SCIENCE CHINA
Information Sciences

Symbolic algorithmic verification of intransitive generalized noninterference

CongHua ZHOU*, ZhiFeng LIU, HaiLing WU, Song CHEN and ShiGuang JU

Citation: SCIENCE CHINA Information Sciences **55**, 1650 (2012); doi: 10.1007/s11432-011-4372-y

View online: https://engine.scichina.com/doi/10.1007/s11432-011-4372-y

View Table of Contents: https://engine.scichina.com/publisher/scp/journal/SCIS/55/7

Published by the Science China Press

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2012 Vol. 55 No. 7: 1650–1665
doi: 10.1007/s11432-011-4372-y

Symbolic algorithmic verification of intransitive generalized noninterference

ZHOU CongHua*, LIU ZhiFeng, WU HaiLing, CHEN Song & JU ShiGuang

School of Computer Science and Telecommunication Engineering, Jiangsu University Zhenjiang 212013*, China*

Received November 2, 2009; accepted September 20, 2010; published online October 17, 2011

Abstract Generalized noninterference can be used to formulate transitive security policies, but is unsuitable for intransitive security policies. We propose a new information flow security property, which we call intransitive generalized noninterference, that enables intransitive security policies to be specified formally. Next, we propose an algorithmic verification technique to check intransitive generalized noninterference. Our technique is based on the search for counterexamples and on the window induction proof, and can be used to verify generalized noninterference. We further demonstrate that the search of counterexamples and induction proof can be reduced to quantified Boolean satisfiability. This reduction enables us to use efficient quantified Boolean decision procedures to perform the check of intransitive generalized noninterference. It also reduces spatial requirement by representing the space compactly, and improves the efficiency of the verification procedure.

Keywords intransitive generalized noninterference, quantified Boolean satisfiability, symbolic verification, multi-level security

Citation Zhou C H, Liu Z F, Wu H L, et al. Symbolic algorithmic verification of intransitive generalized noninterference. Sci China Inf Sci, 2012, 55: 1650–1665, doi: 10.1007/s11432-011-4372-y

1 Introduction

Protecting data confidentiality is the key problem in trusted computing [1, 2], and is usually solved by the method of multi-level security policy, which assigns to all subjects and objects in the system different security levels. In such a system, information flow from a user of lower security level to a user of higher security level is allowed, while the reverse is prohibited. However, as described in the literature [3], this method is not universally satisfactory. The multi-level security policy can prevent the direct flow of information, yet cannot prevent the indirect flow of information, such as covert channel flow [4–6] in secure operating systems. In fact, information flow comprises both the direct and indirect leakage of information. To allow for this, the technique based on information flow analysis is used in all cases to search for all kinds of potential information leakage.

In 1982, Goguen and Meseguer introduced the noninterference approach [7] to controlling the direct and indirect flow of information in deterministic systems. If a system satisfies the noninterference property, the approach claims that no illegal flow of information occurs in the system. Since many non-deterministic systems exist in practice, researchers later introduced the notion of noninterference for non-deterministic

*Corresponding author (email: chzhou@ujs.edu.cn)

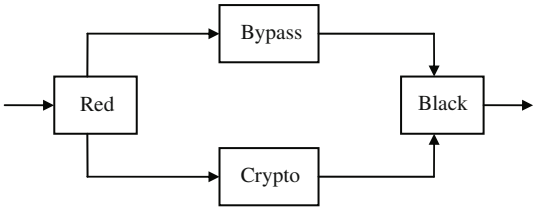

Figure 1 The channel-control security policy.

systems, from which emerged the study of generalized noninterference [8]. Generalized noninterference can formulate transitive information flow security polices only; that is, if information flows from H to D and from D to L are allowed, then information flow from H to L is also allowed. There are, however, a number of practical security problems that apparently lie beyond the scope of generalized noninterference formulations, for example "channel-control", first formulated by Rushby [9].

Consider the typical channel-control problem shown in Figure 1. Plaintext messages arrive at the Red side of the controller; their bodies are sent through the encryption device (Crypto); their headers, which must remain in plaintext so that network switches can interpret them, are sent through the Bypass. Headers and encrypted bodies are reassembled at the Black side and sent out onto the network. The security policy we wish to specify here is the requirement that information flow from Red to Black occurs strictly through the Crypto and the Bypass. An important characteristic of many channel control policies is that the edges indicating allowed information flows are not transitive: information is allowed to flow from Red to Black via the Crypto and Bypass, but can not do so directly. Such intransitive security policy can not be formulated in terms of generalized noninterference. Our first task in this paper is to extend generalized noninterference to intransitive generalized noninterference (IGNI) such that intransitive security policies can be formulated.

The ultimate goal of introducing noninterference is to ensure that no corresponding illegal information flows in multi-level security systems. The problem of how to verify that a system satisfies noninterference is therefore of prime importance. The traditional verification method for noninterference is called "unwinding" [10]. In this method, an "unwinding theorem" is constructed, from which noninterference, a global constraint, is reduced to the local constraint involved in single-step state transitions. The method establishes the relationship between noninterference and single-step state transitions. Currently, the method can check whether a deterministic system satisfies noninterference, but is ineffective for non-deterministic systems. The main problem is that the method is sound, but not complete. By "sound", we mean that if the local constraint is satisfied, then the system satisfies noninterference, while "not complete" means that if the local constraint is not satisfied, then we cannot conclude the system does not satisfy noninterference.

In addition to "unwinding", some other verification methods have been developed. The verification of noninterference has been reduced to the language inclusion problem (in the case of generalized noninterference) [11], and to the reachability problem [12]. It has also been approached from a finite automata perspective [13]. Of these alternatives, the first is not complete, while the others are not applicable to non-deterministic systems.

Our second task is to propose a sound and complete verification method for IGNI, which is also suited to generalized noninterference. The method is designed on two basic verification strategies: refutation and proof. Our method has two advantages over "unwinding". Firstly, it does not require the construction and verification of local constraint , and hence is complete. Secondly, it provides a counterexample in the event that the system does not satisfy IGNI. This counterexample can assist in eliminating illegal information flow.

To improve the efficiency of the verification procedure and reduce the space requirement, we further explore quantified Boolean formula(QBF) [14]-based symbolic computation of the verification procedure. This exploration is encouraged by previous successful application of QBF in the verification of reliability and in hardware and software security [15–17]. The reduction decreases the space requirement of model checking, and improves the efficiency of the verification procedure, as exemplified by Tao et al. [18] and

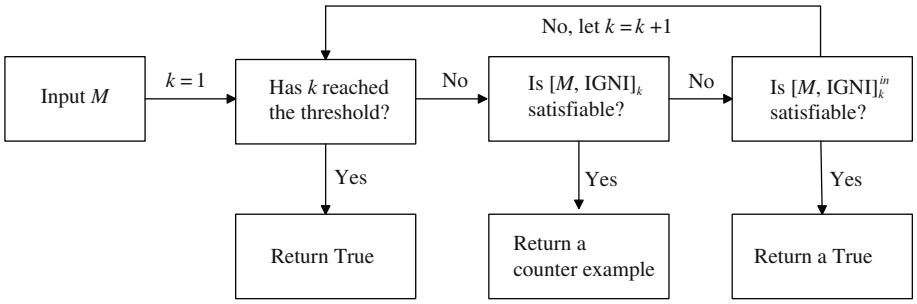

Figure 2 Symbolic algorithmic verification procedure of IGNI.

by Zhou et al. [19], who reduced the model checking problem of CTL* and the temporal knowledge logic CTLK, respectively, to QBF. In all instances in which QBF has been applied successfully, systems are defined as finite state machines. Since the systems we seek to verify are also finite state machines, reducing the verification of IGNI to QBF is a feasible option.

Our verification procedure, shown in Figure 2, consists of three basic steps:

1. Check the Bound: Determine whether the bound reaches the pre-computed threshold. If so, then claim that the system satisfies IGNI.

2. Search for Counterexamples by a QBF Solver: Reduce the existence problem of counterexamples of some length to the quantified Boolean formula satisfiability problem, i.e., there exist counterexamples of length k if and only if the quantified Boolean formula $[M, \text{IGNI}]_k$ is satisfiable ($[M, \text{IGNI}]_k$ encodes a counterexample of length k with boolean variables). If $[M, \text{IGNI}]_k$ is satisfiable, then we claim that the system does not satisfy IGNI, and return a counterexample.

3. Inductive Proof by a QBF Solver: Reduce the window induction proof to the quantified Boolean formula satisfiability problem, i.e., the window induction proof of the size of window k succeeds if and only if $[M, \text{IGNI}]_k^{\text{in}}$ is satisfiable ($[M, \text{IGNI}]_k^{\text{in}}$ encodes the induction proof of length k with Boolean variables). If $[M, \text{IGNI}]_k^{\text{in}}$ is satisfiable, then we claim that the system satisfies IGNI, else let $k = k + 1$, return to step 1.

