Electronic Notes in Theoretical Computer Science 168 (2007) 61–75 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Algorithmic Verification of Noninterference Properties

# Ron van der Meyden Chenyi Zhang

*School of Computer Science and Engineering, University of New South Wales,*

*Sydney, Australia*

*National ICT Australia, Locked Bag 6016, Sydney, NSW 1466, Australia*

Abstract

The paper discusses the problem of model checking a number of noninterference properties in finite state systems: Noninterference, Nondeducibility on Inputs, Generalised Noninterference, Forward Correctability and Restrictiveness. The complexity of these problems is characterized, and a number of possible heuristics for optimization of the model checking are discussed.

*Keywords:* noninterference, model checking, complexity, heuristics

# Introduction

The notion of ‘noninterference’ is a general term applied in the security literature to a number of causality-like notions intended to capture the intuition that informa- tion does not flow from high level users to lower level users, so that confidentiality of high level information is maintained. The main approach to verification that systems satisfy these properties has been proof theoretic methods using so-called ‘unwinding conditions’. In this paper, we investigate the applicability of algorithmic verification techniques when the systems in question are finite state. We develop algorithms for model checking a number of different noninterference notions, and characterize the computational complexity of the associated verification problems. In particular, we deal with Noninterference on deterministic systems [[12](#_bookmark23),[25](#_bookmark36)], Nond- educibility on Inputs [[26](#_bookmark37)], Generalised Noninterference [[18](#_bookmark27)], Forward Correctability

[[15](#_bookmark26)] and Restrictiveness [[18](#_bookmark27)].

*٨* National ICT Australia is funded through the Australian Government’s *Backing Australia’s Ability* ini- tiative, in part through the Australian Research Council

1571-0661 © 2007 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2006.11.002

Noninterference has been studied under several distinct semantic models, includ- ing state based models [[12](#_bookmark23),[25](#_bookmark36)], trace-set models [[20](#_bookmark28),[30](#_bookmark41)] and process algebras [[23](#_bookmark34),[8](#_bookmark19)]. Only for the latter has there been a systematic study of algorithmic verification of these notions [[8](#_bookmark19),[9](#_bookmark20)]. The process algebraic models are the most expressive, and def- initions of noninterference notions on other models can be reduced to definitions of noninterference notions on a process algebraic model by means of natural mappings between the models [[28](#_bookmark39)]. However, state based system modelling approaches are more natural to many, are likely to be adequate for many applications, have a more extensive literature on algorithmic verification, and have a more highly developed set of verification tools. This modelling approach also remains the predominant ap- proach in operating systems verification efforts [[14](#_bookmark25)], the area originally motivating the noninterference literature. It therefore makes sense to consider the algorithmic verification problem also on state based models. This is particularly so with respect to complexity bounds, where lower bounds proved for a more expressive semantic model may not apply on a more restrictive model. We therefore focus in this pa- per on a state based modelling of systems, and (to make the verification problem decidable) restrict attention to finite state systems.

The contributions of the paper are as follows. First, we show that noninterference in deterministic systems can be reduced to a *safety* property, so it is expressible in both branching time and linear time temporal logics and verifiable in polynomial time by existing model checkers. Also in PTIME is the notion of Restrictiveness on nondeterministic systems. We show that the remaining notions of noninterference on nondeterministic systems that we consider are PSPACE-complete. For some of these notions (Restrictiveness and Nondeducibility on Inputs), these results are closely related to results of Focardi and Gorrieri [[8](#_bookmark19),[9](#_bookmark20)] (but on a more restricted semantic model, hence not immediate consequences for the lower bounds). The results on Generalised Noninterference and Forward Correctability are new, as far as we know. Finally, we discuss heuristics that may be applied to the verification of noninterference notions, and give complexity arguments that suggest that these heuristics may sometimes lead to optimizations.

# State-Observed Model

The state based system models in the literature on noninterference can be roughly classified into two distinct types, depending on whether observations are associated with states [[21](#_bookmark29),[3](#_bookmark12),[24](#_bookmark35)] or actions [[12](#_bookmark23),[25](#_bookmark36)]. The system definitions are similar to those of finite state automata, with the distinction between the two types resembling the Mealy/Moore distinction. It can be shown [[28](#_bookmark39)] that there exist natural mappings between these two types of models that preserve all the security notions that we consider in this paper. Consequently, we consider only the state-observed modeling. The systems are input-enabled, in the sense that any action can be taken at any time. Most of the literature restricts attention to two agents High (*H*) and Low (*L*) and the security policy *L* ≤ *H*. This policy permits information to flow from Low to High but not from High to Low. We also make this restriction here, and take

the set of agents (also called *domains*) to be *D* = {*L, H*}.

A nondeterministic *state-observed* state machine is a tuple of the form *M* =

⟨*S, s*0*, next, obs, d om, A*⟩ where *S* is a set of states; *s*0 ∈ *S* is the initial state; *A* is a set of actions; the function *next* : *S* × *A* → P(*S*) \ {∅} is a transition function, such that *next*(*s, a*) defines the set of states to which it is possible to make a transition when action *a* ∈ *A* is performed at a state *s* ∈ *S*; the function *d om* : *A* → *D* associates a security domain with each action, and the function *obs* : *S* × *D* → *O* describes the observation made in each state by each security domain. For readability, we ‘curry’ the function *obs* by *obsu* of type *S* → *O* if *u* ∈ *D*. Such a state-machine is *deterministic* if *next*(*s, a*) is a singleton for all states *s* and actions *a*. In this case we may define a function *step* : *S* × *A* → *S* by *next*(*s, a*) = {*step*(*s, a*)}. We write M*ns* for the set of all nondeterministic state- observed machines, and M*s* for the set of all deterministic state-observed machines. A *run* of a state-observed system is a sequence *r* = *s*0*a*1*s*1*a*2*s*2 *... ansn* ∈ *S*(*AS*)∗ such that for all 1 ≤ *i* ≤ *n*, *si* ∈ *next*(*si*−1*, ai*). Define *r*(*i*) = *si* to be the *i*- th state on the run *r* and *ra*(*j*) = *aj* to be the *j*-th action. We use two types of concatenation operation on sequences. We write *α* · *β* for the usual notion of concatenation. A run can also be described as a *fusion* of two sequences: we write *r* = *r*1 ◦ *r*2 if there exists *m* with 1 ≤ *m* ≤ *n* such that *r*1 = *s*0*a*1*s*1*a*2*s*2 *... amsm* and *r*2 = *smam*+1*sm*+1*am*+2*sm*+2 *... ansn*. With respect to the simple policy *L* ≤ *H*, we

define *AH* = {*a* ∈ *A* | *dom*(*a*)= *H*} and *AL* = {*a* ∈ *A* | *dom*(*a*)= *L*}.

