[2], \sigma_2[1], \sigma_2[2], s_2^{'}[1], s_2^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_2^{'}))$.
- For the case $purge_L(\sigma) = \sigma_1\sigma_2$, we need to represent that σ_0 are high user actions, and for each valid path $s_0^{'}, \sigma_1, s_1^{'}, \sigma_2, s_2^{'}$, $O_L(s_3) \neq O_L(s_2^{'})$. Thus we have that $\eta_6 = \neg\sigma_0[1] \wedge \sigma_1[1] \wedge \sigma_2[1] \wedge \forall s_0^{'} \forall s_1^{'} \forall s_2^{'}(I(s_0^{'}) \wedge R(s_0^{'}[1], s_0^{'}[2], \sigma_1[1], \sigma_1[2], s_1^{'}[1], s_1^{'}[2]) \wedge R(s_1^{'}[1], s_1^{'}[2], \sigma_2[1], \sigma_2[2], s_2^{'}[1], s_2^{'}[2]) \rightarrow O_L(s_3) \neq O_L(s_2^{'}))$.
- For the case $purge_L(\sigma) = \sigma_0\sigma_1\sigma_2$,since there are no high user actions in σ, we do not need to consider this case.

Therefore, we have that $[M, GNI]_3 = \exists s_0 \exists \sigma_0 \exists s_1 \exists \sigma_1 \exists s_2 \exists \sigma_2 \exists s_3 (I(s_0) \wedge [M]_k \wedge (\eta_0 \vee \eta_1 \vee \eta_2 \vee \eta_3 \vee \eta_4 \vee \eta_5 \vee \eta_6))$. It is easy to claim that $[M, GNI]_3$ is satisfiable. The path $s_0, xor0, s_0, xor0, s_0, xor1, s_1$ is an assignment making $[M, GNI]_k$ true.

Now we show how to compute the overapproximation of the minimal counterexample length, i.e. the recurrence diameter of M^{D^2}. We first present how to compute the deterministic system construction M^D of M. First, let $s_0^D = \{s_0\}$. Then we compute the successor states of s_0^D:since $R(s_0, xor0) = \{s_0\}$, $R^D(s_0^D, xor0) = s_0^D$; since $R(s_0, xor1) = \{s_0, s_1\}$, $R^D(s_0^D, xor1) =$

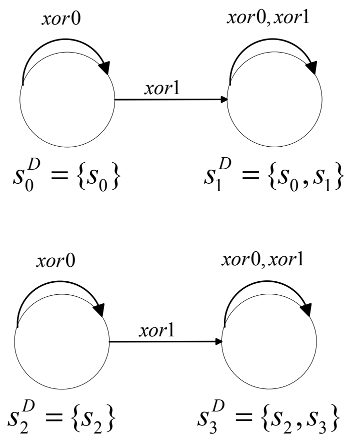

Fig. 4: The deterministic system construction of M: M^D

s_1^D,where $s_1^D = \{s_0, s_1\}$. In the same way, we can other states and local transitions. The final result is shown in Fig. 4.

In the following, we will present how to compute the double construction of M^D. According the definition of double construction, let $S_0^{D^2} = (s_0^D, s_0^D)$. Then we compute the successor states of $S_0^{D^2}$: since $xor0 \in \Sigma_L$, $R^D(s_0^D, xor0) = s_0^D$, we have that $R^{D^2}(S_0^{D^2}, xor0) = S_0^{D^2}$; since $xor1 \in \Sigma_H$, $R^D(s_0^D, xor1) = s_1^D$, we have that $R^{D^2}(S_0^{D^2}, xor1) = (s_1^D, s_0^D)$. In the same way, we can other states and local transitions. The final result is shown in Fig. 5. We recall that the longest loop-free path is the longest path satisfying there are no two same states in the path. Henceforth, $rd(M^{D^2}) = 1$. That is in M, the length of minimal counterexample is no more than 1. It is easy to justify that the action sequence $\sigma = xor1$ is a minimal counterexample.