2 Intransitive generalized noninterference

In this section we use a finite state machine to describe the dynamic behavior of a multi-level security system termed non-deterministic security label Kripke structure (NSLKS). We further extend generalized noninterference to IGNI such that intransitive security policies can be formulated. As in previous studies, without loss of generality we assume only three different security levels, denoted by H, D, and L. Let Domain= $\{H, D, L\}$.

Definition 1. An NSLKS M is a six-tuple $(S, s_{\text{in}}, \Sigma, R, \text{dom}, O)$, where,

- S is a finite set of states;
- $s_{\text{in}} \in S$ is an initial state;
- Σ is a finite set of actions;
- $R \subseteq S \times \Sigma \times S$ is a state transition relation;
- dom: $\Sigma \to$ Domain assigns to every action a security level;
- $O : S \times$ Domain $\to 2^{\text{AP}}$ is an observation function, here AP is a finite set of atomic propositions.

In real systems, an operation can be called by users of different security levels. The security level of the operation is decided by the user. Different users can be distinguished by assigning different names to the operation. Hence it is reasonable that in Definition 1 we assign to every action a security level. In multi-level security systems, since users of different security levels operate in parallel, actions of different security levels are also implemented in parallel. In NSLKS, by introducing the composition pattern used

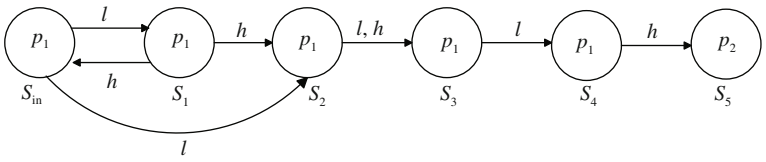

Figure 3 Non-deterministic security label Kripke structure M_1.

in CSP, we simulate the parallel run between actions by defining alternative implementation of actions. The dynamic behavior of a multi-level security system is accurately simulated by this formulation.

The security policy is formulated with a reflexive relation over Domain, denoted by $\gg$. $\not\gg$ denotes the complement of $\gg$, i.e. $\not\gg = (\text{Domain} \times \text{Domain}) \setminus \gg$. We call $\gg$ the interference relation, and $\not\gg$ the noninterference relation. If $\gg$ is intransitive, we let the security policy formulated by $\gg$ be intransitive also. To define IGNI formally, we must identify actions which should not be deleted. To this end, we define the following function *sources*. First, however, for the action sequences $\alpha = \alpha_1 \cdots \alpha_n$, $\beta = \beta_1 \cdots \beta_m$, we define $\alpha \circ \beta = \alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_m$.

Definition 2. Define $sources : \Sigma^* \times \text{Domain} \to 2^{\text{Domain}}$, here

$$\text{sources}(\varepsilon, u) = \{u\},$$

and

$$\text{sources}(a \circ \alpha, u) = \begin{cases} \text{sources}(\alpha, u) \cup \{\text{dom}(a)\}, & \text{if } \exists v : v \in \text{sources}(\alpha, u) \wedge \text{dom}(a) \gg v, \\ \text{sources}(\alpha, u), & \text{otherwise.} \end{cases}$$

Essentially, $v \in \text{sources}(\alpha, u)$ means $u = v$ or that there exists a subsequence of α in which security levels are $w_1, \ldots, w_n$ in turn such that $w_1 \gg w_2 \gg \cdots \gg w_n$, $v = w_1$, $u = w_n$. Before checking whether the action a implemented before α affects u, we consider whether there exists $v \in \text{sources}(\alpha, u)$ such that $\text{dom}(a) \gg v$. Note that the following relations always hold: $\text{sources}(\alpha, u) \subseteq \text{sources}(a \circ \alpha, u), u \in \text{sources}(\alpha, u)$.

Definition 3. Define ipurge: $\Sigma^* \times \text{Domain} \to \Sigma^*$, here

$$\text{ipurge}(\varepsilon, u) = \varepsilon,$$

and

$$\text{ipurge}(a \circ \alpha, u) = \begin{cases} a \circ \text{ipurge}(\alpha, u), & \text{if } \text{dom}(a) \in \text{sources}(a \circ \alpha, u), \\ \text{ipurge}(\alpha, u), & \text{otherwise.} \end{cases}$$

Informally, $\text{ipurge}(\alpha, u)$ is a subsequence of α, obtained by deleting actions which do not interfere with u. Thus the security property can be defined by ipurge: given a security policy $\gg$, the system is secure if for any action sequence α, the user of security level u exhibits the same dynamic behavior when implementing α as when implementing $\text{ipurge}(\alpha, u)$.

Definition 4. We say that an NSLKS M satisfies IGNI, denoted by $M \models \text{IGNI}$, if and only if for any action sequence σ, and for any state $s \in s_{\text{in}} \bullet \sigma$, there exists $s' \in s_{\text{in}} \bullet \text{ipurge}(\sigma, L)$ such that $O(s, L) = O(s', L)$. Here $s_{\text{in}} \bullet \sigma$ is a set of states produced by implementing σ starting from s_{in}.

Consider the security policy $\gg = \{(H, H), (H, D), (D, D), (D, L), (D, H), (L, L), (L, D), (L, H)\}$, and the system M_1 shown in Figure 3. In M_1, each state is labeled by $O(s, L)$, and $\text{dom}(h) = H$, $\text{dom}(l) = L$, $\text{dom}(d) = D$. $\gg$ does not allow information flow from H to L directly. However, the information can be transferred by D, i.e. H can interfere with D, and D can interfere with L. For the action sequence $\sigma =$lhdlh, by Definition 3 we have $\text{ipurge}(\sigma, L) =$lhdl. Also $s_{\text{in}} \bullet \sigma = \{s_5\}$, $s_{\text{in}} \bullet \text{ipurge}(\sigma, L) = \{s_4\}$. Hence for $s \in s_{\text{in}} \bullet \sigma$, there does not exist $s' \in \{s_4\}$ such that $O(s, L) = O(s', L)$; that is, M_1 does not satisfy IGNI.

3 Counterexample guided algorithmic verification of IGNI

From a methodological viewpoint, the verification method can be divided into two categories, proof and refutation. Proof is demonstration that if the system satisfies some constraint, then the security property holds. Refutation is demonstration that the security property does not hold by finding counterexamples. As discussed in the introduction, "unwinding" cannot be used to verify generalized noninterference, hence is expected to be of little use in verifying IGNI. In this section, we tackle IGNI verification using the refutation approach, based on search for counterexamples. This method is not only sound, but also complete. At the same time, the method can be used to verify generalized noninterference.

Definition 5 (Counterexample)**.** Let $M = (S, s_{\text{in}}, \Sigma, R, \text{dom}, O)$ be an NSLKS. A finite action sequence $\sigma \in \Sigma^*$ is called a counterexample making IGNI invalid, if and only if there exists a state $s \in s_{\text{in}} \bullet \sigma$, such that for any $s' \in s_{\text{in}}\bullet$ ipurge(σ, L), $O(s, L) \neq O(s', L)$.

By the definition of IGNI, it is readily concluded that M does not satisfy IGNI if and only if there exists a counterexample. Thus, we can verify IGNI directly by checking all finite action sequences: let $k = 1, 2, 3, \ldots$, check whether there exists a counterexample of length k. If the counterexample exists, then we can claim that M does not satisfy IGNI, otherwise continue the search for counterexamples by incrementing k

Now we encounter the following problem: if $M \models$ IGNI, the above verification procedure will not terminate. We therefore set a threshold length for the shortest counterexample, such that if counterexamples exist at all, then a counterexample of length not more than the threshold exists. Thus we need only check whether there exist counterexamples of length not more than the threshold, and procedure termination is guaranteed.

Definition 6. Let $M = (S, s_{\text{in}}, \Sigma, R, \text{dom}, O)$ be an NSLKS. Define a deterministic system $M^D = (S^D, s^D_{\text{in}}, \Sigma, R^D)$ on M, where

- $S^D = 2^S$;
- $s^D_{\text{in}} = \{s_{\text{in}}\}$;
- $R^D : S^D \times \Sigma \to S^D$ is a state transition function, here $s^D_1 = R^D(s^D, \alpha)$ if and only if $s^D_1 = \{s' | \exists s \in s^D((s, \alpha, s') \in R)\}$.

Definition 6 shows how a behavior equivalent deterministic system can be derived from an NSLKS M. In essence, we reduce the verification of IGNI to the verification of some property on the deterministic system. Next we construct the deterministic system M^D_1 on M_1, as shown in Figure 3. First by Definition 6, in M^D_1 we declare the initial state to be $s^D_{\text{in}} = \{s_{\text{in}}\}$. Then, we compute the state transition relation starting from s^D_{in} induced by action l: since s^D_{in} includes only one element s_{in} and after implementing l in M_1 s_{in} has two successors s_1 and s_2, the successor of s^D_{in} is $s^D_1 = \{s_1, s_2\}$ after implementing l. For s^D_1, since s_1 has two successors s_{in}, s_2 and s_2 has one successor s_3 after implementing action h, $s^D_2 = \{s_{\text{in}}, s_2, s_3\}$. Other state transition relations can be constructed in the same way. The final deterministic system M^D_1 is shown in Figure 4.

