Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070521

Algorithmic Verification of Intransitive Noninterference for 3-domain Security Policies with a SAT Solver

Liu Zhifeng[1,2,*], *Zhou Conghua* [1], *Ge Yun* [2] *and Zhang Dong* [2]

[1] School of Computer Science and Telecommunication Engineering,Jiangsu University, China
[2] School of Electronic Science and Engineering, Nanjing University, China

Received: 14 Jan. 2013, Revised: 17 May. 2013, Accepted: 18 May. 2013
Published online: 1 Sep. 2013

Abstract: In this paper we propose an automated verification approach to checking intransitive noninterference for deterministic finite state systems. Our approach is based on the counterexamples search verification strategy, and is conducted in gradual manner. It produces counterexamples of minimal length. Further, we reduce the counterexamples search to propositional satisfiability. For the case that there are no counterexamples, we also introduce the window induction proof method in order to avoid considering unnecessary iterations, and show that the induction proof can be performed by the boolean decision procedure. In addition, based on graph-theoretic properties of systems we propose an over-approximation to the length of the smallest counterexample, and the over-approximation can also be checked by the boolean decision procedure.

Keywords: noninterference, intransitive noninterference,propositional satisfiability, window induction

1 Introduction

One of the typical problems in computer security is that confidential data needs protecting from undesired accesses. A well known approach to solving this problem is the Multilevel Security, which is a policy for managing objects at various levels of secrecy. In multilevel security systems every object and every user is bound to a secrecy level and the information flow [1] can be directed only from low users to high users. The system achieves this aim by implementing access control policies. As remarked in [2] 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. Information leakage over covert channels [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 and indirect information flow. Noninterference [5–7] captures any causal dependency from a high-level action h to a lower-level action l. By causal dependency we mean that the dependent action can not occur without the occurrence of the preceding action. Such a dependency creates an insecure channel called a covert channel, having the capability of transmitting high-level information concerning h to any low-level agent observing l. In practice, however, many practical security problems go beyond the scope of simple noninterference. In particular, the problem of confidentiality in multilevel security systems, where the relation over the set of security levels capturing allowed information flows, are not transitive. Noninterference cannot cope with intransitive flow policies. Therefore, intransitive noninterference has been proposed by Rushby in the literature [8].

Since Rushby's initial work, many researchers did much work about intransitive noninterference. For example, more definitions about intransitive noninterference have been proposed in the literature [9–11]. For nondeterministic systems, Mantel presented a new model in terms of event system for intransitive flow [12]. However, there is few work on the verification of intransitive noninterference because intransitive noninterference is a global requirement whose verification is usually a complex task. At present, the absence of feasible algorithmic verification approaches is the main problem in checking intransitive noninterference. Prior to our work, only one rigorous

[*] Corresponding author e-mail: liuzf@ujs.edu.cn

algorithm has been proposed by Nejib Ben Hadj-Alouane in the literature [13, 14] to check intransitive noninterference based on some necessary and sufficient conditions. Other algorithms, for example[8], only use a sufficient condition as a basis for checking intransitive noninterference.

In this paper, we focus on using a SAT solver to verify intransitive noninterference on deterministic systems. Our work is motivated by which the verification methods based on SAT solvers [15–18] 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 [19–22]. The successful application of SAT solvers in formal verification is due to dramatic improvements in SAT solver technology over the past decade. At present, several powerful SAT solvers can handle propositional formulas with hundreds of thousands of variables.

We present a symbolic algorithmic approach to the verification of intransitive noninterference. The basic concept of our algorithmic approach consists of two aspects: one aspect is to search for a counterexample of intransitive noninterference in executions whose length is bounded by some integer k, the search works by mapping the problem of the existence of some counterexample of length k to the propositional satisfiability problem; another aspect is to use the window induction technique [23] to verify intransitive noninterference, and the induction hypothesis is checked by a SAT solver. Our algorithmic approach shown in Figure 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 intransitive noninterference.
2. Search for Counterexamples by a SAT Solver: Reduce the existence problem of counterexamples of some length to the propositional satisfiability problem, i.e., there exists counterexamples of length k if and only if the propositional formula $[M,INI]_k$ is satisfiable. If $[M,\ INI\ ]_k$ is satisfiable, then claim that the system does not satisfy intransitive noninterference, and return a counterexample.
3. Inductive Proof by a SAT Solver: Reduce the window induction proof to the propositional satisfiability problem, i.e., the window induction proof of the size of window k succeeds if and only if $[M,INI]_k^{IN}$ is a tautology. If $[M,INI]_k^{IN}$ is a tautology, then claim that the system satisfies intransitive noninterference, else let $k = k+1$, return to Step 1.

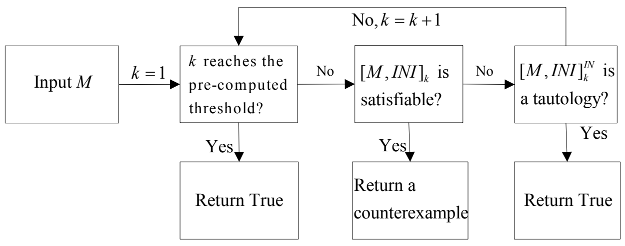
Input M
$k=1$
k reaches the pre-computed threshold?
No
$[M,INI]_k$ is satisfiable?
No
$[M,INI]_k^{IN}$ is a tautology?
No, $k=k+1$
Yes
Return True
Yes
Return a counterexample
Yes
Return True

Fig. 1 A Framework for Verifying Intransitive Noninterference Using a SAT Solver

2 Related Work

Intransitive noninterference is a global requirement. The direct verification of noninterference is a complex task. In [8], Rushby proposed an unwinding approach which reduces the global requirement to more local conditions that involves only individual transitions. The unwinding approach is usually called the traditional verification approach. Here, we first recall the unwinding approach, then compare it with our approach.

The basic concept of the unwinding approach is to reduce checking intransitive noninterference for a deterministic system to determine the existence of some equivalence relation called the unwinding relation. The relationship between the unwinding relation and intransitive noninterference has the following classical results: A deterministic system satisfies intransitive noninterference, if there exists an unwinding relation for the system. Henceforth, how to determine the existence of the unwinding relation is a key problem in verifying intransitive noninterference. However, we note that the existence of the unwinding relation is only a sufficient condition, not a necessary condition. And to the best of our knowledge, there is no efficient algorithmic approach to checking whether there exists the unwinding relation. In addition, the unwinding approach also suffers from the state explosion problem, i.e. the model size is the major factor which affects the performance of decision procedures.

Recently, in [13, 14] for discrete event systems Nejib Ben Hadj-Alouane et al. proposed an algorithmic approach to the verification of intransitive noninterference. The basic concept of their approach is to use an observability based on a purge function, called *iP*-observability, to capture intransitive noninterference, and reduce the checking of *iP*-observability to the checking of *P*-observability, which can be done efficiently.