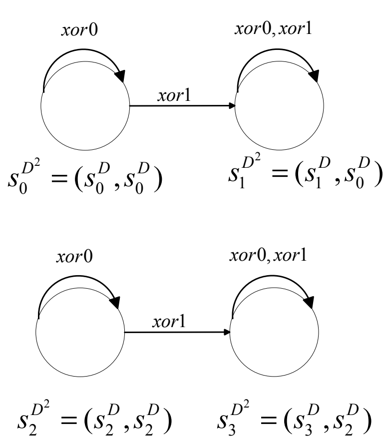

Fig. 5: The double construction of M^D: M^{D^2}

TABLE I: Experiments with the length of the minimal counterexample 4

Problem	States	Actions	A_1	A_2
ELEV(1,4)	158	99	1.97	2.75
ELEV(2,4)	1062	299	52.68	32.43
ELEV(3,4)	7121	783	1771.24	364.83
ELEV(4,4)	43440	1939	N/A	N/A
MMGT(2,4)	817	114	14.01	12.73
MMGT(3,4)	7703	172	71.52	60.97
MMGT(4,4)	66309	232	1277.15	663.78
RING(3,4)	87	33	0.42	0.67
RING(5,4)	1290	55	16.30	14.38
RING(7,4)	17000	77	401.02	219.74
RING(9,4)	211528	99	N/A	783.18

TABLE II: Experiments with the length of the minimal counterexample 6

Problem	States	Actions	A_1	A_2
ELEV(1,6)	158	99	11.65	9.42
ELEV(2,6)	1062	299	346.08	176.39
ELEV(3,6)	7121	783	N/A	2137.25
ELEV(4,6)	43440	1939	N/A	N/A
MMGT(2,6)	817	114	22.98	16.80
MMGT(3,6)	7703	172	213.44	167.62
MMGT(4,6)	66309	232	2050.32	1245.78
RING(3,6)	87	33	2.13	3.01
RING(5,6)	1290	55	59.74	46.20
RING(7,6)	17000	77	1682.94	829.45
RING(9,6)	211528	99	N/A	1752.71

V. Experimental results

The solution we proposed mainly consists of two components: the counterexample search component $[M, GNI]_k$, and the induction proof component $[M, GNI]_k^{IN}$. In this section we will evaluate these two components. We conducted experimental evaluation using a Linux workstation with a 3.06GHZ Pentium processor and 2048MByte memory. We choosed Quantor [12] as the prover. All benchmarks used in the experiment were taken from [22]. They have been converted from communicating state machines to Nomdeterministic Security Labeled Kripke Structures. In the conversion, for each action we assigned a security class randomly, and rename some actions such that systems are nondeterministic.

We first evaluate the counterexample search component $[M, GNI]_k$. For the fairness of evaluation and simplicity, we use $k \leq rd(M^{D^2})$instead of the termination criteria $I(s_0^{D^2}) \wedge loopfree(s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2})$ of Algorithm 2. We collected three kinds of assignment satisfying that the length of the minimal

TABLE III: Experiments with the length of the minimal counterexample 8

Problem	States	Actions	A_1	A_2
ELEV(1,8)	158	99	25.28	18.57
ELEV(2,8)	1062	299	890.15	407.46
ELEV(3,8)	7121	783	N/A	N/A
ELEV(4,8)	43440	1939	N/A	N/A
MMGT(2,8)	817	114	39.88	24.68
MMGT(3,8)	7703	172	418.95	313.01
MMGT(4,8)	66309	232	N/A	1912.76
RING(3,8)	87	33	6.35	7.22
RING(5,8)	1290	55	189.38	113.19
RING(7,8)	17000	77	N/A	N/A
RING(9,8)	211528	99	N/A	N/A