# State Based Security Definitions

In this section we recall from the literature a number of classical security defini- tions. Some of them are state based and some were originally defined as a trace-set property, in which case we give a corresponding definition in our system model.

Several of the definitions are cast in terms of a notion of *view* capturing the information at an agent’s disposal in a run. We take the view to be the maximal information that an agent can have in a nondeterministic asynchronous system: its sequence of actions and observations reduced modulo stuttering. Let *Cond* : *X*∗ → *X*∗ be the function which condenses a sequence of elements into a possibly shorter sequence by removing stuttering, such that for all *a, b* ∈ *X*, *α* ∈ *X*∗, *Cond*(*ϵ*)= *ϵ*, *Cond*(*a*)= *a*,

⎧

and *Cond*(*α* · *a* · *b*)= ⎨ *Cond*(*α* · *a*) · *b* if *a* /= *b,*

⎩ *Cond*(*α* · *a*) otherwise*.*

Definition 3.1 For *u* ∈ *D*, define the observation function *Obsu* : *S*(*AS*)∗ →

*O*+(*AO*+)∗ on a run by *Obsu*(*s*)= *obsu*(*s*), and

*Obsu*

(*δ* · *a* · *s*)= ⎧⎨

⎩

*Obsu*(*δ*) · *a* · *obsu*(*s*) if *dom*(*a*)= *u Obsu*(*δ*) · *obsu*(*s*) otherwise.

Define the function *viewu* : *S*(*AS*)∗ → *O*+(*AO*+)∗ by *viewu*(*r*)= *Cond* (*Obsu*(*r*)).

Note that an agent may make the same observation several times in a row, without an intervening action by that agent. This indicates that another agent has acted. To eliminate this timing-based reasoning, in order to make the definition compatible with the assumption of asynchrony, we apply the function *Cond* in this definition.

* 1. *Noninterference*

Historically, one of the first information flow properties was (transitive) noninter- ference [[12](#_bookmark23),[13](#_bookmark24)], defined with respect to deterministic machines. We base our dis- cussion on the presentation of Rushby [[25](#_bookmark36)], which has been followed in many other works. As noted above, in state-observed deterministic systems, we have a function *step* : *S* ×*A* → *S* to represent the deterministic state evolution as a result of actions. To represent the result of executing a sequence of actions, define the operation • : *S* × *A*∗ → *S*, by *s* • *ϵ* = *s*; and *s* • (*α* · *a*)= *step*(*s* • *α, a*)*.*

With respect to the simple policy *L* ≤ *H*, the definition of Noninterference can

be described in terms of the operation *purgeL* : *A*∗ → *A*∗

*L*

on sequences of actions

that restricts the sequence to the subsequence of actions of *L*. Intuitively, the purged *H* actions are not allowed to lead to any effects observable to *L*. This is formalised as follows in the definition of Noninterference.

Definition 3.2 A (deterministic) system in M*s* satisfies *Noninterference* if for all *α* ∈ *A*∗, we have *obsL*(*q*0 • *α*)= *obsL*(*q*0 • *purgeL*(*α*)). We write *NIs* for the set of such systems.

* 1. *Nondeducibility on Inputs*

One way of understanding the statement that *H* does not interfere with *L* in a deterministic system is as stating that every sequence of *H* actions is compatible with the actions and observations of *L*. This leads to the proposal to take a similar notion as the formulation of noninterference in nondeterministic systems: an ap- proach known as Nondeducibility [[26](#_bookmark37)]. Nondeducibility is defined in a quite general way, in terms of a pair of *views* of runs. We focus here on a commonly used special case: *L*’s nondeducibility of *H*’s actions.

To state the definition of nondeducibility, we also require a function to extract the sequence of actions performed by an agent. We write *Actu*(*r*) for the sequence of actions performed by agent *u* in run *r*, and *Act*(*r*) the whole action sequence from all the agents in *r*.

Definition 3.3 [1](#_bookmark1) A system *M* satisfies *Nondeducibility on Inputs* if for every *α* ∈ *A*∗ , and every observation sequence *β* such that there exists a run *r* of *M* with *view L*(*r*) = *β*, there exists a run *r*' of *M* with *ActH*(*r*') = *α* and *view L*(*r*') = *β*. Write *NDIs* for the set of systems in M*ns* satisfying Nondeducibility on Inputs.

*H*

1 In [[28](#_bookmark39)] it has been shown that Nondeducibility on Inputs is equivalent to Nondeducibility on Strategies in *purely asynchronous* systems, though this is *not* true on synchronous machines due to [[29](#_bookmark40)].

* 1. *Generalised Noninterference*

Generalised Noninterference (GN) was proposed in [[18](#_bookmark27)] to generalise Noninterference to nondeterministic systems. The original definition of *GN* is a trace set property with the intuition that the changes on high-level input must not alter the possible future sequences of low-level events (by modifying *H* outputs somewhere). Here we formulate it on our state based systems as:

Definition 3.4 A system *M* satisfies Generalised Noninterference (GN) if

* + 1. for all runs *r* of *M* with *Act*(*r*)= *α*· *α*', and for all *a* ∈ *AH* , there exists another run *r*' such that *Act*(*r*')= *α* · *a* · *α*' and *view L*(*r*)= *view L*(*r*')
    2. for all runs *r* of *M* with *Act*(*r*)= *α* · *a* · *α*' with *a* ∈ *AH* , there exists another run *r*' such that *Act*(*r*')= *α* · *α*' and *view L*(*r*)= *view L*(*r*').