We now consider the main differences between above approaches and ours. First, we develop an algorithmic approach to checking intransitive noninterference, and can use boolean decision procedures to perform the algorithm. Boolean decision procedures can make us verify large systems. Second, our approach combines the counterexample search strategy and induction proof technique. The counterexample search strategy makes us find counterexamples quickly. And the counterexample can guide designers to modify the design of systems. Third, compared with the unwinding approach, our approach is complete. That is our approach can justify whether or not a system satisfies intransitive noninterference.

The paper is organized as follows. In Section 2, we describe intransitive noninterference in detail. In Section 3, we present our SAT-based automated verification technique to check intransitive noninterference. In Section 4, our experimental results are presented. Some conclusions and ideas for future research are presented in Section 5.

3 Intransitive Noninterference

The definition of noninterference is based on transitive flow policies, i.e. $H \rightsquigarrow D \wedge D \rightsquigarrow L \Rightarrow H \rightsquigarrow L$. Goguen and Meseguer recognized the inability of transitive flow policies to model some security policies such as control policies, and they introduced several extensions to the basic formulation in their paper [24]. However, the first satisfactory formal account of intransitive information flow has been introduced by Rushby [4], followed by definitions by Pinsky [9] and by Roscoe and Goldsmith [10]. Rushby gave a formalization of intransitive noninterference in terms of input-output automata. Intuitively intransitive noninterference says that flows from the high domain H to a trusted domain D and flows from D to the low domain L are admissible while a direct flow from H to L is not allowed. In this section we recall the definition of intransitive noninterference introduced by Rushby. We use a type of state transition graph called Security Labeled Kripke Structure(SLKS) to describe the behavior of a security system.

Definition 3.1. A Security Labeled Kripke Structure (SLKS) M is composed of

–S : a set of states with an initial state $s_{in} \in S$.
–Σ : a set of actions.
–$R : S \times \Sigma \rightarrow S$ a transition function.
–$\mathbb{D}$: a set of security domains.
–$dom : \Sigma \rightarrow \mathbb{D}$ a function associating a security domain with each action.
–$O : S \times \mathbb{D} \rightarrow 2^{AP}$ an observation function, where AP is a set of atomic propositions.

A security policy is specified by a reflexive relation $\rightsquigarrow$ on $\mathbb{D}$. We use $\not\rightsquigarrow$ to denote the complement relation, that is $\not\rightsquigarrow = (\mathbb{D} \times \mathbb{D}) \setminus \rightsquigarrow$, where $\setminus$ denotes set difference. We call $\rightsquigarrow$ and $\not\rightsquigarrow$ as the interference and noninterference relations, respectively. A policy is said to be intransitive if its interference relation has no transitive property.

To formally define intransitive noninterference, a function *sources* on paths is introduced by Rushby. The purpose of the function *sources* is to identify those actions in a path that should not be deleted. For the action sequences $\alpha = \alpha_1 \cdots \alpha_n$, $\beta = \beta_1 \cdots \beta_m$, define $\alpha \circ \beta = \alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_m$.

Definition 3.2. We define the function $sources : \Sigma^* \times \mathbb{D} \rightarrow 2^{\mathbb{D}}$ as follows $sources(\varepsilon, u) = \{u\}$; if $\exists v : v \in sources(\alpha, u) \wedge dom(a) \rightsquigarrow v$ then $sources(a \circ \alpha, u) = sources(\alpha, u) \cup \{dom(a)\}$, otherwise $sources(a \circ \alpha, u) = sources(\alpha, u)$.

In essence $v \in sources(\alpha, u)$ means either that $v = u$ or that there is a subsequence of α consisting of actions performed by domains $w_1, w_2, \cdots, w_n$ such that $w_1 \rightsquigarrow w_2 \rightsquigarrow \cdots \rightsquigarrow w_n$, $v = w_1$, and $u = w_n$.

We can now define the function $ipurge : \Sigma^* \times D \rightarrow \Sigma^*$ as follows: $ipurge(\varepsilon, u) = \varepsilon$; if $dom(a) \in sources(a \circ \alpha, u)$, then $ipurge(a \circ \alpha, u) = a \circ ipurge(\alpha, u)$, otherwise $ipurge(a \circ \alpha, u) = ipurge(\alpha, u)$.

Informally, $ipurge(\alpha, u)$ consists of the subsequence of α with all those actions that should not be able to interfere with u removed. Thus, security is now defined in terms of the *ipurge* function.

Definition 3.3. We call a SLKS M satisfy intransitive noninterference, denoted by $M \models INI$, if for any $\alpha \in \Sigma^*, a \in \mathbb{D} : O(s_{in} \bullet \alpha, a) = O(s_{in} \bullet ipurge(\alpha, a), a)$.

Consider a three domain system, $\mathbb{D} = \{H, D, L\}$, where D is a downgrading domain. The non-interference relation is such that: $\rightsquigarrow = \{(H,D),(D,L),(D,H),(L,D),(L,H)\}$. We consider the systems M_1 and M_2 given in Figures 2 and 3 in which $dom(h) = H$, $dom(l) = L$, $dom(d) = D$. Each state s is labeled by the function $O(s, L)$. It is easy to justify the system M_1 shown in Figure 1 is secure for $\rightsquigarrow$, M_2 shown in Figure 2 is not secure for $\rightsquigarrow$. In M_2, consider the action sequence $hdhl$. $ipurge(hdhl, L) = hdl$. It is easy to compute $O(s_0 \bullet hdhl, L) = \{p_3\}, O(s_0 \bullet ipurge(hdhl, L), L) = \{p_1\}$. That is $O(s_0 \bullet hdhl, L) \neq O(s_0 \bullet ipurge(hdhl, L), L)$.

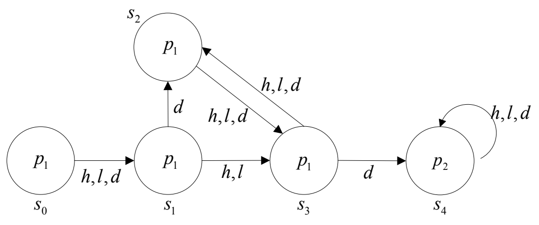

Fig. 2 Intransitive noninterference satisfied

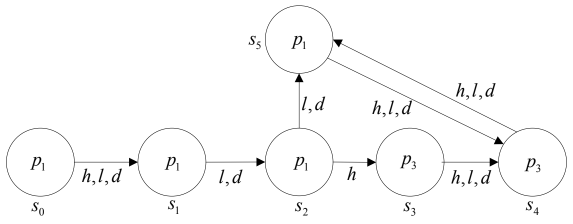

Fig. 3 Intransitive noninterference not satisfied

4 SAT-based Algorithmic Verification of Intransitive Noninterference

4.1 Symbolic Representation of System

In the remainder of this paper, we restrict our attention to systems with only three security domains $\mathscr{R} = \{H, L, D\}$, governed by the following noninterference relation: $\rightsquigarrow = \{(H,D),(D,L),(L,H),(L,D),(D,H)\}$. That is only $H \to L$ is not allowed. For discussion convenience, we first modify the definition of SLKS.