TABLE IV: Experiments with the lengths of the induction depth 3

Problem	States	Actions	A_3
ELEV(1)$\oplus M_1$	162	101	43.71
ELEV(2)$\oplus M_1$	1066	301	569.32
ELEV(3)$\oplus M_1$	7125	785	N/A
ELEV(4)$\oplus M_1$	43444	1941	N/A
MMGT(2)$\oplus M_1$	821	116	27.64
MMGT(3)$\oplus M_1$	7707	174	301.79
MMGT(4)$\oplus M_1$	66313	234	N/A
RING(3)$\oplus M_1$	91	35	19.85
RING(5)$\oplus M_1$	1294	57	537.77
RING(7)$\oplus M_1$	17004	79	N/A
RING(9)$\oplus M_1$	211532	101	N/A

counterexample are 4, 6 and 8 respectively. The experimental results can be found in Table I,II,III. The columns are

- Problem: The problem name with the size of the instance and the length of minimal counterexample in parenthesis.
- States: Number of reachable states in the SLKS.
- Actions: Number of actions in the SLKS.
- A_i: The time required by Algorithm i to find a counterexample for the value of k.

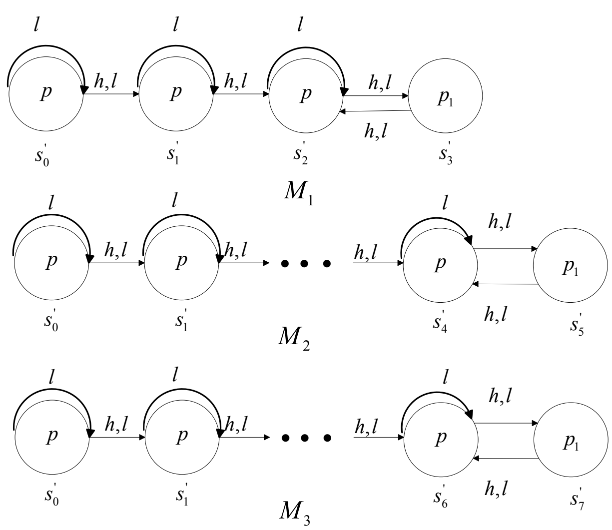

Fig. 6: Three small NSLKSs not satisfying generalized noninterference

We now evaluate the induction proof component $[M, GNI]_k^{IN}$. For evaluation purposes, we delete the counterexample search procedure in Algorithm 3. We now consider how to construct some benchmarks. We first construct three tiny NSLKS M_1, M_2, M_3 shown in Fig. 6 such that the lengths of minimal counterexamples are 3,5,and 7 respectively. Then based on M_1, M_2, M_3, we consider constructing benchmarks such that these benchmarks satisfies generalized noninterference and the induction proof depths are 3,5 and 7 respectively. The construction procedure is designed as follows:

1. For benchmarks used in Table 1 we assigned a security class randomly for each action again such that these benchmarks satisfies generalized noninterference. Let B represent the set of these benchmarks.
2. Define a composition operation. Let $N_1 = (S^1, s_{in}^1, \Sigma^1, \Sigma_L^1, \Sigma_H^1, R^1, \ AP^1, O_L^1)$, $N_2 = (S^2, s_{in}^2, \Sigma^2, \Sigma_L^2, \Sigma_H^2, R^2, AP^2, O_L^2)$. If $S^1 \cap S^2 = \emptyset$, $\Sigma^1 \cap \Sigma^2 = \emptyset$, we define $N_1 \oplus N_2$ as follows: $(S^1 \cup S^2, s_{in}^1, \Sigma^1 \cup \Sigma^2, \Sigma_L^1 \cup \Sigma_L^2, \Sigma_H^1 \cup \Sigma_H^2, R^1 \cup R^2, AP^1 \cup AP^2, O_L)$, where for $s \in S^1, O_L(s) = O_L^1(s)$, for $s \in S^2, O_L(s) = O_L^2(s)$. Note that the initial state of $N_1 \oplus N_2$ is the initial state of N_1.
3. Define the set of benchmarks: $\{M = b_1 \oplus b_2 | b_1 \in B, b_2 \in \{M_1, M_2, M_3\}\}$.

From the definition of $\oplus$ it is easy to justify that if b_1 satisfies noninterference, then $b_1 \oplus b_2$ satisfies generalized noninterference also. That is each element of M satisfies generalized noninterference. The experimental results can be found in Table IV,V,VI. Note that in Algorithm 3, we have deleted the counterexample search procedure.

