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 prob-
lem 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 $[M, GNI]^{[N]}_k$ is satisfiable. If $[M, GNI]^{[N]}_k$ 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 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 $(S, s_i, \Sigma, \Sigma_L, \Sigma_H, R, AP, O_L)$ where

- $S$ is a finite non-empty set of states.
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 \( \Sigma \), 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, \( AP = \{p_1, \ldots, p_k\} \) for some \( k > 0 \).

Let \( \phi : S \mapsto \{0, 1\}^m \) be a bijection function that maps each state of \( S \) to a boolean vector of length \( m \). The initial state \( s_{in} \in S \) can be represented by a boolean vector \( \phi^{-1}(s_{in}) \), denoted by \( I(s_{in}) \). \( \psi : \Sigma \mapsto \{0, 1\}^{n+1} \) be a bijection function satisfying \( \psi : \Sigma_L \mapsto \{0\} \times \{0, 1\}^n \), and \( \psi : \Sigma_H \mapsto \{1\} \times \{0, 1\}^n \). \( \psi \) maps each action of \( \Sigma \) 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: \( \bigwedge_{1 \leq i \leq k} b_i \land \bigwedge_{1 \leq i \leq k} \neg b_i \), 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: \( O_L(s) = \phi^{-1}(s) \wedge \bigwedge_{p \in O_L(s)} p \wedge \bigwedge_{p \not\in O_L(s)} \neg p \). \( O_L(s) \neq O_L(s') \) can be represented as follows: \( O_L(s) \land O_L(s') \land \neg((\bigwedge_{p \in O_L(s')} p \land \bigwedge_{p \not\in O_L(s')} \neg p)) \).
given by \((-w \land -u_1 \land u_2 \land v') \lor (-w \land u_1 \land u_2 \land -v') \lor (v' \land -u_1 \land u_2 \land v') \lor (v' \land u_1 \land u_2 \land -v') \lor (-w \land u_1 \land u_2 \land v') \lor (v' \land -u_1 \land u_2 \land -v')\). The labeling function is represented by \((-w \land p_1) \lor (v \land p_2)\).

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 like, since it is only meaningful for deterministic systems. A more general definition is to say that 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 \(\alpha\), each state \(s \in s_0 \bullet \alpha\), there exists a state \(s' \in s_0 \bullet \text{purge}_L(\alpha)\) such that \(O_L(s) = O_L(s')\), where \(\text{purge}_L : \Sigma^* \rightarrow \Sigma_L^*\) 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 \(\sigma\) and a state \(s \in s_0 \bullet \alpha\) such that for each state \(s' \in s_0 \bullet \text{purge}_L(\sigma)\), \(O_L(s) \neq O_L(s')\).

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 \(k = 0, 1, 2, \ldots\). If a counterexample with length \(k\) is found, then the procedure proves that \(M \models GNI\) 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 \(M = (S, s_0, \Sigma, \Sigma_L, \Sigma_H, R, AP, O_L)\) be a security system, define \(M^D = (S^D, s^D_0, \Sigma, \Sigma_L, \Sigma_H, R^D, AP)\) to be a system as follows:

- \(S^D = 2^S\).
- \(s^D_0 = \{s_0\}\).
- \(R^D : S^D \times \Sigma \rightarrow S^D\) is a transition function given by \(S^D = R^D(S^D, \sigma)\) if and only if \(S^D = \bigcup_{s \in S^D} R(s, \sigma)\).

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 \(M^D = (S^D, s^D_0, \Sigma, \Sigma_L, \Sigma_H, R^D, AP)\) be a system, define \(M^{D^2} = (S^{D^2}, s^{D^2}_0, \Sigma, \Sigma_L, \Sigma_H, R^{D^2}, AP)\) to be the system, where

- \(S^{D^2} = S^D \times S^D\).
- \(s^{D^2}_0 = (s^D_0, s^D_0)\).
- \(R^{D^2} : S^{D^2} \times \Sigma \rightarrow S^{D^2}\) is a transition function given by \(R^{D^2}((s^D_1, s^D_2), a) = (R(s^D_1, a), R(s^D_2, a))\) for \(a \in \Sigma_L\), and \(R^{D^2}((s^D_1, s^D_2), a) = (R(s^D_1, a), s^{D^2}_2)\) 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 \(s^{D^2}_n \bullet \alpha = (s^D, t^D)\) in \(M^{D^2}\), then in \(M^D\) we have \(s^D = s^{D^2}_n \bullet \alpha\) and \(t^D = s^{D^2}_n \bullet \text{purge}_L(\alpha)\). 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 \(s^{D^2}_n\), 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 \(|M^{D^2}| = |M^D|^2 = 2|M|^2 = 2^{|M|}^2\). 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 $M^{DP}$.