Definition 4.1. A SLKS M is a 9-tuple $(S, s_{in}, \Sigma_L, \Sigma_H, \Sigma_D, \Sigma, R, AP, O_L)$ where

- S is a finite non-empty set of states.
- $s_{in} \in S$ is an initial state.
- $\Sigma_L \subset \Sigma$ is a finite set of actions associated with the domain L.
- $\Sigma_H \subset \Sigma$ is a finite set of actions associated with the domain H.
- $\Sigma_D \subset \Sigma$ is a finite set of actions associated with the domain D.
- Σ is a finite set of actions with $\Sigma = \Sigma_L \cup \Sigma_H \cup \Sigma_D$.
- $R : S \times \Sigma \to S$ is a transition function.
- AP is a finite set of propositions.
- $O_L : S \to 2^{AP}$ is defined as $O_L(s) = O(s, L)$.

For simplicity, we define $ipurge_L(\sigma) = ipurge(\sigma, L)$, and a predicate $R(s,\sigma,s')$ iff $s' = R(s,\sigma)$. To represent the result of executing a sequence of actions, define the operation $\bullet : S \times \Sigma^* \to S$, by $s \bullet \varepsilon = s$; and $s \bullet (\alpha \circ a) = R(s \bullet \alpha, a)$. In a SLKS M, a path $\pi = s_0, \sigma_0, s_1, \sigma_1, \ldots, \sigma_k, s_{k+1}$ of M is an alternating sequence of states and actions subject to the following: for each $k \geq i \geq 0, s_i \in S, \sigma_i \in \Sigma$ and $R(s_i, \sigma_i, s_{i+1})$ holds.

We now describe how a SLKS can be represented symbolically. To represent this structure we must describe the set S, the set Σ, the transition relation R, and the labeling function O_L. Without loss of generality, we suppose that there are 2^m states for some $m > 0$, 2^n actions associated with the domain H for some $n > 0$, 2^n actions associated with the domain L, 2^n actions associated with the domain D, $AP = \{p_1, \cdots, p_k\}$ for some $k > 0$.

Let $\phi : S \leftrightarrow \{0,1\}^m$ be a bijection function that maps each state of S to a boolean vector of length m. The initial state s_{in} can be represented by a boolean vector $\phi(s_{in})$, denoted by $I(s_{in})$. $\psi : \Sigma \leftrightarrow \{0,1\}^{n+2}$ be a bijection function satisfying $\psi : \Sigma_L \leftrightarrow \{0\} \times \{1\} \times \{0,1\}^n$, $\psi : \Sigma_H \leftrightarrow \{1\} \times \{1\} \times \{0,1\}^n$, and $\psi : \Sigma_D \leftrightarrow \{0\} \times \{0\} \times \{0,1\}^n$. ψ maps each action of Σ to a boolean vector of length $n+2$. Let $\phi(s) = (b_1, \ldots, b_m)$. Then, the state s can be characterized by a boolean formula as follows: $\bigwedge_{b_i=1}^{1\leq i\leq m} b_i^{'} \wedge \bigwedge_{b_i=0}^{1\leq i\leq m} \neg b_i^{'}$, where $b_i^{'}$ is an atomic proposition. For simplicity, we use $\phi(s)$ instead of the above formula. In the same way an action σ can be characterized by the boolean formula $\psi(\sigma)$. The transition relation $R(s,\sigma,s')$ can be characterized by a boolean formula as follows: $\widehat{R}(\phi^{-1}(s), \psi^{-1}(\sigma), \phi^{-1}(s'))$. For simplicity, we use R instead of $\widehat{R}$. The labeling function $O_L(s)$ can be represented as follows: $\widehat{O_L(s)} = \phi^{-1}(s) \wedge \bigwedge_{p \in O_L(s)} p \wedge \bigwedge_{p \in AP \setminus O_L(s)} \neg p$. $O_L(s) \neq O_L(s')$ can be represented as follows: $\widehat{O_L(s)} \wedge \widehat{O_L(s')} \wedge \neg((\bigwedge_{p \in O_L(s)} p \wedge \bigwedge_{p \in Ap \setminus O_L(s)} \neg p) \leftrightarrow (\bigwedge_{p \in O_L(s')} p \wedge \bigwedge_{p \in Ap \setminus O_L(s')} \neg p))$.

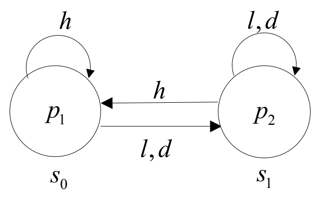

Fig. 4 A Security Labeled Kripke Structure with Two states

In order to illustrate how to represent a SLKS symbolically, we consider the two states structure shown in Figure 4. In this case, there are two states. We need one boolean variable v to encode states: $v = 0$ encoding s_1, $v = 1$ encoding s_2 . We introduce one additional boolean variable v' to encode successor states. There are three actions in this example. Thus we need a two bit boolean vector (u_1, u_2) to encode actions: $(1,1)$ encoding h, $(0,1)$ encoding l, and $(0,0)$ encoding d. Then we can represent the transition from state s_0 to state s_1 enabled by inputting l with $\neg v \wedge \neg u_1 \wedge u_2 \wedge v'$. The boolean formula for the entire transition relation is given by $(\neg v \wedge \neg u_1 \wedge u_2 \wedge v') \vee (\neg v \wedge u_1 \wedge u_2 \wedge \neg v') \vee (v \wedge \neg u_1 \wedge u_2 \wedge v') \vee (v \wedge u_1 \wedge u_2 \wedge \neg v') \vee (v \wedge u_1 \wedge u_2 \wedge \neg v') \vee (\neg v \wedge \neg u_1 \wedge \neg u_2 \wedge v') \vee (v \wedge \neg u_1 \wedge \neg u_2 \wedge v')$. The labeling function is represented by $(\neg v \wedge p_1 \wedge \neg p_2) \vee (v \wedge p_2 \wedge \neg p_1)$.

4.2 Checking Intransitive Noninterference by Searching for Counterexamples

Definition 4.2. (Counterexample for Intransitive Noninterference): Let M be a SLKS. A finite action sequence $\sigma \in \Sigma^*$ is called a counterexample of intransitive noninterference iff $O_L(s_{in} \bullet \sigma) \neq O_L(s_{in} \bullet ipurge_L(\sigma))$.

It is easy to justify that a SLKS M does not satisfy intransitive noninterference iff there is a counterexample. That is we can check intransitive noninterference if we consider all possible actions sequences. This leads to a straightforward intransitive noninterference checking procedure. To check whether $M \models INI$, the procedure checks all action sequences with length k for $k=0,1,2,\cdots$. If a counterexample with length k is found, then the procedure proves that $M \not\models INI$ 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 threshold on k, and have that for all k within the threshold, if there are no counterexamples of length k, we can conclude that $M \models INI$. We call this threshold the *Completeness Threshold*, and denote it by CT. It is clear, if $M \models INI$ then the smallest CT is equal to 0, and otherwise it is equal to the length of the shortest counterexample. This implies that finding the smallest CT is at least as hard as checking whether $M \models INI$. Consequently, we should like to concentrate on computing an over-approximation to the smallest CT based on graph-theoretic properties.