Definition 7. Let $M^D = (S^D, s^D_{\text{in}}, \Sigma, R^D)$, define the double construction $M^{D^2} = (S^{D^2}, s^{D^2}_{\text{in}}, \Sigma, R^{D^2})$ on M^D, here

- $S^{D^2} = S^D \times S^D$;
- $s^{D^2}_{\text{in}} = (s^D_{\text{in}}, s^D_{\text{in}})$;
- $R^{D^2} \subseteq S^{D^2} \times \Sigma \times S^{D^2}$ is a state transition relation. For the action α of security level L or D, $((s^D_1, s^D_2), \alpha, (s^D_3, s^D_4)) \in R^{D^2}$ if and only if $s^D_3 = R^D(s^D_1, \alpha), s^D_4 = R^D(s^D_2, \alpha)$; for the action β of security level H $((s^D_1, s^D_2), \beta, (s^D_3, s^D_4)) \in R^{D^2}$ if and only if $s^D_3 = R^D(s^D_1, \beta) \wedge (s^D_4 = R^D(s^D_2, \beta) \vee s^D_4 = s^D_2)$.

Note that in M^{D^2}, for the action β of security level H, we add an additional local transition relation of the state to itself. The goal of defining M^{D^2} is to merge two paths produced by implementing two action sequences σ and ipurge(σ) into one path such that the verification of IGNI can be reduced to the reachability problem between states in M^{D^2}. Now we compute the double construction on M^D_1 shown in Figure 4. First for the initial state $s^{D^2}_{\text{in}} = (s^D_{\text{in}}, s^D_{\text{in}})$, if action l is implemented, by Definition 7, two elements in $s^{D^2}_{\text{in}}$ are changed. Since $R^D(s^D_{\text{in}}, l) = s^D_1$, $s^{D^2}_1 = (s^D_1, s^D_1)$ and $(s^{D^2}_{\text{in}}, l, s^{D^2}_1) \in R^{D^2}$.

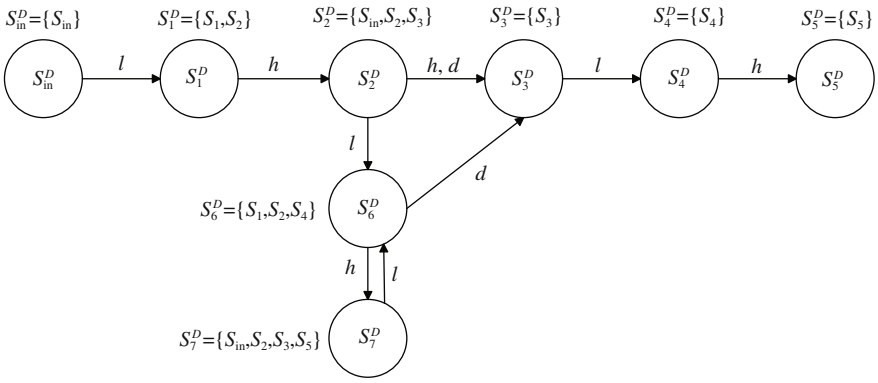

Figure 4 The deterministic construction M_1^D on M_1.

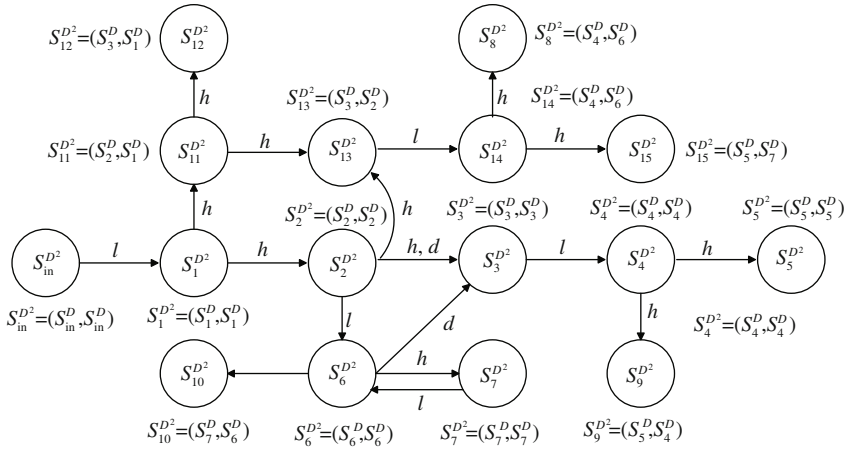

Figure 5 The double construction on M_1^D.

For the state $s_1^{D^2}$, if action h is implemented, the first element s_1^D in $s_1^{D^2}$ is changed according to the transition relation in M_1^D. For the second element, two cases arise: the state is not changed, or is changed according to the transition relation in M_1^D. Thus, after implementing h, $s_1^{D^2}$ has two successors, i.e. $s_2^{D^2}=(s_2^D,s_2^D)$ and $s_{11}^{D^2}=(s_2^D,s_1^D)$. The final system M^{D^2} is shown in Figure 5.

The next key problem is the computation of ipurge(σ). Lemma 1 below shows that σ can be divided into two parts by the last action of security level D in σ. These two parts are then computed separately.

Lemma 1. Let $\sigma=\alpha\circ d\circ\beta$, here $\mathrm{dom}(d)=D$ and in β the security level of every action is L or H, then $\mathrm{ipurge}(\sigma)=\alpha\circ d\circ\mathrm{ipurge}(\beta)$.

Proof. Assume $\alpha=\alpha_1\cdots\alpha_m$, $\beta=\beta_1\cdots\beta_n$. We first compute sources(σ) recursively. By the definition of sources, if $\mathrm{dom}(\beta_n)=L$ then $\mathrm{sources}(\beta_n,L)=\mathrm{sources}(\varepsilon,L)\cup\mathrm{dom}(\beta_n)=\{L\}$, otherwise $\mathrm{sources}(\beta_n,L)=\mathrm{sources}(\varepsilon,L)=\{L\}$. Thus $\mathrm{sources}(\beta_n,L)=\{L\}$. In the same way, we obtain $\mathrm{sources}(\beta_{n-1}\beta_n,L)=\{L\},\ldots,\mathrm{sources}(\beta,L)=\{L\}$.

For the sequence $d\circ\beta$, since $\mathrm{dom}(d)\gg L$, $\mathrm{sources}(d\circ\beta,L)=\mathrm{sources}(\beta,L)\cup\{\mathrm{dom}(d)\}=\{D,L\}$. For $\alpha_m\circ d\circ\beta$, since $H\gg D, L\gg D$, $\mathrm{sources}(d\circ\beta,L)=\mathrm{sources}(\beta,L)\cup\{\mathrm{dom}(\alpha_m)\}$. Hence if $\mathrm{dom}(\alpha_i)=H(1\leqslant i\leqslant m)$, then $\mathrm{sources}(\alpha_i\cdots\alpha_m\circ d\circ\beta,L)=\{H,L,D\}$. That is, if an action α_i of security level H exists, then $\mathrm{sources}(\sigma,L)=\{H,L,D\}$, otherwise $\mathrm{sources}(\sigma,L)=\{L,D\}$.

Next we compute ipurge(β). From the computation of sources, we have that if $\mathrm{dom}(\alpha_1)=H$, then $\mathrm{sources}(\sigma,L)=\{H,D,L\}$, i.e. $\mathrm{dom}(\alpha_1)\in\mathrm{sources}(\sigma,L)$. If $\mathrm{dom}(\alpha_1)=L$ then $L\in\mathrm{sources}(\sigma,L)$. Thus $\mathrm{dom}(\alpha_1)\in\mathrm{sources}(\sigma,L)$. According to the definition of ipurge(σ), $\mathrm{ipurge}(\sigma)=\alpha_1\circ\mathrm{ipurge}(\alpha_2\cdots\alpha_m\circ d\circ$

β). In the same way, ipurge(σ) = $\alpha_1\alpha_2\circ$ ipurge($\alpha_3\cdots\alpha_m\circ d\circ\beta$). Finally we have that ipurge(σ) = $\alpha\circ d\circ$ ipurge(β).

For the action sequence β, if the security level of every action in β is L or H, then sources(β, L) = $\{L\}$. Hence ipurge(β) is obtained by deleting actions of the security level H from β. In the following we discuss the relation between the verification of IGNI and the reachability problem on M^{D^2}.

Definition 8 (D reachable)**.** Let M be an NSLKS, M^D be the deterministic construction on M, and M^{D^2} be the double construction on M^D. In M^{D^2} we say that (s^D, s^D) is D reachable from $(s^D_{\text{in}}, s^D_{\text{in}})$ if and only if $s^D = s^D_{\text{in}}$, or if there exists an action sequence $\alpha_1\cdots\alpha_n$ ending with the action of the security level D, such that $((s^D_{\text{in}}, s^D_{\text{in}}), \alpha_1, (s^D_1, s^D_1)) \in R^{D^2}$, $((s^D_i, s^D_i), \alpha_{i+1}, (s^D_{i+1}, s^D_{i+1})) \in R^{D^2} (1 \leqslant i \leqslant n-1), s^D_{n+1} = s^D$.