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

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

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

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

```plaintext
{k = 1
    While k \leq rd(M^{DP}) 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, $\forall x \exists y \exists z ((x \lor y \lor z) \land (\neg x \lor \neg y \lor \neg z))$ 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 propositional 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 $\neg f, f \land g, f \lor g$, 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 $\pi$, followed by a quantifier free propositional formula, the so-called matrix of the formula. The semantics of a QBF $\Phi$ can be defined recursively as follows. If the prefix is empty, then the satisfiability of $\Phi$ is defined according to the truth tables of propositional logic. If $\Phi$ is $\exists x \phi$ (resp. $\forall x \phi$), $\Phi$ is satisfiable if and only if $\Phi_x$ or (resp. and) $\Phi_{\neg x}$ are satisfiable. Here $\Phi_x$ is the QBF obtained from by substituting $x$ with $True$, $\Phi_{\neg x}$ 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 $s_0, \sigma_0, \ldots, \sigma_{k-1}, s_k$ in $[M, GNI]_k$ denote an alternating finite sequence of states and actions on a path. The formula $[M, GNI]_k$ essentially represents constraints on $s_0, \sigma_0, \ldots, \sigma_{k-1}, s_k$ such that $[M, GNI]_k$ is satisfiable iff there exists a counterexample of length $k$. 

ISSN: 1109-2750

980

Issue 6, Volume 8, June 2009
To construct \([M, GNI]_k\), we first define a formula \([M]_k\) that constrains \(s_0, s_1, \ldots, s_{k-1}, s_k\) 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 \(s_0, \ldots, s_k\), we need to compute \(\text{purge}_L(\alpha)\). Suppose that there are \(i\) low user actions in \(s_0, \ldots, s_k\), define the following \([H]_k\) to encode the distributing of these low user actions in \(s_0, \ldots, s_k\).

\[ [H]_k^i = 0 \leq l_1 < k \land 0 \leq l_i < k \land \bigwedge_{j=1}^{i-1}(l_j < l_{j+1}) \land \bigwedge_{j \in \{1, \ldots, i\}} (\sigma_j \in \Sigma_L) \land \bigwedge_{0 \leq j \leq k-1}(\sigma_j \in \Sigma_H) \]

Then we define \([M]_k^i\) to encode the execution of the system after inputting \(\text{purge}_L(\alpha)\).

\[ [M]_L^i = I(s_1) \land \bigwedge_{j=1}^{i} R(s_j, \sigma_j, s_{j+1}) \]

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, \ldots, \sigma_{k-1}, s_k} I(s_0) \land \bigwedge_{i=0}^{k-1} R(s_i, \sigma_i, s_{i+1}) \land \forall s_1, \ldots, s_{k+1}, ([H]_k^i \land [M]_L^i \rightarrow O_L(s_k) \neq O_L(s_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.** \(\text{loopfree}(s_0^{D^2}, s_0, \ldots, s_{k-1}, s_k^{D^2}) = \bigwedge_{i=0}^{k-1} R^2(s_i^{D^2}, \sigma_i, s_{i+1}^{D^2}) \land \bigwedge_{0 \leq i < j \leq k}(s_i^{D^2} \neq s_j^{D^2}).\)

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

**Algorithm 2.** Checking Generalized Noninterference based QBF

\[
\begin{align*}
&k = 1 \\
&\text{While } I(s_0^{D^2}) \land \text{loopfree}(s_0^{D^2}, s_0, \ldots, s_{k-1}, s_k^{D^2}) \\
&\text{is satisfiable do} \\
&\quad k = k + 1 \\
&\text{End While} \\
&\text{return True}
\end{align*}
\]