Definition 4.3. (Double Construction): Let $M = (S, s_{in}, \Sigma_L, \Sigma_H, \Sigma_D, \Sigma, R, AP, O_L)$ be a security system, define $M^2 = (S^2, s_{in}^2, \Sigma_L, \Sigma_H, \Sigma_D, \Sigma, R^2, AP, O_L^2)$ to be the system with identical actions,where

–$S^2 = S \times S$.
–$s_{in}^2 = (s_{in}, s_{in})$.
–$R^2 \subseteq S^2 \times \Sigma \times S^2$ is a transition relation given by: for $a \in \Sigma_L \cup \Sigma_D$, $((s_1,s_2),a,(s_3,s_4)) \in R^2$ if and only if $R(s_1,a,s_3) \wedge R(s_2,a,s_4)$; and for $a \in \Sigma_H$, $((s_1,s_2),a,(s_3,s_4)) \in R^2$ if and only if $R(s_1,a,s_3) \wedge (R(s_2,a,s_4) \vee (s_2 = s_4))$.
–$O_L^2(s,t) = (O_L(s), O_L(t))$.

Note that in every transition of M^2, for $a \in \Sigma_H$ and the right part of each state, we add an additional transition: from each state to itself. The key in checking intransitive noninterference is to compare $O_L(s_{in} \bullet \sigma)$ and $O_L(s_{in} \bullet ipurge_L(\sigma))$. We now consider the computation of $ipurge_L(\sigma)$.

Lemma 4.1. Let $\sigma = \alpha \circ d \circ \beta$, where $d \in \Sigma_D, \beta \in (\Sigma_L \cup \Sigma_D)^*$. Then $ipurge_L(\sigma) = \alpha \circ d \circ ipurge_L(\beta)$.

Proof. Let $\alpha = \alpha_1 \cdots \alpha_m$, $\beta = \beta_1 \cdots \beta_n$. We first compute $sources(\sigma)$ recursively. According the definition of the function $sources$, we have that if $dom(\beta_n) = L$, then $sources(\beta_n, L) = sources(\varepsilon, L) \cup dom(\beta_n) = \{L\}$, else if $dom(\beta_n) = H$, then $sources(\beta_n, L) = sources(\ \varepsilon,\ L) = \{L\}$. Thus $sources(\beta_n, L) = \{L\}$. In the same way, we have $sources(\beta_{n-1}\beta_n, L) = \{L\}, \cdots,\ sources\ (\ \beta\ ,L\ ) = \{L\}$.

For $d \circ \beta$, since $dom(d) \rightsquigarrow L$, $sources(d \circ \beta, L) = sources(\beta, L) \cup \{dom(d)\} = \{D, L\}$. For $\alpha_m \circ d \circ \beta$, since $H \rightsquigarrow D, L \rightsquigarrow D$, $sources(d \circ \beta, L) = sources(\beta, L) \cup \{dom(\alpha_m)\}$. Therefore, if $dom(\alpha_i) = H$ with $1 \le i \le m$, then $sources(\alpha_i \cdots \alpha_m \circ d \circ \beta) = \{H, L, D\}$. That is if there exists α_i with $1 \le i \le m$ such that $dom(\alpha_i) = H$, then $sources(\sigma) = \{H, L, D\}$,otherwise $sources(\sigma) = \{L, D\}$.

We now consider the computation of $ipurge_L(\sigma)$. From the above computation procedure of $sources$, we have that if $dom(\alpha_1) = H$ then $source(\sigma, L) = \{H, D, L\}$. That is $dom(\alpha_1) \in source(\sigma, L)$. If $dom(\alpha_1) = L$ then it is clear $L \in source(\sigma, L)$. Therefore $dom(\alpha_1) \in source(\sigma, L)$. According the definition of the function $ipurge_L$, $ipurge_L(\sigma) = \alpha_1 \circ ipurge_L(\alpha_2 \cdots \alpha_m \circ d \circ \beta)$. In the same way we have $ipurge_L(\sigma) = \alpha_1 \circ \alpha_2 \circ ipurge_L(\alpha_3 \cdots \alpha_m \circ d \circ \beta) = \cdots = \alpha \circ d \circ ipurge_L(\beta)$.

For $\beta \in (\Sigma_L \cup \Sigma_D)^*$, it is easy to check that $source(\beta, L) = \{L\}$. Henceforth, $ipurge_L(\beta)$ is the subsequence of β by deleting all actions associated with the domain H.

Definition 4.4. (D Reachable): Let M^2 be the double construction of a SLKS M. We call the state (s,s) is D reachable from (s_{in}, s_{in}) if and only if $s = s_{in}$ or there exists a finite action sequence $\sigma_0 \cdots \sigma_n$ such that $\sigma_n \in \Sigma_D$ and $((s_{in}, s_{in}), \sigma_0, (s_1, s_1)) \in R^2$, $((s_i, s_i), \sigma_i,\ (s_{i+1},\ s_{i+1})) \in R^2$ for all $1 \le i \le n$, and $s_{n+1} = s$.

Intuitively, if (s,s) is D reachable from (s_{in}, s_{in}), then there exists an action sequence σ ending with an action in Σ_D such that $s = s_{in} \bullet \sigma$.

Definition 4.5. ((H,L) Reachable): Let M^2 be the double construction of a SLKS M. We call the state (r,t) is (H,L) reachable from (s,s) if and only if there exists a finite action sequence $\sigma \in (\Sigma_L \cup \Sigma_H)^*$ such that $r = s \bullet \sigma, t = s \bullet ipurge_L(\sigma)$.

Lemma 4.2. Let M be a security system model, we have $M \not\models INI$ iff in M^2, there are two states $(s,s), (r,t)$ such that (s,s) is D reachable from (s_{in}, s_{in}), (r,t) is (H,L) reachable from (s,s), and $O_L(r) \neq O_L(t)$.