The set of experiments we used is too small to say anything conclusive about the performance of our methods. There are, however, still some interesting observations to be made as follows.

- Whether the explicit algorithm i.e. Algorithm 1 or the QBF-based symbolic algorithm i.e. Algorithm 2, then can handle with systems with small minimal counterexamples quickly. And for systems with small induction depth, the induction proof can also be implemented

TABLE V: Experiments with the lengths of the induction depth 5

Problem	States	Actions	A_3
ELEV(1)$\oplus M_2$	164	101	215.22
ELEV(2)$\oplus M_2$	1068	301	N/A
ELEV(3)$\oplus M_2$	7127	785	N/A
ELEV(4)$\oplus M_2$	43446	1941	N/A
MMGT(2)$\oplus M_2$	822	117	196.15
MMGT(3)$\oplus M_2$	7708	175	N/A
MMGT(4)$\oplus M_2$	66315	234	N/A
RING(3)$\oplus M_2$	93	35	94.41
RING(5)$\oplus M_2$	1296	57	2249.38
RING(7)$\oplus M_2$	17006	79	N/A
RING(9)$\oplus M_2$	211534	101	N/A

TABLE VI: Experiments with the lengths of the induction depth 7

Problem	States	Actions	A_3
ELEV(1)$\oplus M_3$	166	101	1155.73
ELEV(2)$\oplus M_3$	1070	301	N/A
ELEV(3)$\oplus M_3$	7129	785	N/A
ELEV(4)$\oplus M_3$	43448	1941	N/A
MMGT(2)$\oplus M_3$	825	116	1763.39
MMGT(3)$\oplus M_3$	7711	174	N/A
MMGT(4)$\oplus M_3$	66317	234	N/A
RING(3)$\oplus M_3$	95	35	778.88
RING(5)$\oplus M_3$	1298	57	N/A
RING(7)$\oplus M_3$	17008	79	N/A
RING(9)$\oplus M_3$	211536	101	N/A

quickly.

- For very small systems, the explicit approach outperforms the QBF-based verification approach since the latter needs time to encode counterexamples. While for large systems, QBF-based verification approach outperforms the explicit approach very much. That is there exists lots of systems which can be verified by Algorithm 2, but can not be verified by Algorithm 1 in limited time.
- In our approach the bound k is increased until a counterexample is found, or the induction proof holds, or some pre-computed bound is reached. Unfortunately, the pre-computed bounds may be too large to effectively explore the associated bounded search space, such as in Table I. ELEV(4) with $k = 4$. Therefore, for a large system with a large bound, our approach is complete in theory. However, in practice limited by space and time our approach is not feasible.

VI. Conclusions and Future Work

The main contribution of this paper is to present an algorithmic approach to checking generalized noninterference, and our approach is sound and complete. The main advantage of our approach includes two aspects. First, our approach combines the counterexamples search strategy and the window induction proof technique. The counterexamples search strategy makes us find the counterexample of minimal length rapidly. The window induction proof technique strengthens the induction hypothesis. Second, our approach can be implemented using a QBF-solver. Other contributions includes: in order to make the search procedure terminate as soon as possible, we discuss a over approximation on the length of minimal counterexamples.

There are many interesting avenues for future research. Our current work concentrates on three directions. First we are extending our approach to other information flow security properties. Second, we are introducing the abstraction technique such that we can abstract the finite state behaviors from infinite state systems while preserving noninterference. Third, since our technique translates the search of counterexamples of increasing length into a sequence of quantified propositional satisfiability checks, we will exploit the similarity of these QBF instances by conflict-driven learning during conflict analysis from one instance to the next.

Acknowledgment

The authors' work is supported by the National Natural Science Foundation of China No. 60773049, the People with Ability Foundation of Jiangsu University No. 07JDG014,the Fundamental Research Project of the Natural Science in Colleges of Jiangsu Province No. 08KJD520015

References

[1] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. IEEE Transactions on Software Engineering, 27(1997),pp.550-571.