Write *GNs* for the set of systems in M*ns* satisfying Generalised Noninterference.

Note that this definition implies that every possible *L* observation is consistent with every sequence of *H* Actions, so *GNs* is at least as strong as *NDIs*. However, *GNs* is seemingly a stronger notion than *NDIs* in that the latter allows *L* to rule out certain possible *H/L* action interleavings whereas *GNs* requires that all interleavings are consistent.

* 1. *Forward Correctability*

Forward Correctability (FC) was first introduced in [[15](#_bookmark26)]. Similar to Generalised Noninterference, FC was defined as a property on sets of traces. We formulate it as follows in state-observed systems. Define the relation ≡ on runs by *r*1 ≡ *r*2

if *r*1 and *r*2 have the same length *n* ∈ N, *ra*(*j*) = *ra*(*j*) for all 1 ≤ *j* ≤ *n*, and

1 2

*obsu*(*r*1(*i*)) = *obsu*(*r*2(*i*)) for all 0 ≤ *i* ≤ *n* and *u* ∈ *D*.

Definition 3.5 A system *M* satisfies *Forward Correctability* (FC) if

* + 1. for all runs *r* = *r*1 ◦ *r*2 of *M* such that *Act*(*r*1)= *α* and *Act*(*r*2)= *α*' with *α*' ∈

*A*∗ , for all *a* ∈ *AH*, there exists a run *r*' = *r*' ◦ *r*' with *r*1 ≡ *r*' , *Act*(*r*' )= *a* · *α*'

*L*

and *view L*(*r*)= *view L*(*r*')

1 2 1 2

* + 1. for all runs *r* = *r*1 ◦ *r*2 of *M* such that *Act*(*r*1)= *α* and *Act*(*r*2)= *a* · *α*' with

*a* ∈ *AH* and *α*' ∈ *A*∗ , there exists a run *r*' = *r*' ◦ *r*' with *r*1 ≡ *r*' , *Act*(*r*' )= *α*'

*L*

and *view L*(*r*)= *view L*(*r*').

1 2 1 2

Write *FCs* for the set of systems in M*ns* satisfying Forward Correctability.

*FCs* is seemingly stronger than *GNs* because any ‘perturbation’ must be cor- rectable in the future for *FCs* but *GNs* allows it to be correctable either in the past or in the future.

* 1. *Restrictiveness*

There are two versions of ‘Restrictiveness’ introduced in McCullough’s early works. The former [[17](#_bookmark30)] is a trace based definition, while the latter is essentially defined on

labelled transition systems [[18](#_bookmark27),[19](#_bookmark31)]. In [[19](#_bookmark31)] McCullough mentions both definitions and concludes that the one on labelled transition systems is stronger. Here we follow the latter definition. This is close to the notion of unwinding relation which is a way of facilitating proofs of traditional noninterference on deterministic systems [[25](#_bookmark36)].

Definition 3.6 An *unwinding relation* for a system *M* ∈ M*s* is an equivalence relation ∼*L* on the states of *M* satisfying the following conditions, for all states *s, t* and actions *a*: [2](#_bookmark4)

* *Output Consistency*: if *s* ∼*L t* then *obsL*(*s*)= *obsL*(*t*);
* *Locally Respects*: if *a* ∈ *AH* then *s* ∼*L step*(*s, a*);
* *Step Consistency*: if *a* ∈ *AL* and *s* ∼*L t* then *step*(*s, a*) ∼*L step*(*t, a*).

The relationship between unwinding conditions and noninterference is given by the following classical results:

Theorem 3.7 [[13](#_bookmark24),[25](#_bookmark36)]

* 1. *If there exists an unwinding relation for M* ∈ M*s, then M* ∈ *NIs.*
  2. *If M* ∈ *NIs then there exists an unwinding relation for M.*

The following is a natural generalization of Definition [3.6](#_bookmark2) to nondeterministic systems.

Definition 3.8 An unwinding relation for a system *M* ∈ M*ns* is an equivalence relation satisfying

* OC: if *s* ∼*L t* then *obsL*(*s*)= *obsL*(*t*).
* LR: if *a* ∈ *AH* and *t* ∈ *next*(*s, a*) then *s* ∼*L t*,
* SC: if *a* ∈ *AL* and *s* ∼*L s*' and *t* ∈ *next*(*s, a*), then there exists *t*' ∈ *next*(*s*'*, a*) such that *t* ∼*L t*'.

McCullough’s [[19](#_bookmark31)] Restrictiveness definition is similar in spirit to unwinding but distinguishes between inputs and outputs on actions. Since on state-observed sys- tems, outputs are ‘embedded’ in states, the above is a somewhat simplified version of what was introduced by McCullough, and we have the following definition of Restrictiveness on state-observed systems.

Definition 3.9 *M* ∈ M*ns* satisfies *Restrictiveness*, written *M* ∈ *RESs*, if there exists an unwinding relation for *M* .

The property *RESs* is inherently stronger than the other security notions we have introduced here, since intuitively every ‘perturbation’ from *H* always leads *L* to a state which is observationally bisimilar to its original state. The following summarizes the known relations between the definitions we have introduced above.

2 We present a slight modification of the usual definition, which would have an equivalence relation ∼*u* for each agent *u*, satisfying a similar set of conditions for each *u*. For the policy *L* ≤ *H* we can take ∼*H* to be the universal relation, which automatically satisfies the necessary conditions.

Proposition 3.10 *The following containments are strict: RESs* ⊂ *FCs* ⊂ *GNs* ⊂ *NDIs. On deterministic systems, the notions NIs, NDIs, GNs, FCs and RESs are equivalent.*

# Verifying Noninterference Properties

We now turn to the main interest of this paper: verification methods and complexity results for the security properties introduced in the previous section. None of these properties is directly expressible in the traditional safety-liveness framework [[1](#_bookmark13)], as they express not a constraint on single system execution, but rather a constraint on the set of all possible executions. We therefore need to develop new techniques for their verification.

* 1. *Unwinding Characterizable Properties*

In this section we consider the property *NIs* and its generalization *RESs*, both of which can be characterized by an unwinding relation.