Proof. ($\Rightarrow$) We suppose $M \not\models INI$. Then there exists a counterexample $\sigma = \sigma_0 \cdots \sigma_k$. We consider two cases. On case is that $\bigcup_{i=0}^{k} \{dom(\sigma_i\} \subseteq \{H, L\}$,i.e. there are no actions associated with the domain D occurring in σ. Without loss of generality, we assume that $dom(\sigma_0) = \cdots = dom(\sigma_i) = L, dom(\sigma_{i+1}) = \cdots = dom(\sigma_k) = H$. Then $ipurge_L(\sigma) = \sigma_0 \cdots \sigma_i$. Let $(s_{in}, s_{in}) = (s,s)$. We consider the run of M after inputting σ. According to the definition of R^2, we have that $((s_j, s_j), \sigma_j, (s_{j+1}, s_{j+1})) \in R^2$ with $0 \le j \le i$. For the state (s_{i+1}, s_{i+1}), since $\sigma_{i+1} \in \Sigma_H, ((s_{i+1}, s_{i+1}), \sigma_{i+1}, (R(s_{i+1}, \sigma_{i+1}), s_{i+1})) \in R^2$. In the same way a run of system from (s_{i+1}, s_{i+1}) after inputting $\sigma_{i+1} \cdots \sigma_k$ can be represented as (s_{i+1}, s_{i+1}) $,\sigma_{i+1}$, $(R(s_{i+1}, \sigma_{i+1}), s_{i+1})$, σ_{i+2}, $(R(s_{i+1} \bullet (\sigma_{i+1}\sigma_{i+2})), s_{i+1})$, $\cdots, (R(s_{i+1} \bullet (\sigma_{i+1} \cdots \sigma_k)), s_{i+1})$. From above analysis, it is easy to justify $s_{i+1} = s_{in} \bullet ipurge_L(\sigma), (s_{i+1} \bullet (\sigma_{i+1} \cdots \sigma_k)) = s_{in} \bullet \sigma$. Let $s_{i+1} = s = t, r = R(s_{i+1} \bullet (\sigma_{i+1} \cdots \sigma_k))$. From the definition of counterexample, $O_L\ (\ r\ ) \neq O_L\ (t)$.

Another case is $D \in \bigcup_{i=0}^{k} \{dom(\sigma_i\}$,i.e. there exists an action associated with the domain D occurring in σ. Let i be an integer satisfying $\sigma_i \in \Sigma_D, \sigma_j \in \Sigma_H \cup \Sigma_L$ with $i < j \le k$. From Lemma 4.1, $ipurge_L(\sigma) = \sigma_0 \cdots \sigma_i \circ ipurge_L(\sigma_{i+1} \cdots \sigma_k)$. Let $s_{in} = s_0$. Then $s_0 \bullet ipurge_L(\sigma)$

$= (s_0 \bullet (\sigma_0 \cdots \sigma_i)) \bullet ipurge_L(\sigma_{i+1} \cdots \sigma_k)$. Let $s_{i+1} = s_0 \bullet (\sigma_0 \cdots \sigma_i))$. We now consider the computation of $s_{i+1} \bullet ipurge_L(\sigma_{i+1} \cdots \sigma_k)$. Since $\sigma_j \in \Sigma_H \cup \Sigma_L$ with $i < j \leq k$, the computation $s_{i+1} \bullet ipurge_L(\sigma_{i+1} \cdots \sigma_k)$ can be reduced to the first case. Let $s = s_{i+1}, r = s_{i+1} \bullet (\sigma_{i+1} \cdots \sigma_k), t = s_{i+1} \bullet ipurge_L(\sigma_{i+1} \cdots \sigma_k)$. Then $r = s_{in} \bullet \sigma, t = s_{in} \bullet ipurge_L(\sigma)$.From the definition of counterexample, $O_L\ (\ r\ ) \not\models O_L\ (t)$.

($\Leftarrow$)For the case $s = s_{in}$, there exists an action sequence σ such that $r = s_{in} \bullet \sigma, t = s_{in} \bullet ipurge_L(\sigma)$. Since $O_L(r) \neq O_L(t)$, σ is a counterexample.

We now suppose the action sequence $\sigma_0 \cdots \sigma_n$ satisfies $\sigma_n \in \Sigma_D$ and $((s_{in}, s_{in}),\ \sigma_0\ ,\ (\ s_1,\ s_1\ )) \in R^2$, $((\ s_i\ ,s_i)\ ,\ \sigma_i\ ,\ (s_{i+1}\ ,\ s_{i+1}\ )) \in R^2$ for all $1 \leq i \leq n$, and $s_{n+1} = s$. Consider $((s_i, s_i), \sigma_i, (s_{i+1}, s_{i+1})) \in R^2$ which implies that $(s_i, \sigma_i, s_{i+1}) \in R$. Henceforth, $s = s_{in} \bullet \sigma$. From definition 4.5, there exists an action sequence $\alpha \in (\Sigma_L \cup \Sigma_H)^*$ such that $r = s \bullet \alpha, r = s \bullet ipurge_L(\alpha)$. That is $r = s_{in} \bullet (\sigma \circ \alpha), t = s_{in} \bullet (\sigma \circ ipurge_L(\alpha))$. Form Lemma 4.1, $s_{in} \bullet ipurge_L(\sigma \circ \alpha) = s \bullet \sigma \bullet ipurge_L(\alpha)$. Since $O_L(r) \neq O_L(t)$, $\sigma \circ \alpha$ is a counterexample.

Let $|M^2|$ be the number of states in M^2. In M^2, if a state s^2 is reachable from another state s'^2, then s^2 is also reachable from s'^2 within $|M^2|$ steps. Henceforth, from Lemma 4.2 we have that if $M \not\models INI$, then there does not exist counterexamples of length no more than $2|M^2|$. However, it is unsatisfactory because the algorithm considers some unnecessary number of iterations before it terminates, for a system satisfying INI. We would like to consider only loop-free paths between pairs of states which more less than $|M^2|$. This insight leads to a CT.

Definition 4.6. In a SLKS M, we call a finite path s_0, σ_0, $\cdots$, σ_{k-1}, s_k of M is a loop-free path if and only if for any $0 \leq i < j \leq k, s_i \neq s_j$.

Definition 4.7. (Recurrence Diameter):The recurrence diameter of a SLKS M, denoted by $rd(M)$ is the length of the longest loop-free path (defined by the number of its edges) in M between any two reachable states.

From the above definition, it is easy to justify that for the double construction M^2 of the SLKS M, any reachable states are reachable from another state within $rd(M^2)$ steps. Henceforth, in previous algorithms we can restrict the paths to be loop free. For simplicity, we use s_i^2 instead of (s_i^1, s_i^2).

Theorem 4.1. Let M be a security system model, we have $M \not\models INI$ iff there exists counterexamples of length no more than $2 \times rd(M^2)$.

Theorem 4.1 says that when checking whether $M \models INI$, we only need to check whether there are counterexamples of length no more than $2 \times rd(M^2)$.

4.3 Reducing Counterexample Search to SAT

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

Given a SLKS structure M, and a bound k, we will construct a propositional formula $[M, INI]_k$. The variables $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ in $[M, INI]_k$ denote an alternating finite sequence of states and actions on a path. The formula $[M, INI]_k$ essentially represents constraints on $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ such that $[M, INI]_k$ is satisfiable iff there exists a counterexample of length k. To construct $[M, INI]_k$, we first define a propositional formula $[M]_k$ that constrains $s_0, \sigma_0, ..., \sigma_{k-1}, s_k$ to be a valid path in M. Second, we give the translation of a counterexample of length k to a propositional formula.