[2] Jonathan Millen, 20 Years of Covert Channel Modeling and Analysis, Proceedings of the 1999 IEEE Symposium on Security and Privacy, Page(s):113-114.

[3] S. H. Qing. Covet channel analysis in secure operating systems with high security levels. Journal of Software, 15(12)(2004),pp.1837-1849.

[4] Goguen and J. Meseguer, Security Policies and Security Models. Proceedings of the IEEE Symposium on Security and Privacy, Oackland, California, 1982, Pages: 11-21.

[5] D. McCullough, Specifications for multilevel security and a hookup property, Proceedings of the 18th IEEE Computer Society Symposium on Research in Security and Privacy, 1987, pp. 161-166.
[6] McLean, J., A general theory of composition for trace sets closed under selective interleaving functions, Proceedings of the IEEE Symposium on Research in Security and Privacy (1994), pp. 79-93.
[7] A. Zakinthinos and E. S. Lee, A general theory of security properties, Proceedings of the 18th IEEE Computer Society Symposium on Research in Security and Privacy, 1997.
[8] Ron van der Meyden, Chenyi Zhang, Algorithmic Verification of Noninterference Properties, Electronic Notes in Theoretical Computer Science, Volume 168, 8 February 2007, Pages 61-75.
[9] Enrico Giunchiglia, Massimo Narizzano and Armando Tacchella, QBF Reasoning on Real-World Instances, Lecture Notes in Computer Science, 2005,3542:105-121.
[10] Luca Pulina, Armando Tacchella, QuBIS : An (In)complete Solver for Quantified Boolean Formulas , Lecture Notes in Computer Science, Volume 5317, 34-43,2008.
[11] Rowley, A.G.D., Gent, I.P., Hoos, H.H., Smyth, K.: Using Stochastic Local Search to Solve Quantified Boolean Formulae. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 348-362. Springer, Heidelberg (2003)
[12] A. Biere. Resolve and Expand. In Seventh Intl. Conference on Theory and Applications of Satisfiability Testing (SAT'04), volume 3542 of LNCS, 59-70,2005.
[13] Pan, G., Vardi, M.Y.: Symbolic Decision Procedures for QBF. In:Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 453-67. Springer, Heidelberg (2004).
[14] Nachum Dershowitz , Ziyad Hanna and Jacob Katz, Bounded Model Checking with QBF, Lecture Notes in Computer Science, Volume 3569, 408-414,2005.
[15] Toni Jussila, Armin Biere, Compressing BMC Encodings with QBF, Electronic Notes in Theoretical Computer Science (ENTCS), Volume 174 , Issue 3, Pages 45-56,2007.
[16] Conghua Zhou, Zhenyu Chen, and Zhihong Tao, QBF-Based Symbolic Model Checking for Knowledge and Time, Lecture Notes in Computer Science, Volume 4484, 386-397,2007.
[17] Roy Armoni, Limor Fix, Ranan Fraer, Scott Huddleston, Nir Piterman, Moshe Y. Vardi,SAT-based Induction for Temporal Safety Properties, Electronic Notes in Theoretical Computer Science Volume 119, Issue 2, 14 March 2005, Pages 3-16.
[18] Hantao Zhang. SATO: An Efficient Propositional Prover. In William McCune, editor, Proceedings of the 14th International Conference on Automated Deduction (CADE), volume 1249 of Lecture Notes in Computer Science, pages 272-275. Springer, July 1997.
[19] Joseph A. Goguen and Sose Meseguer. Unwinding and Inference Control, Proceedings of the Symposium on Security and Privacy. pages 75-86. IEEE Computer Society, May 1984.
[20] Lintao Zhang, Searching for Truth:techniques for satisfiability of boolean formulas, Ph. D. Thesis, Princeton University.
[21] Matt Bishop, Computer Security: Art and Science. Addison Wesley, 2002.
[22] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. IEEE Transactions on Software Engineering, Volume 22 , Issue 3,Pages: 161-180, 1996.