The following is a way to decide noninterference *NIs* on deterministic state- observed systems by a doubling construction. Given a deterministic system *M* =

⟨*S, s*0*, step, obs, d om, A*⟩, define *M* 2 = ⟨*S*2*, s*2*, step*2*, obs*2*, d om, A*⟩ to be the system with identical actions and domains, with states *S*2 = *S*×*S*, initial state *s*2 = (*s*0*, s*0), observation function *obs*2 : *D*×*S*2 → (*O*×*O*) given by *obs*2 (*s, t*)= (*obsu*(*s*)*, obsu*(*t*)) for *s, t* ∈ *S*, and transition function *step*2 : *S*2 × *A* → *S*2 given by *step*2((*s*1*, s*2)*, a*)= (*step*(*s*1*, a*)*, step*(*s*2*, a*)) for *a* ∈ *AL* and *step*2((*s*1*, s*2)*, a*) = (*step*(*s*1*, a*)*, s*2) for *a* ∈ *AH* .

0

0

*u*

Note that in every transition, *a* ∈ *AH* is applied only on the left part of each state pair. An easy induction shows that for every sequence of actions *α* ∈ *A*∗, if *s*2 • *α* = (*s, t*) in *M* 2, then in *M* we have *s* = *s*0 • *α* and *t* = *s*0 • *purgeL*(*α*). We therefore obtain the following:

0

Proposition 4.1 *For M* ∈ M*s, we have M* ∈ *NIs iff in M* 2*, for all states* (*s, t*)

*reachable from s*2*, we have that obs*2 ((*s, t*)) = (*o, o*') *implies o* = *o*'*.*

0 *L*

Now *NIs* is reduced to a safety property, which says *M* 2 will never reach a pair of states (*s, t*) on which *L* has a pair of different views. This enables noninterference to be checked using standard model checking technology, for both linear and branching time.

Corollary 4.2 *For M* ∈ M*s, checking M* ∈ *NIs can be done in time* O(|*S*|2 × |*A*|)

*and additional space* O(|*S*|2)*.*

Proof. The system *M* 2 has at most |*S*|2 states. Performing a search algorithm to traverse every possibly reachable state by trying every possible action takes

|*S*|2 × |*A*|. Marking states reached requires space |*S*|2, in addition to the space needed to represent *M* .

Barthe et al. [[2](#_bookmark14)] and Davas et al. [[6](#_bookmark17)] proposed a self-composition technique to

reason about language based noninterference properties, which is somewhat similar to our method for *NIs*. However, their definitions of noninterference are targeted at reasoning about programming languages and assume that a single input is given at the beginning and a single output observed at the end of the computation, whereas we deal with systems permitting an arbitrary sequence of actions to be performed by two distinct agents, and generating outputs throughout the computation.

The property *RESs* can be regarded as a nondeterministic version of *NIs* be- cause they both are characterizable by the existence of an unwinding relation. This property can be characterized using fixpoints as follows. Define the opera- tor *TL* : P(*S* × *S*) → P(*S* × *S*) by (*p, q*) ∈ *TL*(*X*) iff

* (*p, q*) ∈ *X* and *obsL*(*p*)= *obsL*(*q*)
* for all *p*' ∈ *next*(*p, a*) and *a* ∈ *AL* there exists *q*' ∈ *next*(*q, a*) such that (*p*'*, q*') ∈ *X*
* for all *q*' ∈ *next*(*q, a*) and *a* ∈ *AL* there exists *p*' ∈ *next*(*p, a*) such that (*p*'*, q*') ∈

*X*.

The operator *TL* is *monotonic*, in the sense that ∼1⊆∼2 implies *TL*(∼1) ⊆ *TL*(∼2). The set of binary relations on *S* and the subset relation (⊆) have the structure of a complete lattice. The Knaster-Tarski theorem [[27](#_bookmark38)] asserts the exis- tence of least and greatest fixpoints of a monotonic operator on a complete lattice. We write *νX.TL*(*X*) for the greatest binary relation ∼ satisfying ∼⊆ *TL*(∼). The following result characterizes *RESs* in terms of *TL* and the property LR (in Defini- tion [3.8](#_bookmark3)).

Proposition 4.3

1. *M* ∈ M*ns satisﬁes RESs iff there exists an equivalence relation* ∼⊆ *S* × *S*

*satisfying* ∼= *TL*(∼) *and LR.*

1. *M* ∈ M*ns satisﬁes RESs iff νX.TL*(*X*) *satisﬁes LR.*

The understanding of the property *RESs* in Proposition [4.3](#_bookmark7)([ii](#_bookmark8)) yields several algorithmic approaches to its verification. One approach is symbolic, using Binary Decision Diagrams [[5](#_bookmark16)] to relations. The operation *TL* is readily encoded as an operation on BDDs, and the computation of *νX.TL*(*X*) and the verification that it satisfies LR can be implemented using standard operations on BDDs.

We also obtain a bisimulation-based approach. From Definition [3.8](#_bookmark3), an unwind- ing relation is essentially a strong bisimulation relation with respect to *AL*, and Proposition [4.3](#_bookmark7) requires the largest ∼. It is known that computing the largest bisimulation on a labelled transition system can be reduced to the problem of find- ing the coarsest partition, which is computable in O(|*M* |× |*S*|) by Kanellakis and Smolka’s algorithm [[16](#_bookmark32)] and in O(|*M* |× *log*2(|*S*|)) by Paige and Tarjan’s algorithm [[22](#_bookmark33)], where |*M* | is the number of transitions. In our case |*M* | = |*S*|2 × |*AL*|, and we start from an initial partition corresponding to the equivalence relation ≈ defined by *s* ≈ *t* iff *obsL*(*s*)= *obsL*(*t*). To verify LR we need to check every *H* transition, which takes |*S*|2 × |*AH* |. The space requirement is comparable to the size of the system itself in [[22](#_bookmark33)].