Definition 4.8. (Unfolding the Transition Relation): For a SLKS M, a positive integer k, $[M]_k = \bigwedge_{i=0}^{k-1} R(s_i,\ \sigma_i,\ s_{i+1})$.

We recall that intransitive noninterference says that the purged H actions are not allowed to lead to any effects observable to L. Henceforth, for the action sequence $\sigma = \sigma_0 \cdots \sigma_{k-1}$, we need to compare $O_L(s_0 \bullet \sigma)$ and $O_L(s_0 \bullet ipurge_L(\sigma))$. Lemma 4.1 has shown how to compute $O_L(s_0 \bullet ipurge_L(\sigma))$. First, find the last position of actions associated with the domain D occurring in σ. Let m represent the position. If there are no actions associated with the domain D, let $m = -1$. Second, compute $ipurge_L(\sigma_{m+1} \cdots \sigma_{k-1})$. We suppose that there are i actions associated with the domain L in $\sigma_{m+1} \cdots \sigma_{k-1}$, define the following $[H]_k^{m,i}$ to encode the distributing of these actions in σ.

$$[H]_k^{m,i} = \sigma_m \in \Sigma_D \wedge m < l_1 < k \wedge m < l_i < k \wedge \bigwedge_{j=1}^{i-1} (l_j < l_{j+1}) \wedge \bigwedge_{j=1}^{i} (\sigma_{l_j} \in \Sigma_L) \wedge \bigwedge_{0 \leq j \leq k-1}^{j \notin \{l_1, \cdots, l_i\}} (\sigma_j \in \Sigma_H),$$ where $\sigma_{-1} \in \Sigma_D$ is always true.

Then we define $[M]_L^{m,i}$ to encode the execution of the system after inputting $ipurge_L(\alpha)$.

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

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

Definition 4.9. (General Translation): For a SLKS M, a positive integer k,
$$[M, INI]_k = I(s_0) \wedge [M]_k \wedge \bigvee_{m=-1}^{k-1} (\bigvee_{i=0}^{\min(k-1-m, k-1)} ([H]_k^{m,i} \wedge [M]_L^{m,i} \rightarrow O_L(s_k) \neq O_L(s'_{l_i})))$$

Theorem 4.2. For a SLKS M, a positive integer k, $[M, INI]_k$ is satisfiable if and only if for intransitive noninterference there exists a counterexample of length k.

Theorem 4.2 says that we can check whether there exists a counterexample of length k by a SAT solver. We now consider establishing a propositional formula to encode the recurrence diameter. We define $loopfree(s_0^2, \sigma_0, ..., \sigma_{k-1},$

$s_k^2) = \bigwedge_{i=0}^{k-1} R^2(s_i^2, \sigma_i, s_{i+1}^2) \wedge \bigwedge_{0\leq i<j\leq k} (s_i^2 \neq s_j^2)$. It is easy to justify $rd(M^2)$ is the minimal integer such that $loopfree(s_0^2,\ \sigma_0,\ ...,\ \sigma_{k-1},\ s_k^2)$ unsatisfiable. The solution of using a SAT solver to checking intransitive noninterference is given in pseudo-code below (Algorithm 1).

Algorithm 1. Checking Intransitive Noninterference based on SAT

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

4.4 An Example

In this subsection we use an example to show our translation procedure. Consider a machine M shown in Figure 3. In M, there are six states:$s_0, \ldots, s_5$. We need three boolean variables v_1, v_2, v_3 to encode states. And need to introduce three additional boolean variables $v_1^{'}, v_2^{'}, v_3^{'}$ to encode successor states. The machine has three actions:h, l, d. Since these commands belong to three different domains, we need two boolean variables u_1, u_2 to encode actions. The detailed encoding is given in Table 1.

The recurrence diameter of M is 5. We consider whether there are counterexamples of length 2. Let $k = 2$. The variables $s_0, \sigma_0, s_1, \sigma_1, s_2$ denote an alternating finite sequence of states and actions on a path. For simplicity, in the boolean variables encoding states(actions), for $0 \leq i \leq k$ we use $s_i[1](\sigma_i[1])$ to represent the first boolean variable, $s_i[2](\sigma_i[2])$ to represent the second boolean variable, $s_i[3]$ to represent the third boolean variable. Thus $[M]_2 = (\neg s_0[1] \wedge \neg s_0[2] \wedge \neg s_0[3]) \wedge\ R(s_0[1], s_0[2], s_0[3],\ \sigma_0[1], \sigma_0[2],\ s_1[1],\ s_1[2],\ s_1[3])\ \wedge\ R(s_1[1],\ s_1[2], s_1[3],\ \sigma_1[1],\ \sigma_1[2], s_2[1], s_2[2], s_2[3])$.

For the action sequence $\sigma = \sigma_0\sigma_1$, according the distribution of action d and l, $ipurge_L(\sigma)$ belongs to the following action sequence set: $\{\varepsilon, \sigma_0, \sigma_1, \sigma_0\sigma_1\}$. For the case $ipurge_L(\sigma) = \sigma_0\sigma_1$, it shows that h does not occur in σ. Therefore, we do not need to consider this case. We use a boolean variable l_1 to encode positions of l in σ. The detailed computation of $[H]_2^{m,i}$ and $[M]_L^{m,i}$ is given in Table 2.

Therefore, we have that $[M, INI]_2 = I(s_0) \wedge [M]_2 \wedge \bigvee_{m=-1}^{1} (\bigvee_{i=0}^{\min(1,1-m)} ([H]_2^{m,i} \wedge [M]_L^{m,i} \rightarrow O_L(s_2) \neq O_L(s_{l_i}^{'})))$. It is easy to justify that $[M, INI]_2$ is not satisfiable. That is, there are no counterexamples of length 2. Further, we find that $[M, INI]_4$ is satisfiable. That is the machine M does not satisfy intransitive noninterference. For example $hdhl$ is a counterexample.

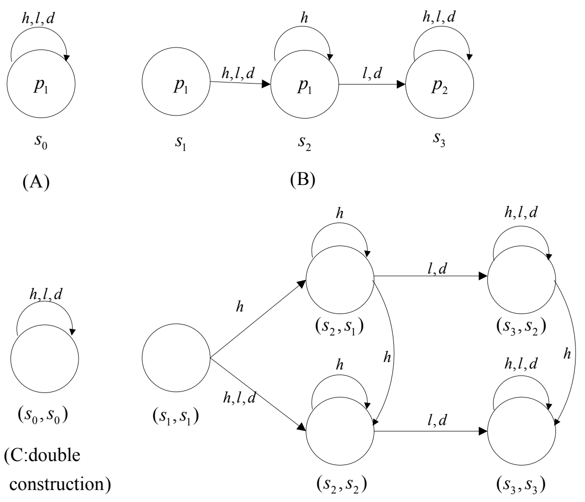

Fig. 5 An example showing the failure of the classical induction

5 Combining Induction

In Algorithm 1, if $M \models INI$, then the program must iterate $2 \times rd(M^2)$ times. This is not feasible. In this section 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 it possible to handle larger models since the induction step has to consider only paths of length k.