**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 \(i\)th 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 \(s_0^{D^2}, s_0, s_1^{D^2}\), 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: \(\forall s_0^{D^2}(\forall s_0 \forall s_1^{D^2}(R(s_0^{D^2}, s_0, s_1) = s_1^{D^2} \land \forall s \in s_0^{D^2}(1) \exists s' \in s_0^{D^2}(2)(O_L(s) = O_L(s')) \rightarrow \forall s_1 \in s_1^{D^2}(1) \exists s' \in s_1^{D^2}(2)(O_L(s_1) = O_L(s'))\).
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 \( s^0_D, \sigma_0, \ldots, \sigma_{k-1}, s^D_k \), if \( I(s^0_D) \) holds, then for each state \( s_i \) of \( s^D_i(1) \), there exists a state \( s'_i \in s^D_i(2) \) such that \( O_L(s_i) = O_L(s'_i) \) for all \( 0 \leq i \leq k \).

- Prove that for all paths \( s^0_D, \sigma_0, \ldots, \sigma_{k-1}, s^D_k, \sigma_k, s^{D_2} \), if for each state \( s_i \) of \( s^D_i(1) \), there exists a state \( s'_i \in s^D_i(2) \) 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 \( s^{D_2}_{k+1}(1) \), there exists a state \( s'_{k+1} \in s_k^{D_2}(2) \) 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 \( [M^{D_2}]_k = \bigwedge_{i=0}^{k-1} R^2(s^D_i, \sigma_i, s^D_{i+1}) \). Let \( [M,GNI]_{k^n} = \forall s^D_0 \forall \sigma_0 \ldots \forall \sigma_{k-1} \forall s^D_k \forall \sigma_k \forall s^{D_2}_{k+1} \big(([M^2]_{k+1} \land \bigwedge_{i=0}^{k-1} \forall s_i \in s^D_i(1) \exists s'_i \in s^D_i(2)(O_L(s_i) = O_L(s'_i) \land \bigwedge_{i=0}^{k-1} s^{D_2}_{k+1}(1) \exists s'_{k+1} \in s^{D_2}_{k+1}(2)(O_L(s'_{k+1}) = O_L(s_{k+1})))) \). It is easy to justify that \( [M,GNI]^{k^n} \) 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

\[
\begin{align*}
&k = 1 \\
&\text{While } I(s^0_D) \land \text{loopfree}(s^0_D, \sigma_0, \ldots, \sigma_{k-1}, s^D_k) \text{ is satisfiable do} \\
&\quad \text{if } [M,GNI]_k \text{ is satisfiable return the counterexample } \sigma_0 \ldots \sigma_{k-1} \\
&\quad \text{if } [M,GNI]_{k^n} \text{ is satisfiable return True} \\
&\quad k = k + 1 \\
&\text{End While} \\
&\text{return True}
\end{align*}
\]

**D. Example**

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 \( u_1, u_2 \) 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 $R(v_1,v_2,u_1,u_2,v_1',v_2') = (\neg v_1 \land v_2 \land \neg u_1 \land u_2 \land \neg v_1 \land v_2') \lor (\neg v_1 \land \neg v_2 \land u_1 \land u_2' \land \neg u_1 \land u_2 \land v_1 \land v_2) \lor (v_1 \land \neg v_2 \land u_1 \land u_2 \land v_1 \land \neg v_2') \lor (v_1 \land \neg v_2 \land u_1 \land u_2' \land v_1 \land v_2') \lor (v_1 \land v_2 \land u_1 \land \neg u_2 \land v_1 \land \neg v_2') \lor (v_1 \land v_2 \land u_1 \land \neg u_2' \land v_1 \land v_2') \lor (\neg v_1 \land v_2 \land u_1 \land \neg u_2 \land v_1 \land v_2') \lor (\neg v_1 \land v_2 \land u_1 \land \neg u_2' \land v_1 \land v_2').$

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 $O_L(s) = (\neg v_1 \land v_2 \land p) \lor (v_1 \land v_2 \land \neg p) \lor (v_1 \land v_2 \land p).$ The initial state can be encoded as $(\neg v_1 \land v_2).$