Intuitively, if (s^D, s^D) is D reachable from $(s^D_{\text{in}}, s^D_{\text{in}})$, then there exists an action sequence σ ending with the action of security level D such that $s^D = s^D_{\text{in}} \bullet \sigma$.

Definition 9 ((H, L) reachable)**.** Let M be an NSLKS, M^D be the deterministic construction on M, and M^{D^2} be the double construction on M^D. We say that (r^D, t^D) is (H, L) reachable from (s^D, s^D) if and only if there exists an action sequence σ in which the security level of every action is L or H such that $r^D = s^D \bullet \sigma$,$t^D = s^D \bullet$ ipurge(σ).

Lemma 2. Let M be an NSLKS, M^D be the deterministic construction on M, and M^{D^2} be the double construction on M^D. M does not satisfy IGNI, if and only if there exist (s^D, s^D) and (r^D, t^D) such that (s^D, s^D) is D reachable from $(s^D_{\text{in}}, s^D_{\text{in}})$, (r^D, t^D) is (H, L) reachable from (s^D, s^D), and if, in addition, there exists $s \in r^D$ such that for any $t \in t^D, O(s, L) \neq O(t, L)$.

The proof of Lemma 2, which is trivial, can be obtained directly from M^D, M^{D^2}, the definition of counterexample, and Lemma 1. Lemma 2 shows that we can bound the threshold of counterexamples by the reachable depth of M^{D^2}. For the system M, let $|M|$ be the number of states in M. Then $|M^{D^2}| = |M^D|^2 = |2^{|M|}|^2 = 2^{2|M|}$. In M^{D^2}, every reachable state is attainable from the initial state within $|M^{D^2}|$ steps. By Lemma 2 we have the following theorem.

Theorem 1 (The completeness theorem)**.** Let M be an NSLKS. M does not satisfy IGNI if and only if there exists a counterexample of length not more than $2^{2|M|+1}$.

Proof. The necessity is clear. Now we consider the sufficiency. By Lemma 2, since M does not satisfy IGNI, in M^{D^2} there exist (s^D, s^D) and (r^D, t^D) such that (s^D, s^D) is D reachable from $(s^D_{\text{in}}, s^D_{\text{in}})$, (r^D, t^D) is (H, L) reachable from (s^D, s^D), and there exists $s \in r^D$ such that for any $t \in t^D$, $O(s, L) \neq O(t, L)$. By the definition of $|M^{D^2}|$, (s^D, s^D) is D reachable from $(s^D_{\text{in}}, s^D_{\text{in}})$ within $2^{2|M|}$ steps. Assume that the corresponding action sequence is σ_1. (r^D, t^D) is (H, L) reachable from (s^D, s^D) within $2^{2|M|}$ steps. Assume that another corresponding action sequence is σ_2. Therefore $\sigma_1 \circ \sigma_2$ is the counterexample and the length of $\sigma_1 \circ \sigma_2$ is not more than $2^{2|M|+1}$.

Theorem 1 shows that to verify whether M satisfies IGNI, we need only check whether there exist counterexamples of length not more than $2^{2|M|+1}$. In the actual verification process, it is inefficient to check unnecessary loops before termination. In theory, we should compute the shortest path between any two states. Applying theory in practice, however, renders the search for the shortest path is as hard as checking M in terms of computer run-time. Hence we focus on the over-approximation of the shortest path based on graph theory.

Definition 10. In the double construction M^{D^2}, we call a finite path $s^{D^2}_0, \sigma_0, \ldots, \sigma_{k-1}, s^{D^2}_k$ a loop free path if and only if $s^{D^2}_i \neq s^{D^2}_j (0 \leqslant i < j \leqslant k)$.

Definition 11 (Recurrence diameter)**.** Given a double construction M^{D^2}, the recurrence diameter of M^{D^2}, denoted by rd(M^{D^2}), is the length of the longest loop free path between any two reachable states.

Lemma 3. In the double construction M^{D^2}, if s is reachable from s' then s must be reachable from s' within rd(M^{D^2}) steps.

The proof of Lemma 3 follows from the definition of rd(M^{D^2}). Lemma 3 shows that the threshold can be bounded by rd(M^{D^2}).

Theorem 2. Let M be an NSLKS. M does not satisfy IGNI if and only if there exists a counterexample of length not more than $2 \times \mathrm{rd}(M^{D^2}) + 1$.

Proof. The necessity is clear. Now we consider the sufficiency. By Lemma 2, since M does not satisfy IGNI, in M^{D^2} there exist (s^D, s^D) and (r^D, t^D) such that (s^D, s^D) is D reachable from $(s^D_{\mathrm{in}}, s^D_{\mathrm{in}})$, (r^D, t^D) is (H, L) reachable from (s^D, s^D). In addition, there exists $s \in r^D$ such that for any $t \in t^D$, $O(s, L) \neq O(t, L)$. By Lemma 3, (s^D, s^D) must be D reachable from $(s^D_{\mathrm{in}}, s^D_{\mathrm{in}})$ within $rd(M^{D^2})$ steps. Assume that the corresponding action sequence is σ_1. (r^D, t^D) must be (H, L) reachable from (s^D, s^D) within $\mathrm{rd}(M^{D^2})$ steps. Assume another corresponding action sequence to be σ_2. Therefore $\sigma_1 \circ \sigma_2$ is the counterexample, and it's length is not more than $2 \times \mathrm{rd}(M^{D^2}) + 1$.

Based on Theorem 2, a sound and complete verification procedure of IGNI is shown below:

Algorithm 1 Verify IGNI based on the search for counterexamples.

{
$k = 1$
While $k \leqslant 2 \times rd(M^{D^2}) + 1$ do
    {
    if there exists a counterexample of length k, then return "False"
    otherwise, let $k = k + 1$
    }
return "True"
}

4 Symbolic computation of algorithmic verification

In this section, we explore how to reduce the search for counterexamples to the satisfiability problem of QBF such that with the help of a QBF solver we can verify IGNI efficiency.

4.1 Quantified Boolean formula

QBF is an extension of Boolean formulae by the introduction of quantifiers. Two kinds of quantifiers exist: the existential quantifier $\exists$, and the universal quantifier $\forall$. For example, $\forall x \exists y \exists z((x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z))$ is a QBF. Formally, QBF is defined as follows:

Definition 12 (Quantified Boolean formula)**.**

- if f is a Boolean formula, then f is a QBF;
- if f is a QBF, x is a Boolean variable, then $\exists x f$ and $\forall x f$ are QBFs;
- if f and g are QBFs, then $\neg f$,$f \wedge g$, $f \to g$ are QBFs.

Each QBF has an equivalent prefix form: $\Phi = Q_1 x_1 \cdots Q_n x_n \phi$, where $1 \leqslant i \leqslant n, Q_i \in \{\exists, \forall\}$ and x_i is a Boolean variable. Formally, the truth of QBF is defined recursively as follows:

- if there are no quantifies in Φ, i.e. Φ is a Boolean formula, the truth of Φ is decided by its truth table;
- Φ equals $\exists x \phi$, then Φ is satisfiable if and only if Φ_x or Φ_{-x} are satisfiable, here $\Phi_x(\Phi_{-x})$ is obtained from Φ by replacing x in Φ with True(False);
- if Φ equals $\forall x \phi$, then Φ is satisfiable if and only if Φ_x and Φ_{-x} are satisfiable, here $\Phi_x(\Phi_{-x})$ is obtained from Φ by replacing x in Φ with True(False).

For example $\forall x \exists y(x \leftrightarrow y)$ is satisfiable. By the truth of QBF, $\forall x \exists y(x \leftrightarrow y)$ is satisfiable if and only if $\exists y(1 \leftrightarrow y)$ and $\exists y(0 \leftrightarrow y)$ are satisfiable, while $\exists y(1 \leftrightarrow y)$ is satisfiable if and only if $1 \leftrightarrow 1$ or $1 \leftrightarrow 0$ is satisfiable. Clearly, $1 \leftrightarrow 1$ is satisfiable. Given a QBF Φ, to check whether Φ is satisfiable is called the satisfiability problem of QBF, or simply the QBF problem.

4.2 Symbolic representation of NSLKS

In this subsection we explore the representation of an NSLKS symbolic. To represent an NSLKS $(S, s_{\text{in}}, \Sigma, R, \text{dom}, O)$, we must describe the set of states S, the set of actions Σ, the transition relation R, and the observation function O. Note that we need only focus on the observation of users of security level L. Without loss of generality, we assume that the number of states is $2^m (m > 0)$, and that the number of actions of every security level is $2^n (n > 0)$. AP= $\{p_1, \ldots, p_k\} (k > 0)$.