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

– For all states (s_0^1, s_0^2), if $I((s_0^1, s_0^2))$ holds, then $O_L(s_0^1) = O_L(s_0^2)$.
– For all paths $(s_k^1, s_k^2), \sigma_k, (s_{k+1}^1, s_{k+1}^2)$, if $O_L(s_k^1) = O_L(s_k^2)$ and $\sigma_k \in (\Sigma_H \cup \Sigma_L)$, then $O_L(s_{k+1}^1) = O_L(s_{k+1}^2)$.

Consider the following tiny example shown in Figure 5. In this example, the system M consists of two components: A and B. If s_0 is the initial state of the system, then it is easy to justify that the system M satisfies intransitive noninterference. However, in this case the classical induction technique can not be used to prove $M \models INI$ successfully. The reason of the classical induction technique failure is that the technique considers all states including reachable and unreachable states. For example, in the double construction of M shown in Figure 5, although the state (s_2, s_1) is not reachable from the

Table 1 Encoding

object	s_0	s_1	s_2	s_3	s_4	s_5	h	l	d
encoding	000	001	010	011	100	101	11	01	00

Table 2 Computation of $[H]_2^{m,i}$ and $[M]_L^{m,i}$

m	i	$ipurge_L(\sigma)$	$[H]_2^{m,i}$	$[M]_L^{m,i}$
-1	1	σ_{l_1}	$\neg\sigma_{l_1}[1]\wedge\sigma_{l_1}[2]\wedge\sigma_{\neg l_1}[1]\wedge\sigma_{\neg l_1}[2]$	$s_0 = s_0^{'}\wedge R(s_0^{'},\sigma_{l_1},s_1^{'})$
-1	0	ε	$\sigma_0[1]\wedge\sigma_0[2]\wedge\sigma_1[1]\wedge\sigma_1[2]$	$s_0 = s_0^{'}$
0	0	ε	$\neg\sigma_0[1]\wedge\neg\sigma_0[2]\wedge\sigma_1[1]\wedge\sigma_1[2]$	$s_0 = s_0^{'}$

initial state, the classical induction technique still needs to prove that $O_L(s_2) = O_L(s_1)$ implies $O_L(s_3) = O_L(s_2)$. But this is not true.

Window induction is a modified induction technique which has been used to prove a hardware system design [23]. 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 intransitive noninterference windowed induction with window size $k \geq 0$ consists of the following two steps:

–Prove that for all paths $(s_0^1,s_0^2),\sigma_0,\ldots,\sigma_{k-1},(s_k^1,s_k^2)$, if $I((s_0^1,s_0^2))$, then $O_L(s_i^1) = O_L(s_i^2)$ for all $0 \leq i \leq k$.
–Prove that for all paths $(s_0^1,s_0^2),\sigma_0,\ldots,\sigma_{k-1},(s_k^1,s_k^2)$, $\sigma_k,(s_{k+1}^1,s_{k+1}^2)$, if for all $0 \leq i \leq k$, $O_L(s_i^1) = O_L(s_i^2)$, then $O_L(s_{k+1}^1) = O_L(s_{k+1}^2)$.

If the first subgoal can be proved, then there are no counterexamples of length k. Therefore, the first step can be completed by checking whether $[M,INI]_k$ is satisfiable. The second step can be completed by checking whether a corresponding propositional formula is a tautology. We first recall the definition of $[M]_k$. Then we have that $[M^2]_k = \bigwedge_{i=0}^{k-1} R^2((s_i^1,s_i^2),\sigma_i,\ (s_{i+1}^1,s_{i+1}^2))$. Let $[M,INI]_k^{IN} = ([M^2]_{k+1} \wedge \bigwedge_{i=0}^{k}(O_L(s_i^1) = O_L(s_i^2)) \rightarrow O_L(s_{k+1}^1) = O_L(s_{k+1}^2)$. It is easy to justify that $[M,INI]_k^{IN}$ is a tautology if and only if the conclusion we must prove in the second step of windowed induction is correct. We can then safely conclude that the system satisfies noninterference. This solution is given in pseudo-code (Algorithm 2).

Algorithm 2. Checking Intransitive Noninterference based on Recurrence Diameter and SAT

```
{
$k = 1$
While $loopfree(s_0^2,\sigma_0,\ldots,\sigma_{k-1},s_k^2)$ is satisfiable do
    if $[M,INI]_k$ is satisfiable return the counterexample $s_0,\sigma_0,\ldots,\sigma_{k-1},s_k$
    if $[M,INI]_k^{In}$ is a tautology return True
    $k = k+1$
End While
return True
}
```

6 Experimental results

The solution we proposed mainly consists of two components: the counterexample search component $[M,INI]_k$, and the induction proof component $[M,INI]_k^{IN}$.

In this section we will evaluate these two components. We conducted experimental evaluation using a Linux workstation with a 3.06GHZ Pentium processor and 2048MByte memory. We choosed SATO [18] as the propositional prover since it is a very efficient implementation of the Davis&Putnam procedure. All benchmarks used in the experiment were taken from [25]. They have been converted from communicating state machines to Security Labeled Kripke Structures.

In the conversion, for each action we assigned a security class randomly. We collected three kinds of assignment satisfying that the length of the minimal counterexample are 4, 12 and 16 respectively. The experimental results can be found in Table 3. The times reported are the average of 5 runs. The columns are

–Problem: The problem name with the size of the instance in parenthesis.
–States: Number of reachable states in the SLKS.
–Actions: Number of actions in the SLKS.
–k: The time in seconds required by Algorithm 2 to find a counterexample for the value of k.

In Table 3 N/A means the computation time is too long and over the pre-set time. The set of experiments we used is too small to say anything conclusive about the performance of our methods. There is, however, still an interesting observation to be made: SAT based verification of intransitive noninterference is typically

Table 3 Experiments

Problem	States	Actions	$k = 4$	$k = 12$	$k = 16$
ELEV(1)	158	99	1.13	11.76	20.74
ELEV(2)	1062	299	18.02	314.56	412.76
ELEV(3)	7121	783	411.95	N/A	N/A
ELEV(4)	43440	1939	3256.28	N/A	N/A
MMGT(2)	817	114	3.06	85.41	159.91
MMGT(3)	7703	172	56.87	765.87	N/A
MMGT(4)	66309	232	363.15	N/A	N/A
RING(3)	87	33	0.45	8.86	26.45
RING(5)	1290	55	3.88	47.09	210.19
RING(7)	17000	77	93.82	843.67	N/A
RING(9)	211528	99	625.84	N/A	N/A
RW(9)	523	181	4.91	57.24	108.79
RW(12)	4110	313	77.63	963.95	N/A
RURNACE(1)	344	37	0.41	4.23	32.73
FURNACE(2)	3778	65	10.26	119.15	544.45
FURNACE(3)	30861	99	152.94	N/A	N/A

faster in finding counterexamples; the deeper the counterexample is, the less advantage our approach has.