Theorem 4.4 *Given M* ∈ M*ns, M in RESs is veriﬁable in* O(|*S*|2*log*2(|*S*|)×|*AL*|+

|*S*|2 × |*AH* |) *time and space* O(|*M* |)*.*

If *M* is deterministic, the size of the transition relation |*M* | becomes |*S*|×|*A*| and the time complexity of the bisimulation algorithm in [[16](#_bookmark32)] reduces to O(|*S*|2 × |*AL*|). Thus, the complexity for the whole procedure becomes O(|*S*|2 × |*AL*| + |*S*|× |*AH*|) time and and space linear in |*M* |. This is marginally better than than the result in Proposition [4.2](#_bookmark6) on time and better on space. However, the reduction to a classical model checking problem in Proposition [4.2](#_bookmark6) permits various optimization techniques to be used (e.g., partial order reductions) so it is unclear which technique will perform better in practice.

We note that another way to approach these results is by a reduction to results of Bossi and Focardi et al. [[4](#_bookmark15),[10](#_bookmark21)] who defined a bisimulation based property *P BNDC* on labelled transition systems (LTS) and proved a polynomial time complexity result for it. We may define a linear time translation *Fsl* : M*ns* → L*IO* from state-observed system into *τ* -free and input-enabled LTS as follows. Assuming *O* = *OH* ∪ *OL*, for *M* = ⟨*S, s*0*, step, obs, dom, A*⟩∈ M*ns*, define *Fsl*(*M* )= ⟨*P, p*0*,* →*,* L⟩ where

.

* 1. *P* = *S*, *p*0 = *s*0
  2. L = *A* ∪ *O*
  3. →= {(*s, a, t*)|∃*a* ∈ *A* : *t* ∈ *step*(*s, a*)}∪ {(*s, o, s*)|∃*o* ∈ *O, u* ∈ *D* : *o* = *obsu*(*s*)}. The following relates *RESs* to a property on labelled transition systems.

Proposition 4.5 *M* ∈ M*ns in RESs iff Fsl*(*M* ) *is in P BNDC.*

It follows that checking *RESs* has a polynomial time upper bound, from [[10](#_bookmark21)]. In particular, [[10](#_bookmark21)]’s algorithm for *P BNDC* reduces a weak bisimulation problem into checking strong bisimulation with an additional step of transitive closure on *AH* . Their algorithm works in O(|→ | × *log*2(|*S*|)) in general, where |→ | is comparable to |*S*|2 × |*A*| in our approach. The complexities of our direct approach and this approach by reduction are therefore essentially equivalent.

* 1. *Trace-Set Properties*

Verifying the remaining properties *NDIs*, *GNs* and *FCs* proves to be more complex than *NIs* and *RESs*. In this section we prove the following:

Theorem 4.6 *For* P *any of NDIs, GNs or FCs, the problem of deciding M* ∈P

*is PSPACE-complete.*

For the lower bound part of this result, we use the following polynomial time reduction to convert the classical problem of deciding if the language accepted by a nondeterministic finite state automaton is equal to Σ∗ into the problem of verifying any of the following: *NDIs*, *GNs* and *FCs*. Let *A* = ⟨*S,* →*,* Σ*, s*0*,* F⟩ be a nonde- terministic finite state automaton (without *ϵ*-transitions) which does not accept *ε*, where *S* is the set of states, Σ the alphabet, → the transition relation, *s*0 the initial

state and F the set of final states. Define *M* (*A*)= ⟨*Sm, s*0*, obs, next, d om, Am*⟩ to be the syst.em with

* *Sm* = *S* ∪.*S*Σ, where *S*Σ = {*s*' *, s*' *, s*' }

0

1

2

* *Am* =Σ ∪ {*h*} with *dom*(*a*)= *L* for all *a* ∈ Σ and *dom*(*h*)= *H*
* *obs* : *D* × *Sm* → {0*,* 1} with *obsH*(*s*) = 0 for all *s* ∈ *Sm* and *obsL*(*s*) = 0 for all

*s* ∈ *S* ∪ {*s*' *, s*' }, and *obsL*(*s*)= 1 if *s* = *s*'

0 2 1

* *next* : *Sm* × *A* → P(*Sm*) defined as follows
* For *a* = *h*: *next*(*s*0*, h*)= {*s*' }, *next*(*s, h*)= {*s*} if *s* /= *s*0

0

* For *a* ∈ Σ: *next*(*s*' *, a*) = *S*Σ, *next*(*s*' *, a*) = {*s*' } and *next*(*s*' *, a*) = {*s*' }; for

0 1 1

'

2 *a* 2

*s* ∈ *S*, *next*(*s, a*)= {*s*2} if there does not exist *t* such that *s* −→ *t*; otherwise,

*next*(*s, a*) = {*t* ∈ *S*|*s* −*a*→ *t*}∪ {*s*' } if {*t* ∈ F|*s* −*a*→ *t*} /= ∅, and *next*(*s, a*) =

*a a* 1

{*t* ∈ *S*|*s* −→ *t*} if {*t* ∈ F|*s* −→ *t*} = ∅.

The construction of *M* (*A*) from *A* can be done in polynomial time.

Proposition 4.7 *Let* P *be any of the properties NDIs, GNs, FCs. Then* L(*A*)= Σ∗\{*ε*} *iff M* (*A*) ∈P

Deciding if the language accepted by a nondeterministic finite state automaton equals Σ∗\{*ε*} with |Σ|≥ 2 is known to be a PSPACE-complete problem [[11](#_bookmark22)], so all the above security properties are PSPACE hard. Next we will show each is solvable in polynomial space.

Lemma 4.8 *M* ∈ M*ns in NDIs iff for every possible low observation β* ∈

*O*+(*AO*+)∗*, there exists a run r* ∈ *S*(*AS*)∗ *such that viewL*(*r*)= *β and ActH* (*r*)= *ϵ.*

Lemma 4.9 *If M* ∈ M*ns in NDIs then for every reachable state s and t* ∈

*next*(*s, a*) *with a* ∈ *AH, we have obsL*(*s*)= *obsL*(*t*)*.*

Lemma [4.8](#_bookmark9) shows a system is in *NDIs* iff *H*’s actions do not cause more obser- vations to *L* than if *H* does nothing. The following definitions sketch a reduction from *NDIs* into a regular language equivalence problem.