Let $\phi : S \to \{0,1\}^m$ be a bijective function. ϕ establishes the map between S and Boolean vectors of length m. The initial state is represented by $\phi^{-1}(s_{\text{in}})$, denoted $I(s_{\text{in}})$. $\psi : \Sigma \to \{0,1\}^{n+2}$ is an injective function, and satisfies $\psi : \Sigma_L \to \{0\} \times \{0\} \times \{0,1\}^n, \psi : \Sigma_H \to \{0\} \times \{1\} \times \{0,1\}^n, \psi : \Sigma_D \to \{1\} \times \{1\} \times \{0,1\}^n$, where $\Sigma_U (U \in \{L, H, D\})$ is the set of actions of security level U. ψ establishes the map between the set of actions and Boolean vectors of length $n + 2$.

For the state s assume $\phi^{-1} = (b_1, \ldots, b_m)$, then s can be encoded with a Boolean formula $\bigwedge_{b_i=1}^{1 \leqslant i \leqslant m} b_i' \wedge \bigwedge_{b_i=0}^{1 \leqslant i \leqslant m} \neg b_i'$. Since $(b_1, \ldots, b_m)$ is the only assignment making $\bigwedge_{b_i=1}^{1 \leqslant i \leqslant m} b_i' \wedge \bigwedge_{b_i=0}^{1 \leqslant i \leqslant m} \neg b_i'$ true, the encoding is reasonable. For simplicity in the following section, we use the notation $\phi^{-1}(s)$ to represent $\bigwedge_{b_i=1}^{1 \leqslant i \leqslant m} b_i' \wedge \bigwedge_{b_i=0}^{1 \leqslant i \leqslant m} \neg b_i'$. The transition relation $(s, \alpha, s') \in R$ is encoded with $\phi^{-1}(s) \wedge \psi^{-1}(\alpha) \wedge \phi^{-1}(s')$. The label function $O(s, L)$ is encoded with $\phi^{-1}(s) \wedge \bigwedge_{p \in O(s,L)} p \wedge \bigwedge_{p \notin O(s,L)} \neg p$. To simplify the notation, we let $O(s, L)$ represent $\phi^{-1}(s) \wedge \bigwedge_{p \in O(s,L)} p \wedge \bigwedge_{p \notin O(s,L)} \neg p$. $O(s, L) \neq O(s', L)$ is encoded with

$$\phi^{-1}(s) \wedge \phi^{-1}(s') \wedge \neg \Bigg(\bigwedge_{p \in O(s,L)} p \wedge \bigwedge_{p \notin O(s,L)} \neg p \leftrightarrow \bigwedge_{p \in O(s',L)} p \wedge \bigwedge_{p \notin O(s',L)} \neg p \Bigg).$$

Now we show how to represent an NSLKS symbolic by M_1 shown in Figure 3. Since in M_1 there are six states, we need a three bit Boolean vector $(s(1), s(2), s(3))$ to encode the state s: $(0,0,0) \leftrightarrow s_{\text{in}}$, $(0,0,1) \leftrightarrow s_1$, $(0,1,0) \leftrightarrow s_2$, $(0,1,1) \leftrightarrow s_3$, $(1,0,0) \leftrightarrow s_4$,$(1,0,1) \leftrightarrow s_5$. Thus the initial state s_{in} is encoded with $\neg s(1) \wedge \neg s(2) \wedge s(3)$. Since for each security level there is only one action, we need a three bit Boolean vector (c_1, c_2, c_3) to encode actions, in which the first two bits represent the security level. The detailed encoding for actions is as follows: $(0,0,0) \leftrightarrow l$,$(0,1,0) \leftrightarrow h$, $(1,1,0) \leftrightarrow d$. The local transition relation, for example $(s_1, h, s_2) \in R$, is encoded with $\neg s_1(1) \wedge \neg s_1(2) \wedge s_1(3) \wedge \neg c_1 \wedge c_2 \wedge \neg c_3 \wedge \neg s_2(1) \wedge s_2(2) \wedge s_2(3)$. The global transition relation is the disjunction of all local transition relations, that is $R = (s_{\text{in}}, l, s_1) \vee (s_{\text{in}}, l, s_2) \vee (s_1, h, s_{\text{in}}) \vee (s_1, h, s_2) \vee (s_2, d, s_3) \vee (s_2, h, s_3) \vee (s_3, l, s_4) \vee (s_4, h, s_5)$. The label function $O(s_1, L)$ is encoded with $\neg s_1(1) \wedge \neg s_1(2) \wedge s_1(3) \wedge p_1 \wedge \neg p_2$. The global label function O is encoded with $O(s_{\text{in}}, L) \vee O(s_1, L) \vee O(s_2, L) \vee O(s_3, L) \vee O(s_4, L) \vee O(s_5, L)$.

4.3 Reduction

In the previous section we presented the verification of IGNI by searching for counterexamples. Here we explore how to reduce the search for counterexamples to the QBF problem such that the verification procedure can be performed by an efficient QBF solving program.

Given an NSLKS M and the length k of the counterexample, we construct a QBF $[M, \text{IGNI}]_k$. Variables $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ encode a path in which states and actions are alternating. Essentially $[M, \text{IGNI}]_k$ encodes the constraint on $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ such that $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ is a counterexample of length k. To construct $[M, \text{IGNI}]_k$, we first define $[M]_k$ to restrict the sequence $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ to that of a valid path. Then we translate the counterexample into a QBF.

Definition 13. Given a NSLKS M and a positive integer k, define $[M]_k := \bigwedge_{i=0}^{k-1} R(s_i, \alpha_i, s_{i+1})$.

Recalling the definition of the counterexample, we note that for the action sequence $\alpha = \alpha_0 \cdots \alpha_{k-1}$, we need to compute $\text{ipurge}(\alpha_0 \cdots \alpha_{k-1}, L)$. This computation has been outlined in Lemma 1. First, we find the last action of security level D in α, and let m be the position of the action in α. If no actions of security level D are present in α, then let $m = -1$. Second, we compute $\text{ipurge}(\alpha_{m+1} \cdots \alpha_{k-1}, L)$. Assume that in $\alpha_{m+1} \cdots \alpha_{k-1}$ there are i actions of security level L. We introduce the notation $[H]_k^{m,i}$ to encode the distribution of these i actions.

If $i = 0$ define

$[H]_k^{m,0} := (\mathrm{dom}(\alpha_m) = D) \wedge \bigwedge_{j=m+1}^{k-1}(\mathrm{dom}(\alpha_j) = H)$.

If $i > 0$ define

$[H]_k^{m,i} := (\mathrm{dom}(\alpha_m) = D) \wedge (m < l_1 < k) \wedge (l_{i+1} = k) \wedge \bigwedge_{j=1}^{i}(l_j < l_{j+1}) \wedge \bigwedge_{j=1}^{i}(\mathrm{dom}(\alpha_{l_j})) = L) \wedge \bigwedge_{m<j\leqslant k-1}^{j\notin\{l_1,\ldots,l_i\}}(\mathrm{dom}(\alpha_j) = H)$.

If $m = -1$, then the truth of $(\mathrm{dom}(\alpha_m) = D)$ is set to True. Let $[M]_L^{m,i}$ encode the dynamic behavior produced by implementing the action sequence $\mathrm{ipurge}(\alpha_0 \cdots \alpha_{k-1}, L)$.

If $i = 0$ define

$[M]_L^{m,0} := I(s'_0) \wedge \bigwedge_{h=0}^{m} R(s'_h, \alpha_h, s'_{h+1})$.

If $i > 0$ define

$[M]_L^{m,i} := I(s'_0) \wedge \bigwedge_{h=0}^{m} R(s'_h, \alpha_h, s'_{h+1}) \wedge (s'_{l_1} = s'_{m+1}) \wedge \bigwedge_{j=1}^{i} R(s'_{l_j}, \alpha_{l_j}, s'_{l_{j+1}})$.

By combining all above components, we obtain the encoding of the counterexample of length k as follows:

Definition 14. Let M be an NSLKS, and k be a positive integer. Define $[M,\mathrm{IGNI}]_k := \exists s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k\ \exists m((-1 \leqslant m < k-1) \wedge I(s_0) \wedge [M]_k \wedge (\forall s'_0, \ldots, s'_{m+1}([H]_k^{m,0} \wedge [M]_L^{m,0} \rightarrow O(s_k, L) \neq O(s'_{m+1}, L)) \vee \bigvee_{i=1}^{k-1} \forall s'_0, \ldots, s'_{m+1}, s'_{l_1}, \ldots, s'_{l_{i+1}}([H]_k^{m,i} \wedge [M]_L^{m,i} \rightarrow O(s_k, L) \neq O(s'_{l_{i+1}}, L))))$.

Let $|[M,\ \mathrm{IGNI}]_k|$ denote the size of the formula $[M,\ \mathrm{IGNI}]_k$. Clearly, $|[M,\ \mathrm{IGNI}]_k|$ is polynomial on the length k.

Theorem 3. Let M be an NSLKS, and k be a positive integer. $[M,\ \mathrm{IGNI}]_k$ is satisfiable if and only if there exists a counterexample of length k in M.