7 Conclusions and Future Work

The main contribution of this paper is to present an algorithmic approach to checking intransitive noninterference.
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 plain SAT-solver.

Other contributions includes: in order to make the search procedure terminate as soon as possible, to the length of minimal counterexamples we propose an over-approximation which also can be checked by a plain SAT-solver .

There are many interesting avenues for future research. Our current work concentrates on two directions. First, we are extending our approach to other information flow security properties. Second, we are modifying the predicate abstraction technique such that we can abstract the finite state behaviors from infinite state systems while preserving intransitive noninterference.

Acknowledgement

The authors acknowledge the financial support of the National Natural Science Foundation of China(project No. 61003288, 61111130184), the Ph.D. Programs Foundation of Ministry of Education of China (project No. 20093227110005), the Natural Science Foundation of Jiangsu Province (project No. BK2010192), and the Fundamental Research Project of the Natural Science in Colleges of Jiangsu Province (project No. 12JDG061).
The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] ZHOU Conghua, LIU Zhifeng, WU Hailing, et al. Syntactic information flow analysis based on security policy. Journal of Frontiers of Computer Science and Technology, **5**, 179-192 (2011).
[2] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. IEEE Transactions on Software Engineering, **27**, 550-571 (1997).
[3] J. Millen. 20 Years of Covert Channel Modeling and Analysis. In Proceedings of IEEE Symposium on Security and Privacy, 113-114, (1999).
[4] Goguen and J. Meseguer, Security Policies and Security Models. Proceedings of the IEEE Symposium on Security and Privacy, Oackland, California, 11-21 (1982).
[5] Conghua Zhou, Li Chen, Shiguang Ju, Zhifeng Liu, State Space Reduction for Verifying Noninterference, The Third IEEE International Conference On Secure Software Integration and Reliability Improvement,132-140 (2009).
[6] Conghua Zhou, Symbolic Algorithmic Verification of Generalized Noninterference, WSEAS TRANSACTIONS on COMPUTERS, **8**, 976-987 (2009).

[7] Zhou Conghua, Chen Li, Ju Shiguang, Petri Nets based Noninterference Analysis, Journal of Computational Information Systems, **5**, 1231-1240 (2009).

[8] J. Rushby, Noninterference, transitivity, and channelcontrol security, Technical report, Computer Science Laboratory, SRI International, (1992).

[9] S. Pinsky, Absorbing covers and intransitive noninterference, In Proc. 16th IEEE Symposium on Security and Privacy, 102-113 (1995).

[10] A. Roscoe and M. Goldsmith, What is intransitive noninterference?, In Proc. 12th IEEE Computer Security Foundations Workshop (CSFW), 226-238 (1999).

[11] Ron van der Meyden. What, Indeed, Is Intransitive Noninterference Proc. of ESORICS'07, Springer LNCS, **4734**, 235-250 (2007).

[12] H. Mantel, Information flow control and applications-bridging a gap, In Proc. 10th Symposium on Formal Methods Europe (FME 2001), of Lecture Notes in Computer Science, Springer-Verlag, **2021**, 153-172 (2001).

[13] Hadj-Alouane, N. B. Lafrance, S. Feng Lin Mullins, J. Yeddes, M. M. On the verification of intransitive noninterference in mulitlevel security. IEEE Transactions on Systems, Man, and Cybernetics, Part B, **35**, 948-958 (2005).

[14] Hadj-Alouane, N. B. Lafrance, S. Feng Lin Mullins, J. Yeddes, M. , Characterizing intransitive noninterference for 3-domain security policies with observability. IEEE Transactions on Automatic Control, **50**, 920-925 (2005).

[15] E. Goldberg and Y. Novikov. BerkMin: a Fast and Robust Sat-Solver. In Proceedings of Design Automation and Test in Europe (DATE), 142-149 (2002).

[16] H. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In Proceedings of the 38th Design Automation Conference (DAC), 530-535 (2001).

[17] Hantao Zhang. SATO: An Efficient Propositional Prover. In William McCune, editor, Proceedings of the 14th International Conference on Automated Deduction (CADE), volume 1249 of Lecture Notes in Computer Science, Springer, 272-275 (1997).

[18] Joao P. Marques-Silva and Karem A. Sakallah. GRASP-a search algorithm for propositional satisfiability. IEEE Transactions in Computers, **48**, 506-521 (1999).

[19] Nina Amla, Robert Kurshan, Kenneth McMillan, and Ricardo Medel. Experimental Analysis of Different Techniques for Bounded Model Checking. Proceedings of the 9th International Conference on Tools and Algorithms for the Construction andAnalysis of Systems (TACAS), of Lecture Notes in Computer Science, Springer, **2619**, 34-48 (2003).

[20] Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. Proceedings of the 12th International Conference on Computer-Aided Verification (CAV), number 1855 in Lecture Notes in Computer Science, Springer, 99-113 (2000).

[21] Per Bjesse, Tim Leonard, and Abdel Mokkedem. Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers. Proceedings of the 13th International Conference on Computer Aided Verification (CAV), of Lecture Notes in Computer Science, Springer, **2102**, 454-464 (2001).

[22] Fady Copti, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of Bounded Model Checking in an Industrial Setting. Proceedings of the 13th International Conference on Computer Aided Verification (CAV), of Lecture Notes in Computer Science, Springer, **2102**, 436- 453 (2001).

[23] Roy Armoni, Limor Fix, Ranan Fraer, Scott Huddleston, Nir Piterman, Moshe Y. Vardi,SAT-based Induction for Temporal Safety Properties, Electronic Notes in Theoretical Computer Science, **119**, 3-16 (2005) .

[24] Joseph A. Goguen and Sose Meseguer. Unwinding and Inference Control, Proceedings of the Symposium on Security and Privacy. IEEE Computer Society, 75-86, (1984).

[25] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. IEEE Transactions on Software Engineering, **22**, 161-180 (1996).

Liu Zhifeng was born in 1981. He received his Ph.D degree in Acoustics from Nanjing University in 2011. His research interests include model checking, information security, modal logics. He is a lecture of school of computer science and telecommunication engineering, Jiangsu University.

Zhou Conghua was born in 1978. He received his Ph.D degree in mathematics from Nanjing University in 2006. His research interests include model checking, information security, modal logics, trusted embedded software, and multi-agent systems. He is an associate professor of school of computer science and telecommunication engineering, Jiangsu University.

Ge Yun was born in 1970. He received his Ph.D degree in biomedical engineering from Southeast University in 2001. He is a professor of School of Electronic Science and Engineering, Nanjing University, China. His research interests model checking,medical image processing.

Zhang Dong was born in 1969. He received his Ph.D degree in Acoustics from Nanjing University in 1995. He is a professor of School of Electronic Science and Engineering, Nanjing University,China. His research interests model checking, nonlinear acoustics, biomedical engineering.