We consider whether there are counterexamples of length 3. Let $k = 3$. The variables $s_0,s_0',s_1,s_1',s_2,s_2',s_3,s_3'$ 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 $s_i[1]$ to represent the first boolean variable, $s_i[2]$ to represent the second boolean variable. Thus $[M]_2 = (\neg s_0[1] \land s_0[2]) \land R(s_0[1],s_0[2],s_0[1],s_0[2],s_1[1],s_1[2]) \land R(s_1[1],s_1[2],s_1[1],s_1[2],s_2[1],s_2[2]) \land R(s_2[1],s_2[2],s_3[1],s_3[2]).$

For the action sequence $\sigma = s_0\sigma_1s_2,$ $purge_L(\sigma) \in \{\epsilon, s_0\sigma_1s_2, s_0\sigma_1s_2\sigma_0s_0s_2s_0\sigma_1s_2\sigma_0s_2\sigma_0s_1s_2\sigma_0s_2s_0s_1s_2s_0s_1s_2s_0s_1s_2s_0s_1\}$.

In the following for each value of $purge_L(\sigma)$, we show how to encode counterexamples.

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

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

Now we show how to compute the overapproximation of the minimal counterexample length, i.e. the recurrence diameter of $M^D.$ 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 $R(s_0,xor1) = \{s_0,s_1\}, R^D(s_0^D,xor1) =$
TABLE I: Experiments with the length of the minimal counterexample 4

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Actions</th>
<th>$A_1$</th>
<th>$A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEV(1,4)</td>
<td>158</td>
<td>99</td>
<td>1.97</td>
<td>2.75</td>
</tr>
<tr>
<td>ELEV(2,4)</td>
<td>1062</td>
<td>299</td>
<td>52.68</td>
<td>32.43</td>
</tr>
<tr>
<td>ELEV(3,4)</td>
<td>7121</td>
<td>783</td>
<td>1711.24</td>
<td>364.83</td>
</tr>
<tr>
<td>ELEV(4,4)</td>
<td>43440</td>
<td>1939</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>MMTG(2,4)</td>
<td>817</td>
<td>114</td>
<td>14.01</td>
<td>12.73</td>
</tr>
<tr>
<td>MMTG(3,4)</td>
<td>7703</td>
<td>172</td>
<td>71.52</td>
<td>60.97</td>
</tr>
<tr>
<td>MMTG(4,4)</td>
<td>66309</td>
<td>232</td>
<td>1277.15</td>
<td>663.78</td>
</tr>
<tr>
<td>RING(3,4)</td>
<td>87</td>
<td>33</td>
<td>0.42</td>
<td>0.67</td>
</tr>
<tr>
<td>RING(5,4)</td>
<td>1290</td>
<td>55</td>
<td>16.30</td>
<td>14.38</td>
</tr>
<tr>
<td>RING(7,4)</td>
<td>17000</td>
<td>77</td>
<td>401.02</td>
<td>219.74</td>
</tr>
<tr>
<td>RING(9,4)</td>
<td>211528</td>
<td>99</td>
<td>N/A</td>
<td>783.18</td>
</tr>
</tbody>
</table>

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

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

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 2048MBByte memory. We chose Quantor [12] as the prover. All benchmarks used in the experiment were taken from [22]. They have been converted from communicating state machines to Non-deterministic 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 $I(s_0^{D^2}) \land loopfree(s_0^{D^2}, \sigma_0, ..., \sigma_{k-1}, s_k^{D^2})$ 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

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Actions</th>
<th>$A_1$</th>
<th>$A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEV(1,8)</td>
<td>158</td>
<td>99</td>
<td>25.28</td>
<td>18.57</td>
</tr>
<tr>
<td>ELEV(2,8)</td>
<td>1062</td>
<td>299</td>
<td>890.15</td>
<td>407.46</td>
</tr>
<tr>
<td>ELEV(3,8)</td>
<td>7121</td>
<td>783</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ELEV(4,8)</td>
<td>43440</td>
<td>1939</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>MGMT(2,8)</td>
<td>817</td>
<td>114</td>
<td>39.88</td>
<td>24.68</td>
</tr>
<tr>