Definition 4.10 The *H*-*Condenser* is the function *CondH* : M*ns* → M*ns* de-

fined as follows on a machine *M* ∈ M*ns*. For *s* ∈ *S* let [*s*] = {*t*|∃*α* ∈ *A*∗ :

*H*

*t* is reachable from *s* by *α*}. Define *CondH* (*M* )= ⟨*Sc, s*0*, nextc, obsc, AL*⟩, where

* *Sc* = {[*s*] | *s* ∈ *S*}.
* *nextc* : *Sc* × *AL* → P(*Sc*) such that [*t*] ∈ *nextc*([*s*]*, a*) if there exists *s*' ∈ [*s*], such that *t* ∈ *next*(*s*'*, a*).
* *obsc* : *Sc* → P(*O*) such that *obsc*([*s*]) = {*o* ∈ *O*|∃*s*' ∈ [*s*]: *obsL*(*s*')= *o*}.

Definition 4.11 The *H*-*Restrictor* is the function *RestH* : M*ns* → M*ns* such that for *M* ∈ M*ns*, *RestH* (*M* ) = ⟨*Sr, s*0*, nextr, obsL, AL*⟩, where *Sr* ⊆ *S* is the set of states reachable from *s*0 by actions in *AL* only, and *nextr* : *Sr* × *AL* → *Sr* is the restriction of the ‘*next*’ function to *Sr* × *AL*.

The systems *CondH* (*M* ) and *RestH* (*M* ) can be regarded as Moore machines

with the same input set *AL* and output set *O*. If all the values *obsc*(*s*) are singletons and we shift the outputs on states to their incoming transitions, we get two finite automata on the same alphabet *AL* × *O*.

Proposition 4.12 *M* ∈ *NDIs iff in CondH* (*M* ) *for all s* ∈ *Sc, the set obsc*(*s*) *is a singleton, and CondH* (*M* ) *and RestH* (*M* ) *are language equivalent on AL* × *O.*

Every state in *CondH* (*M* ) having single observation is a necessary condition by Lemma [4.9](#_bookmark10), and this is linearly checkable. The language equivalence between the two regular language is a PSPACE-complete problem [[11](#_bookmark22)]. Both *CondH* (*M* ) and *RestH* (*M* ) have state space at most *S*, so generating them can be done in polynomial time. Thus, *NDIs* is in PSPACE.

This result is related to work of Focardi Gorrieri et al. [[9](#_bookmark20)], who studied the complexity problem of the information flow properties *NNI*, *SNNI*, *NDC*, *BNNI* and *SBNNI* in a process algebraic framework. For input-enabled systems, *NNI, SNNI* and *NDC* are equivalent.

Proposition 4.13 *For M* ∈ M*ns, we have M* ∈ *NDIs iff Fsl*(*M* ) *is in SNNI.*

Focardi et al. give an exponential time subset-construction based algorithm for the property *SNNI*. Our result for *NDIs* states the complexity more precisely, but also yields exponential time in practice, pending advances in complexity theory.

The security property *GNs* requires arbitrary *H* action interleavings to be con- sistent with *L* views, and it is seemingly more complicated than *NDIs*. However, the following analysis yields an in-place exhaustive solution to refute *GNs*.

Define *V iewL* : *A*∗ → P(*O*+(*AO*+)∗) such that an observation *β* ∈ *V iewL*(*α*) if there exists a run *r* with *viewL*(*r*)= *β* and *Act*(*r*)= *α*. Intuitively *V iewL*(*α*) is the set of *L* observations compatible with *α*.

Lemma 4.14 *M* ∈ M*ns is in GNs iff for all α* ∈ *A*∗*, V iewL*(*α*)= *V iewL*(*α*|*L*)*.*

Proof. For the ‘only if’ part, suppose *M* ∈ *GNs*. We need to show *V iewL*(*α*) = *V iewL*(*α*|*L*). For a particular *β* ∈ *V iewL*(*α*), there exists a run *r* with *Act*(*r*)= *α* and *viewL*(*r*) = *β*. From *M* ∈ *GNs* we can delete actions in *AH* from *r* to get a new run *r*' such that *viewL*(*r*')= *β*. If all actions in *AH* are deleted, *Act*(*r*)= *α*|*L*, so *β* ∈ *V iewL*(*α*|*L*). So *V iewL*(*α*) ⊆ *V iewL*(*α*|*L*). *V iewL*(*α*|*L*) ⊆ *V iewL*(*α*) can be proved similarly by inserting actions in *AH* into a run compatible with *L*’s observation in *V iewL*(*α*|*L*). For the ‘if’ part, suppose *V iewL*(*α*) = *V iewL*(*α*|*L*) for all *α* ∈ *A*∗. It follows that *V iewL*(*α*') = *V iewL*(*α*) for any *α, α*' ∈ *A*∗ with *α*'|*L* = *α*|*L*. For a particular run *r* with *Act*(*r*) = *α*, an arbitrary insertion or deletion of actions in *AH* into *α* generates a new action sequence *α*' with *α*'|*L* = *α*|*L*. From *V iewL*(*α*')= *V iewL*(*α*), we have *viewL*(*r*) ∈ *V iewL*(*α*'), so that there exists a run *r*' with *Act*(*r*')= *α*' and *viewL*(*r*')= *viewL*(*r*). So *M* in *GNs*.

From Lemma [4.14](#_bookmark11), we have *M* /∈ *GNs* iff there exists some action sequence *α* ∈ *A*∗ and some *L* observation *β* ∈ *O*+(*AO*+)∗, such that *β* ∈ *V iewL*(*α*) \ *V iewL*(*α*|*L*) or *β* ∈ *V iewL*(*α*|*L*) \ *V iewL*(*α*). This motivates the following algorithm *DecGN* ,

which, given a system *M* , nondeterministically guesses an action sequence *α* and a low observation *β* and checks consistency with *α* and *α*|*L*.