Proof. Assume that $\sigma = \alpha_0 \cdots \alpha_{k-1}$ is the counterexample of length k in M, m is the position of the last action of security level D in α. Without loss of generality assume $m \geqslant 0$. By the definition of the counterexample, there exists $s \in s_{\mathrm{in}} \bullet \sigma$ such that for any $s' \in s_{\mathrm{in}} \bullet$ $\mathrm{ipurge}(\sigma, L)$, $O(s, L) \neq O(s', L)$. Let $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ represent a path produced by implementing σ such that $s_0 = s_{\mathrm{in}}, s = s_k$. Then the truth of $(-1 \leqslant m < k-1) \wedge I(s_0) \wedge [M]_k$ is True. Assume there exist i actions of security level L in $\alpha_{m+1} \cdots \alpha_{k-1}$, and that $\alpha_{l_1} \cdots \alpha_{l_i} (m+1 \leqslant l_1 < \cdots < l_i \leqslant k-1)$ is the distribution of the actions. Then by definition of $[H]_k^{m,i}$ and $[M]_L^{m,i}$, $\alpha_{l_1} \cdots \alpha_{l_i}$ makes $[H]_k^{m,i}$ true, and $\alpha_0 \cdots \alpha_m \alpha_{l_1} \cdots \alpha_{l_i}$ makes $[M]_L^{m,i}$ true. Since $s_{l_{i+1}} \in s_{\mathrm{in}} \bullet$ $\mathrm{ipurge}(\sigma, L)$, $O(s_k, L) \neq O(s'_{l_{i+1}}, L)$ is true. Hence, by the semantics of QBF, $[M,\ \mathrm{IGNI}]_k$ is satisfiable.

Assume that $s_0, \alpha_0, \ldots, \alpha_{k-1}, s_k$ is a sequence making $[M,\ \mathrm{IGNI}]_k$ satisfiable. Since $[M,\ \mathrm{IGNI}]_k$ is satisfiable, $(-1 \leqslant m < k-1) \wedge I(s_0) \wedge [M]_k$ is true, and $s_0, \alpha_0 \cdots \alpha_{k-1}, s_k$ is a path starting from the initial state. Without loss of generality assume $m \geqslant 0$, that there exist i actions of security level L in $\alpha_{m+1} \cdots \alpha_{k-1}$, with distribution of actions $\alpha_{l_1} \ldots \alpha_{l_i} (m+1 \leqslant l_1 < \ldots < l_i \leqslant k-1)$. Since $\forall s'_0, \ldots, s'_{m+1}, s'_{l_1}, \ldots, s'_{l_{i+1}}([H]_k^{m,i} \wedge [M]_L^{m,i} \rightarrow O(s_k, L) \neq O(s'_{l_{i+1}}, L))$ is satisfiable, for any $s' \in s_{\mathrm{in}} \bullet$ $\mathrm{ipurge}(\sigma, L)$, $O(s, L) \neq O(s', L)$. By the definition of counterexample, we have that $\alpha_0 \cdots \alpha_{k-1}$ is a counterexample of length k.

Theorem 3 shows that we can search for counterexamples with the help of QBF solvers. Hence, in Algorithm 1 we can use a QBF solver instead of directly searching for counterexamples. Now we consider how the computation of the recurrence diameter may be reduced to QBF. The following encoding follows directly from the definition of the recurrence diameter.

Definition 15. $\mathrm{loopfree}(s_0^{D^2}, \alpha_0, \ldots, \alpha_{k-1}, s_k^{D^2}) := \bigwedge_{i=0}^{k-1} R^2(s_i^{D^2}, \alpha_i, s_{i+1}^{D^2}) \wedge \bigwedge_{0\leqslant i<j\leqslant k}(s_i^{D^2} \neq s_j^{D^2})$.

For the integer k, if $\mathrm{loopfree}(s_0^{D^2}, \alpha_0, \ldots, \alpha_{k-1}, s_k^{D^2})$ is satisfiable, then $k \leqslant \mathrm{rd}(M^{D^2})$. Below is the QBF-based verification procedure of IGNI:

Algorithm 2 QBF-based verification of IGNI.

{

While $\mathrm{loopfree}(s_0^{D^2}, \alpha_0, \ldots, \alpha_{k-1}, s_k^{D^2})$ is satisfiable do

{

if $[M,\ \mathrm{IGNI}]_k$ or $[M,\ \mathrm{IGNI}]_{2k}$ is satisfiable, then return "False"

else, let $k = k + 1$

}

return "True"
}

4.4 A case study

We now present the construction of the QBF $[M_1,\ \mathrm{IGNI}]_k$ from the example M_1 shown in Figure 3. In subsection 4.3 we discussed how to construct$[M_1,\ \mathrm{IGNI}]_k$. We proceed to construct $[M_1,\ \mathrm{IGNI}]_2$ in the same way.

Let m be the position of the last action of security level D in α. First define $[M_1]_2 = \bigwedge_{i=0}^{1} R(s_i, \alpha_i, s_{i+1})$ to restrict $s_0, \alpha_0, s_1, \alpha_1, s_2$ to be a path in M_1. Now we consider the computation of ipurge$(\alpha_0\alpha_1, L)$. There exist several cases:

(1) $m = -1$, i.e. there are no actions of security level D. Depending on number of actions of security level L, two cases exist:

• $i = 0$, i.e. there are no actions of security level L. Then $[H]_2^{-1,0} = \bigwedge_{j=0}^{1}(\neg\alpha_j(1) \wedge \alpha_j(2))$;

• $i = 1$, i.e. there is one action of security level L. Then $[H]_2^{-1,1} = (-1 < l_1 < 2) \wedge (l_2 = 2) \wedge (\neg\alpha_{l_1}(1) \wedge \neg\alpha_{l_1}(2) \wedge \bigwedge_{\substack{j\notin\{l_1\} \\ 0\leqslant j\leqslant 1}} (\neg\alpha_j(1) \wedge \alpha_j(2))$.

(2) $m = 0$, i.e. α_0 is the action of security level D. Depending on number of actions of security level L, again two cases exist:

• $i = 0$, i.e. there are no actions of security L. Then $[H]_2^{0,0} = \alpha_0(1) \wedge \alpha_0(2) \wedge \neg\alpha_1[1] \wedge \alpha_1(2)$;

• $i = 1$, i.e. there is one action of security level L. Then ipurge$(\alpha_0 \cdots \alpha_{k-1}, L) = \alpha_0 \cdots \alpha_{k-1}$, i.e $\alpha_0 \cdots \alpha_{k-1}$ is not a counterexample.

(3) $m = 1$, i.e. α_1 is the action of security level D. Then ipurge$(\alpha_0 \cdots \alpha_{k-1}, L) = \alpha_0 \cdots \alpha_{k-1}$, i.e. $\alpha_0 \cdots \alpha_{k-1}$ is not a counterexample.

Now we let $[M]_L^{m,i}$ encode the dynamic behavior produced by implementing the action sequence ipurge$(\alpha_0 \cdots \alpha_{k-1}, L)$. There exist two cases:

(1) $m = -1$, depending on number of actions of security L, there exist two sub-cases:

• $i = 0$, i.e. there are no actions of security level L. Then $[M]_L^{-1,0} := I(s'_0)$;

• $i = 1$, i.e. there is one action of security level L. Then $[M]_L^{-1,1} := (s_0 = s'_{l_1}) \wedge R(s'_{l_1}, \alpha_{l_1}, s'_{l_2})$.

(2) $m = 0$, i.e. α_0 is the action of security level D. According to the number of actions of security L, there again exist two sub-cases:

• $i = 0$, i.e. there are no actions of security level L. Then $[M]_L^{0,0} := I(s'_0) \wedge R(s'_0, \alpha_0, s'_1)$;

• $i = 1$, i.e. there is one action of security level L. Since ipurge$(\alpha_0 \cdots \alpha_{k-1}, L) = \alpha_0 \cdots \alpha_{k-1}$, we do not compute the encoding.

The final QBF is $[M,\ \mathrm{IGNI}]_2 = \exists s_0, \alpha_0, s_1, \alpha_1, s_2 \exists m((-1 \leqslant m < 1) \wedge I(s_0) \wedge [M]_2 \wedge (\forall s'_0, \ldots, s'_{m+1}([H]_2^{m,0} \wedge [M]_L^{m,0} \to O(s_k, L) \neq O(s'_{m+1}, L)) \vee \forall s'_0, \ldots, s'_{m+1}, s'_{l_1}, s'_{l_2}([H]_k^{m,1} \wedge [M]_L^{m,1} \to O(s_k, L) \neq O(s'_{l_2}, L))))$. With the help of the QBF solver Quantor [14], $[M,\ \mathrm{IGNI}]_2$ is found to be not satisfiable. Hence there are no counterexamples of length 2 in M_1.

5 Combining window induction

In Algorithms 1 and 2, provided the system M satisfies IGNI, the verification procedure must repeat search for counterexamples an inefficient $2 \times \mathrm{rd}(M^2) + 1$ times In this section we explore, under the condition that M satisfies IGNI, how to terminate the procedure early with the help of window induction.