<td>MGMT(3,8)</td>
<td>7703</td>
<td>172</td>
<td>418.95</td>
<td>313.01</td>
</tr>
<tr>
<td>MGMT(4,8)</td>
<td>66309</td>
<td>232</td>
<td>N/A</td>
<td>1912.76</td>
</tr>
<tr>
<td>RING(3,8)</td>
<td>87</td>
<td>33</td>
<td>6.35</td>
<td>112.39</td>
</tr>
<tr>
<td>RING(5,8)</td>
<td>1290</td>
<td>55</td>
<td>189.38</td>
<td>113.19</td>
</tr>
<tr>
<td>RING(7,8)</td>
<td>17000</td>
<td>77</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(9,8)</td>
<td>211528</td>
<td>99</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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](image)

We now evaluate the induction proof component $[M, G N I]_{k}^{N}$. 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 satisfy 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 $N_1 = (S_1, \Sigma_1, \Sigma_L, \Sigma_H, R_1, AP_1, O_L_1)$, $N_2 = (S_2, \Sigma_2, \Sigma_L, \Sigma_H, R_2, AP_2, O_L_2)$. If $S_1 \cap \Sigma_2 = \emptyset$, we define $N_1 \oplus N_2$ as follows: $(S_1 \cup S_2, \Sigma_1 \cup \Sigma_2, \Sigma_L \cup \Sigma_L, \Sigma_H \cup \Sigma_H, R_1 \cup R_2, AP_1 \cup AP_2, O_L)$, 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: $\{M = b_1 \oplus b_2 | b_1, b_2 \in \{M_1, M_2, M_3\}\}$.

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 QBK-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

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Actions</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEV(1)⊕M₂</td>
<td>164</td>
<td>101</td>
<td>215.22</td>
</tr>
<tr>
<td>ELEV(2)⊕M₂</td>
<td>1068</td>
<td>301</td>
<td>N/A</td>
</tr>
<tr>
<td>ELEV(3)⊕M₂</td>
<td>7127</td>
<td>785</td>
<td>N/A</td>
</tr>
<tr>
<td>ELEV(4)⊕M₂</td>
<td>43446</td>
<td>1941</td>
<td>N/A</td>
</tr>
<tr>
<td>MMGT(2)⊕M₂</td>
<td>822</td>
<td>117</td>
<td>196.15</td>
</tr>
<tr>
<td>MMGT(3)⊕M₂</td>
<td>7708</td>
<td>175</td>
<td>N/A</td>
</tr>
<tr>
<td>MMGT(4)⊕M₂</td>
<td>66315</td>
<td>234</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(3)⊕M₂</td>
<td>93</td>
<td>35</td>
<td>94.41</td>
</tr>
<tr>
<td>RING(5)⊕M₂</td>
<td>1296</td>
<td>57</td>
<td>2249.38</td>
</tr>
<tr>
<td>RING(7)⊕M₂</td>
<td>17006</td>
<td>79</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(9)⊕M₂</td>
<td>211534</td>
<td>101</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Actions</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEV(1)⊕M₃</td>
<td>166</td>
<td>101</td>
<td>1155.73</td>
</tr>
<tr>
<td>ELEV(2)⊕M₃</td>
<td>1070</td>
<td>301</td>
<td>N/A</td>
</tr>
<tr>
<td>ELEV(3)⊕M₃</td>
<td>7129</td>
<td>785</td>
<td>N/A</td>
</tr>
<tr>
<td>ELEV(4)⊕M₃</td>
<td>43448</td>
<td>1941</td>
<td>N/A</td>
</tr>
<tr>
<td>MMGT(2)⊕M₃</td>
<td>825</td>
<td>116</td>
<td>1763.39</td>
</tr>
<tr>
<td>MMGT(3)⊕M₃</td>
<td>7711</td>
<td>174</td>
<td>N/A</td>
</tr>
<tr>
<td>MMGT(4)⊕M₃</td>
<td>66317</td>
<td>234</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(3)⊕M₃</td>
<td>95</td>
<td>35</td>
<td>778.88</td>
</tr>
<tr>
<td>RING(5)⊕M₃</td>
<td>1298</td>
<td>57</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(7)⊕M₃</td>
<td>17008</td>
<td>79</td>
<td>N/A</td>
</tr>
<tr>
<td>RING(9)⊕M₃</td>
<td>211536</td>
<td>101</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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