*DecGN* (*M* ):

1. Place a red marker and a blue marker on the initial state.
2. Repeat the next step 22×|*S*| times, where |*S*| is the number of states in *M* .
3. Nondeterministically select *a* ∈ *A*,
   * If *a* ∈ *AH*, then for every *s* with a blue marker, erase the old marker on *s* and place new blue markers on all *t* ∈ *next*(*s, a*). If there exists any states *s, t* marked either red or blue with *obsL*(*s*) /= *obsL*(*t*) return *true*, else proceed.
   * If *a* ∈ *AL*, then nondeterministically choose *o* ∈ *O*. For every *s* with a red marker, erase the old red marker on *s* and place new red markers on every *t* ∈ *next*(*s, a*) with *obsL*(*t*) = *o*. Then do the same on blue markers. After that, if there is only one colour remaining, return *true*, if no colour remaining, return *f alse*, otherwise, proceed.
4. Return *f alse* in the end.

Intuitively, the blue markers are tracing the executions of a possible input action sequence *α* with respect to a particular *L* view, the red markers are tracing the executions of *α* restricted to *L* with respect to the same *L* view. Whenever any *H* action changes *L*’s local view or only one of the sets of executions can follow a step in the *L* view, *GNs* is detected to be false. The number 22×|*S*| covers all possible pairs of sets of marked states. Formally, we claim the following.

Proposition 4.15 *M* /∈ *GNs iff there exists a computation of DecGN* (*M* ) *which returns true.*

Since the algorithm is nondeterministic, *M* /∈ *GNs* is decidable in NPSPACE. Savitch’s theorem states PSPACE = NPSPACE, so deciding *GNs* is in PSPACE. *FCs* can be shown in NPSPACE in a similar procedure as *DecGN* by fixing not only *L*’s observations, but also *H*’s observations.

# Heuristics

We now consider some heuristic approaches which may optimize the verification of the properties we have considered, which work by reducing the problem of verifying a property on a system to a verification on an “equivalent” system.

In particular, we define a relation capturing equivalence on states with respect to *L*’s actions, and consider the use of this to compress the state space of the system. Define an *L*-bisimulation on a system *M* to be an equivalence relation ∼*L*⊆ *S* × *S* such that *s*1 ∼*L s*2 iff

1. *obsL*(*s*1)= *obsL*(*s*2)

2

1. for all *a* ∈ *A* and *s*'

1

∈ *next*(*s*1*, a*) there exists *s*'

∈ *next*(*s*2*, a*) such that

' ∼*L s*' .

*s*

2

1

Write [*s*] for the equivalence class of *s* with respect to ∼*L*.

Let *min* be a function M*ns* → M*ns* such that *min*(*M* ) = ⟨*Sm, sm,*

0

*nextm, obsm, d om, A*⟩, where, with ∼*L* the maximal *L*-bisimulation on *M* ,

* 1. *Sm* = *S/* ∼*L* and *sm* = [*s*0]

0

* 1. For [*s*]*,* [*t*] ∈ *Sm*, *a* ∈ *A*, [*t*] ∈ *next*([*s*]*, a*) if there exists *s*' ∈ [*s*] and *t*' ∈ [*t*] such that *t*' ∈ *next*(*s*'*, a*)
  2. For [*s*] ∈ *Sm*, *obsm*([*s*]) = *o* if *obsL*(*s*)= *o*, and *obsm*([*s*]) = ⊥ for all [*s*] ∈ *Sm*,

*L H*

where ⊥ is any observation on the range of *obs*.

Theorem 5.1 *For M* ∈ M*ns, M* ∈ P *iff min*(*M* ) ∈ P*, where* P *is any of the properties NDIs, GNs, FCs and RESs. In particular, M* ∈ *RESs iff for all s* ∈ *Smin*(*M* )*, a* ∈ *AH, next*(*s, a*)= {*s*}*.*

This result may produce optimizations since the size of *min*(*M* ) may be sig- nificantly smaller than *M* . In general, ‘bisimulation minimization’ is not a vi- able approach for the verification of invariance properties since the partition based bisimulation usually takes more resources than it saves in the subsequent model checking [[7](#_bookmark18)]. However, for properties such as *NDIs*, *GNs* and *FCs*, which seem to unavoidably take exponential time, spending polynomial time on minimization may benefit the verification significantly. Also, on deterministic systems, as described in Proposition [4.1](#_bookmark5), we reduce the *NIs* problem into a safety problem which is verifiable by running a model checker on a larger state space (precisely, from |*S*| to |*S*|2). This extra cost may mean that a prior minimization step is beneficial.

In this case, another consideration may result in further reductions. In state based systems, a state is usually represented as an assignment to a set of variables. Let *s* ∈ *S* be represented as a function *s* : *V* → *U* where *V* is a set of variables and *U* is a universe of values. Let the *L* observation function be represented so that *obsL*(*s*) is the restriction of *s* to a set *VL* ⊆ *V* . For *v* ∈ *VL*, define the function

*obsv* (*s*)= *s*(*v*), and let *Mv* be the system in which *obsL* is replaced by *obsv* . Then

*L L*

we have the following:

Proposition 5.2 *For M* ∈ M*s, we have M* ∈ *NIs iff Mv* ∈ *NIs for all v* ∈ *VL.*

This result suggests an approach where we apply the bisimulation minimization approach to each *Mv* before applying the doubling construction. Each of these systems may be significantly smaller than *min*(*M* ). Approximately, the bisimu- lation algorithm takes O(|*AL*| × |*S*|*log*2(|*S*|)) to get a partition and the further exhaustive search from Theorem [4.2](#_bookmark6) takes O(|*Sr*|2 × |*A*|) where *Sr* is the maximal size quotient state space obtained for the *Mv*. So the whole time complexity is

|*VL*|× O(|*AL*|× |*S*|*log*2(|*S*|)+ |*Sr*|2 × |*A*|). Since |*S*|≈ 2|*V* | if all variables are binary,

|*VL*| is far less than |*S*|. Since observations in *Mv* are based on a single variable, it seems likely that |*Sr*| is far less than |*S*|, and |*Sr*|2 × |*A*| may be far less than

|*S*|2 × |*A*|. Also |*AL*|× |*S*|*log*2(|*S*|) is far less than |*S*|2 × |*A*|. So we conclude it is very likely that |*VL*|× (|*AL*|× |*S*|*log*2(|*S*|)+ |*Sr*|2 × |*A*|) is far less than |*S*|2 × |*A*|. For the space complexity, we are using additional space O(|*Sr*|2) repeatedly in the model checking phase instead of O(|*S*|2). Since space costs are often the critical factor in model checking, this gain may be significant.