Let M be an NSLKS, M^D be the deterministic construction on M, and M^{D^2} be the double construction on M^D. By Lemma 2, M does not satisfy IGNI, if and only if there exist (s^D, s^D) and (r^D, t^D) such that (s^D, s^D) is D reachable from $(s_{\mathrm{in}}^D, s_{\mathrm{in}}^D)$, (r^D, t^D) is (H, L) reachable from (s^D, s^D), and if, in addition, there exists $s \in r^D$, such that for any $t \in t^D$,$O(s, L) \neq O(t, L)$. In other words, M satisfies IGNI, if and only if for any (s^D, s^D) and (r^D, t^D), if (s^D, s^D) is D reachable from $(s_{\mathrm{in}}^D, s_{\mathrm{in}}^D)$, and (r^D, t^D) is (H, L) reachable from (s^D, s^D), then for any $s \in r^D$, there exists $t \in t^D$ such that $O(s, L) = O(t, L)$. For convenience of expression, let $s^{D^2}[i]$ denote the ith element in s^{D^2}.

The classic induction proof consists of two sub-goals:

• For $s_0^{D^2}$, if $I(s_0^{D^2})$ also holds, then for each state s in $s_0^{D^2}[1]$, there exists $s' \in s_0^{D^2}[2]$ such that $O(s, L) = O(s', L)$. This sub-goal can be encoded with QBF as follows: $\forall s_0^{D^2}(I(s_0^{D^2}) \to \forall s \in s_0^{D^2}[1]\exists s' \in s_0^{D^2}[2](O(s, L) = o(s', L)))$;

• For the path $s_0^{D^2}, \alpha_0, s_1^{D^2}$, if for each state s in $s_0^{D^2}[1]$ there exists $s' \in s_0^{D^2}[2]$ such that $O(s, L) = O(s', L)$, then for each state s_1 in $s_1^{D^2}[1]$, there exists $s_1' \in s_1^{D^2}[2]$ such that $O(s_1, L) = O(s_1', L)$. This sub-goal can be encoded with QBF as follows: $\forall s_0^{D^2}\forall\alpha_0\forall s_1^{D^2}(R^{D^2}(s_0^{D^2}, \alpha_0) = s_1^{D^2} \wedge \forall s \in s_0^{D^2}[1]\exists s' \in s_0^{D^2}[2](O(s, L) = O(s', L))) \to \forall s_1 \in s_1^{D^2}[1]\exists s_1' \in s_1^{D^2}[2](O(s_1, L) = O(s_1', L)))$.

Consider a system M_2 comprising two independent components A and B, where A satisfies IGNI and B does not satisfy IGNI. The initial state of A is the initial state of M_2, and there exists a counterexample of length 1 in B. Since the initial state of A is also the initial state of M_2, M_2 also satisfies IGNI. However the classic induction proof becomes ineffective since it attempts to prove that B satisfies IGNI. The scope of the classic induction proof is therefore restricted.

Window induction, currently widely used in the verification of hardware, improves the classic induction proof by changing the induction depth. Formally for IGNI, window induction of length k consists of two sub-goals:

• For the path $s_0^{D^2}, \alpha_0, \ldots, \alpha_{k-1}, s_k^{D^2}$, if $I(s_0^{D^2})$ also holds, then for each state s in $s_i^{D^2}[1](0 \leqslant i \leqslant k)$, there exists $s' \in s_i^{D^2}[2]$ such that $O(s, L) = O(s', L)$;

• For the path $s_0^{D^2}, \alpha_0, \ldots, \alpha_k, s_{k+1}^{D^2}$, if for each state s_i in $s_i^{D^2}[1](0 \leqslant i \leqslant k)$ there exists $s_i' \in s_i^{D^2}[2]$ such that $O(s_i, L) = O(s_i', L)$, then for each state s_{k+1} in $s_{k+1}^{D^2}[1]$ there exists $s_{k+1}' \in s_{k+1}^{D^2}[2]$ such that $O(s_{k+1}, L) = O(s_{k+1}', L)$.

The first sub-goal can be achieved by checking the satisfiability of $[M,\ \text{IGNI}]_k$: if $[M,\ \text{IGNI}]_k$ is satisfiable then the first sub-goal holds. The second sub-goal can be achieved by checking the satisfiability of the following QBF $[M,\ \text{IGNI}]_k^{\text{in}}$: if $[M,\ \text{IGNI}]_k^{\text{in}}$ is satisfiable then the second sub-goal holds.

By recalling the definition of $[M]_k$, we have $[M^{D^2}]_k = \bigwedge_{i=0}^{k-1} R^2(s_i^{D^2}, \alpha_i, s_{i+1}^{D^2})$. We then define $[M,\ \text{IGNI}]_k^{\text{in}} := \forall s_0^{D^2}\alpha_0 \cdots \alpha_{k-1}s_k^{D^2}\alpha_k s_{k+1}^{D^2}(([M^{D^2}]_{k+1} \wedge \bigwedge_{i=0}^{k}(\forall s_i \in s_i^{D^2}[1]\exists s_i' \in s_i^{D^2}[2](O(s_i, L) = O(s_i', L)))) \to \forall s_{k+1} \in s_{k+1}^{D^2}[1]\exists s_{k+1}' \in s_{k+1}^{D^2}[2](O(s_{k+1}, L) = O(s_{k+1}', L)))$.

Algorithm 3 Verify IGNI based on the search for counterexamples and window induction.

```
{
k = 1
While loopfree(s_0^{D^2}, α_0, ..., α_{k-1}, s_k^{D^2}) is satisfiable do
      {
      if [M, IGNI]_k or [M, IGNI]_{2k} is satisfiable, return "False"
      if [M, IGNI]_k^in or [M, IGNI]_{2k}^in is satisfiable, return "True"
      let k = k + 1
      }
return "True"
}
```

6 Experimental results

In this section we analyze how the symbolic verification procedure depends on the number of actions in counterexamples and the size of models. We also compare Algorithm 1 and Algorithm 2 in terms of computer run-time.

We use the dining philosophers' problem as our benchmark. Philosophers are divided into three sets: users of security level H, users of security level D, and users of security level L. The security policy $\gg$ is defined as $\gg = \{(H, H), (H, D), (D, D), (D, L), (D, H), (L, L),\ (L, D), (L, H)\}$. Assign to a randomly-selected philosopher the security level H, to a second randomly-selected philosopher the security level L, and assign security level L to other philosophers. The philosopher of security level H and the philosopher of security level L cannot sit adjacent. For such an assigning the benchmark satisfies IGNI.

Table 1 Experimental results

Problem	S	E	$k=3$			$k=5$		
			T_1	V	T_2	T_1	V	T_2
DP(4)	7	8	0.32	69	0.65	0.95	102	1.13
DP(6)	18	12	1.11	100	1.78	7.14	160	5.06
DP(8)	47	16	4.94	121	3.46	19.73	197	12.99
DP(10)	123	20	11.57	159	7.01	24.03	253	16.20

We have performed our experiments on a PC equipped with a Pentium 3.06 GHz processor, 512 M RAM, running under the virtual Windows XP operating system Red Hat 9.0. The QBF solver used is Quantor 3.0. The experimental results are shown in Table 1. In Table 1, Problem is the benchmark, S is number of reachable states, E is the number of actions, k is the length of counterexamples, T_1(in seconds) is the time consumed by Algorithm 1, V is the number of Boolean variables, T_2(in seconds) is the time consumed by Algorithm 2. Since the set of benchmarks is small, our conclusions are not comprehensive. However, from the experimental results we directly observe that

1. The verification procedure based on the search for counterexamples can find counterexamples quickly, but as the search depth increases the time consumed by both algorithms increases at a greater than linear rate. Hence the verification procedure is efficient for systems with short counterexamples.
2. With increasing of system size, Algorithm 2 requires less time to complete than Algorithm 1.

7 Application

SEC_VISTA is a B1-level security database management system developed by our research group. The system uses the Bell-LaPadula multi-level security model, and has realized some important security features such as the autonomy and mandatory access control, authentication, events auditing, and backup and recovery of data. Since in SEC_VISTA two access control mechanisms are utilized, each operation of every user is legal. However the user of the higher security level can transfer information to the user of the lower security level by a series of legal operations, i.e. covert information flow occurs from the user of the higher security level to the user of the lower security level. To test the practicality and effectiveness of our algorithmic verification procedure, we use the algorithm to search for covert illegal information flow occurring in the file management component of SEC_VISTA.

7.1 Operations in the file management component

In the database management system, the function managing files can perform several operations on the files. When the user submits a request related to the file, the security mechanism first checks the permission of the operation by the autonomy and mandatory access controls in the data dictionary, then calls the function managing files to implement these operations. All file management functions are shown in Table 2.

7.2 Build the NSLKS for the file management component

In this subsection we build an NSLKS for the file management component.