# Conclusion

We have considered a number of security properties on state-observed systems and have studied the complexity of the verification problems for all these properties. The unwinding characterizable properties *NIs* and *RESs* are tractable, based on the result of deciding bisimulation on finite states. Both symbolic and explicit state methods are applicable to these properties. The trace based properties (*NDIs*, *GNs*, *FCs*) are PSPACE-complete. Furthermore, we have proposed some heuristics based on bisimulation minimization and argued that they may be effective. We leave the work of implementation and empirical evaluation of this claim for future work. It will be interesting, in particular, to compare the performance of BDD and explicit state model checking approaches with the compositional approaches to noninterference verification of [[9](#_bookmark20)], which are based on process algebraic modelling.

# References

1. Alpern, B. and F. B. Schneider, *Defining liveness*, Information Processing Letters 21 (1985), pp. 181– 185.
2. Barthe, G., P. R. D’Argenio and T. Rezk, *Secure Information Flow by Self-Composition*, in: *17th IEEE* *Computer Security Foundation Workshop* (2004), pp. 100–114.
3. Bevier, W. R. and W. D. Young, *A State-Based Approach to Noninterference*, in: *Proc. 7th Computer* *Security Foundations Workshop*, 1994, pp. 11–21.
4. Bossi, A., R. Focardi, C. Piazza and S. Rossi, *Bisimulation and Unwinding for Verifying Possibilistic* *Security Properties*, in: *Proc. of Int. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI’03)*, 2003.
5. Bryant, R. E., *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers C-35 (1986), pp. 677–691.
6. Darvas, A., R. H¨ahnle and D. Sands, *A Theorem Proving Approach to Analysis of Secure Information Flow*, in: *Workshop on Issues in the Theory of Security, (WITS’03)*, 2003.
7. Fisler, K. and M. Y. Vardi, *Bisimulation and Model Checking*, in: *Conference on Correct Hardware* *Design and Verification Methods (CHARME’99)*, 1999, pp. 338–341.
8. Focardi, R. and R. Gorrieri, *A Classification of Security Properties for Process Algebras*, in: *Journal* *of Computer Security*, 1, IOS Press, 1995 pp. 5–33.
9. Focardi, R. and R. Gorrieri, *The Compositional Security Checker: A Tool for the Verification of* *Information Flow Security Properties*, Technical Report UBLCS-96-14, Universit`a di Bologna (1996).
10. Focardi, R., C. Piazza and S. Rossi, *Proofs Methods for Bisimulation Based Information Flow Security*, in: *Proc. of Int. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI’02)*, 2002, pp. 16–31.
11. Garey, M. R. and D. S. Johnson, “Computers and Intractability — A Guide to the Theory of NP- Completeness,” W.H. Freeman and Company, 1979.
12. Goguen, J. and J. Meseguer, *Security Policies and Security Models*, in: *IEEE Symp. on Security and Privacy*, 1982, pp. 11–20.
13. Goguen, J. and J. Meseguer, *Unwinding and Inference Control*, in: *IEEE Symp. on Security and Privacy*, 1984.
14. Greve, D., M. Wilding and W. van Fleet, *A Separation Kernel Formal Security Policy*, in: *ACL2 Workshop*, 2003.
15. Johnson, D. M. and F. J. Thayer, *Security and the composition of machines*, in: *Proc. IEEE Computer Security Foundations Workshop*, 1988, pp. 72–89.
16. Kanellakis, P. C. and S. A. Smolka, *CCS Expressions, Finite State Processes, and Three Problems of Equivalence*, in: *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, 1983, pp. 228–240.
17. McCullough, D., *Specifications for Multi-Level Security and a Hook-Up Property*, in: *Proc. IEEE Symp.* *on Security and Privacy*, 1987, pp. 161–166.
18. McCullough, D., *Noninterference and the composability of security properties*, in: *Proc. IEEE Symp. on Security and Privacy*, 1988, pp. 177–186.
19. McCullough, D., *A hookup theorem for multi-level security*, IEEE Transactions on Software Engineering

16 (1990), pp. 563–568.

1. McLean, J., *A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions*, in: *Proc. IEEE Symp. on Security and Privacy*, 1994, pp. 79–93.
2. Oheimb, D. v., *Information flow control revisited: Noninfluence = Noninterference + Nonleakage*, in:

*Computer Security – ESORICS’04*, LNCS 3193 (2004), pp. 225–243.

1. Paige, R. and R. Tarjan, *Three Partition Refinement Algorithms*, in: *SIAM Journal of Computing*, 16, 1987, pp. 973–989.
2. Roscoe, A., *CSP and determinism in security modelling*, in: *Proc. IEEE Symp. on Security and Privacy*, 1995, pp. 114–221.
3. Rushby, J., *Proof of Separability – a Verification Technique for a Class of Security Kernels*, in: *Proc. 5th International Symposium on Programming, Turin, Italy*, 1982, pp. 352–367.
4. Rushby, J., *Noninterference, Transitivity, and Channel-Control Security Policies*, Technical report, SRI international (1992).

URL <http://www.csl.sri.com/papers/csl-92-2/>

1. Sutherland, D., *A model of information*, in: *Proc. 9th National Computer Security Conference*, 1986,

pp. 175–183.

1. Tarski, A., *A lattice-theoretical fixpoint theorem and its applications*, in: *Pacific J. Math*, 1955 pp. 285–309.
2. van der Meyden, R. and C. Zhang, *A comparison of semantic models for noninterference*, in: *4th International Workshop in Formal Aspects in Security and Trust (FAST’06)*, 2006, to appear, available at [http://www.cse.unsw.edu.au/˜c](http://www.cse.unsw.edu.au/)zhang/fast.ps.
3. Wittbold, J. T. and D. M. Johnson, *Information flow in nondeterministic systems*, in: *Proc. IEEE Symp. on Security and Privacy*, 1990, pp. 144–161.
4. Zakinthinos, A. and E. Lee, *A general theory of security properties*, in: *Proc. IEEE Symp. on Security and Privacy*, 1997, pp. 94–102.