7.2.1 *The set of states and the initial state*

Every file has many attributes. Since only the shared attributes can be used to transfer the covert information, we need only consider four shared attributes: "name", "filelocked", "filelocker", "len", here "name" is the name of the file, "filelocked" is flag of the file being locked, "filelocker" is the set of users which locked the file, "len" is the length of the file. The state is defined as a valuation for the four attributes. All valuations constitute the set of states. Since the range of each attribute is finite, the set of states is finite. Let s_{in} be the initial state, where name=$\emptyset$, filelocked=0, filelocker=$\emptyset$, len=0.

Table 2 Functions for managing files

Function Name	Features
int file_create (name, len)	create the file: the caller provides two parameters: the file name, and the record length; if the file is created successfully its security level is the security level of the caller; return 1, otherwise return 0.
int file_open (fp, name)	open the file: the caller provides the parameter: the file name; call the function file_lock to lock the file; if the file is locked successfully then open the file, assign fp the number of the file , and assign to the security level of the caller the security level of the file, otherwise return 0.
int file_lock(fp)	lock the file: if the file is locked successfully then increase filelocked by 1, place the identifier of the caller into the array filelocker and return 1, otherwise return 0.
int file_unlock (fp)	unlock the file: if the file is unlocked successfully then decrease filelocked by 1, delete the identifier of the caller from the array filelocker, and return 1, otherwise return 0.
int file_close (fp)	close the file: the caller provides the file number fp; call the function file_unlock to unlock the file; if filelocker is a empty set, then close the file and return 1, otherwise return 0.
int file_delete (name)	delete the file: the caller provides the name of the file; if the filelocker is an empty set return 1, otherwise return 0.
int get_record (fp, rcdno, bf)	read the record from the file and put the record into the buffer: the caller provides the number of the file fp, the record number RPTR rcdno, and the point of the buffer *bf; if read successfully return 1, otherwise return 0.
int put_record (fp, rcdno, bf)	put the content in the buffer into the file: the caller provides the file number fp, the record number RPTR rcdno, and the point of the buffer *bf; if successful return 1, otherwise return 0.
int delete_record (fp , rcdno)	delete the record from the file: the caller provides the file number fp, the record number RPTR rcdno; if file deleted successfully return 1, otherwise return 0.

Table 3 The state transition relation created by implementing primitive operations

Primitive Operation	Function
file_create	"name" is assigned the name of the created file; "len" is assigned the length of the file; other attributes are not changed
file_open	increase "filelocked" by 1; place the identifier of the caller into "filelocker"; other attributes are not changed
file_lock	increase "filelocked" by 1; place the identifier of the caller into "filelocker"; other attributes are not changed
file_unlock	decrease "filelocked" by 1; delete the identifier of the caller from "filelocker"; other attributes are not changed
file_close	decrease "filelocked" by 1; delete the identifier of the caller from "filelocker"; other attributes are not changed
file_delete	"name" equals a empty set; "len" equals 0; "filelocked"equals 0; "filecker"equals $\emptyset$
get_record	attributes are not changed
put_record	attributes are not changed
delete_record	attributes are not changed

7.2.2 *The set of actions and the set of security levels*

From the functions listed in Table 2, we encode the following primitive operations: file_create, file_open, file_lock, file_unlock, file_close, file_delete, get_record, put_record, delete_record. We define these primitive operations as a set of actions. The security level of every action is decided by the caller. Each operation is assigned an index to show its security level, i.e. the security level of the caller. For example file_creat$_L$ represents the security level L. of the user calling file_create.

7.2.3 *The state transition relation*

The state transition relation created by implementing primitive operations is defined in Table 3.

7.2.4 *The observation function*

We define the set of atomic propositions AP to be the characterization of the shared attributes. For example, define the proposition p as "file_lock equals 1". Since "name", "filelocked", "filelocker", "len" are shared attributes, they are visible to users at higher and lower security levels.

7.3 Search for illegal information flow

For the above built NSLKS, we use Algorithm 3 to identify illegal information flow. For example, the higher-security user can perform two operations, put_record and delete_record, on the database system such that "filelocked" is increased or decreased by 1. The lower-security user can observe changes in the value of "filelocked" by performing the same two operations on the database system, enabling information to be translated in the agreed manner. We can identify one illegal information flow only by implementing Algorithm 3 a single time. We therefore impose a constraint on the path such that the identified legal information flow is not searched again.

To evaluate the practicality and effectiveness of the algorithmic verification procedure, we use the share resource method to identify illegal information flow in the file management component. Many potential illegal information flows are identified, some of which are false illegal information flows. Our algorithmic verification procedure avoids the search for false legal information flow. A second advantage of our procedure over the shared resource method is that the counterexample provides the scene producing the legal information flow.

8 Conclusions

In this paper we first extended generalized noninterference to IGNI such that intransitive information flow security policies such as the "channel-control" can be formulated. Based on the search for counterexamples and induction proof, we then proposed a sound and complete method for verifying IGNI. Furthermore, we reduced the search for counterexamples and induction proof to the QBF problem to enable use of a QBF solver in performing these tasks. This reduction improves the efficiency of the method, and decreases the space requirement. Finally, we illustrated use of the method in identifying illegal information flow, by applying the method to a B1-level security database management system developed by our research group.

Identification of the covert channel is a key problem in high security level systems. At present there are only manual methods by which to identify covert channels. Our method can be used to identify covert channels automatically by the following procedure: First, use the predicate abstraction [20] to obtain the finite state model from the system specification, then apply our method to check whether IGNI holds. If IGNI is not satisfied then covert channels exist.

Current QBF decision procedures are based on general purposes. In future extensions of the work presented here, we aim to develop an automatic tool to search for covert channels, and to design a more efficient QBF decision procedure by introducing the features of counterexamples.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant Nos. 60773049, 61003288), PhD Programs Foundation of Ministry of Education of China (Grant No. 20093227110005), Natural Science Foundation of Jiangsu Province (Grant No. BK2010192), People with Ability Foundation of Jiangsu University (Grant No. 07JDG014), Fundamental Research Project of the Natural Science in Colleges of Jiangsu Province (Grant No. 08KJD520015).

References

1 Shen C X, Zhang H G, Wang H M, et al. Trusted computing research and development (in Chinese). Sci China Inf Sci, 2010, 40: 139–166
2 Zhang H G, Yan F, Fu J M, et al. Theory and key technology evaluation of trusted computing platform (in Chinese). Sci China Inf Sci, 2010, 40: 167–188
3 Focardi R, Gorrieri R. The compositional security checker: a tool for the verification of information flow security properties. IEEE Trans Softw Eng, 1997, 27: 550–571
4 Qing S H, Shen C X. Design on high secure operating system (in Chinese). Sci China Ser E-Tech Sci, 2007, 37: 238–253
5 Liu W Q, Han N P, Chen Z. Identifying and dealing with covert channel of the secure OS-SLinux (in Chinese). ACTA Electron Sin, 2007, 35: 153–156
6 Qing S H. Covert channel analysis in secure operating systems with high security levels. J Softw, 2004, 15: 1837–1849
7 Goguen J A, Meseguer J. Security policies and security models. In: Peter G N, Robert M, eds. IEEE Symposium On Security and Privacy. Washington: IEEE Computer Society Press, 1982. 11–20
8 Daryl M. Specifications for multi-level security and a hook-up property. In: David B, Steve L, eds. IEEE Symposium On Security and Privacy. Washington: IEEE Computer Society Press, 1987. 161–166
9 Rushby J M. Noninterference, Transitivity, and Channel Control Security Policies. Technical Report CSL-92-02, SRI International, 1992
10 Goguen J A, Meseguer J. Unwinding and inference control. In: Dorothy E D, Jonathan K M, eds. IEEE Symposium On Security and Privacy. Washington: IEEE Computer Society Press, 1984. 75–86
11 Deepak D, Raghavendra K R, Barbara S. An automata based approach for verifying information flow properties. Elec Note Theor Comput Sci, 2005, 135: 39–58
12 Ron M, Zhang C Y. Algorithmic verification of noninterference properties. Elec Note Theor Comput Sci, 2007, 168: 61–75
13 Zi X C, Yao L H, Li L. A state-based approach to information flow analysis (in Chinese). Chinese J Comput, 2006, 29: 1460–1467
14 Een N, Biere A. Effective preprocessing in SAT through variable and clause elimination. LNCS, 2005, 3569: 61–75
15 Pan G, Vardi M Y. Symbolic decision procedures for QBF. LNCS, 2004, 3258: 453–567
16 Biere A. Resolve and expand. LNCS, 2005, 3542: 59–70
17 Ian P G, Holger H H, Andrew G D R, et al. Using stochastic local search to solve quantified Boolean formulae. LNCS, 2003, 2833: 348–362
18 Tao Z H, Zhou C H, Chen Z, et al. Bounded model checking of CTL*. J Comput Sci Technol, 2007, 22: 39–43
19 Zhou C H, Chen Z Y, Tao Z H. QBF based symbolic model checking for knowledge and time. LNCS, 2007, 4484: 386–397
20 Qu W X, Li T, Guo Y, et al. Advances in predicate abstraction (in Chinese). J Softw, 2008, 19: 27–38