Absorbing Covers and Intransitive Non-Interference

Sylvan Pinsky
National Security Agency

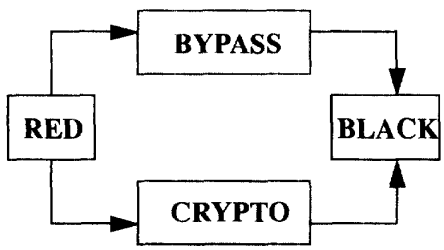
Abstract

This paper gives necessary and sufficient conditions for a system to satisfy intransitive non-interference. Security is defined in terms of allowable flows of information among action domains as represented by an interferes relation ~>. We examine properties of special sets called basis elements generated from the relation ~> and introduce the notion of absorbing covers which is associated with the standard unwinding theorems for non-interference. Our approach separates the equivalence relation arguments from the non-interference properties, and as a by product, we develop a decision procedure for non-interference. An upper bound on the number of iterations needed for termination of the procedure is provided.

1 Introduction

Non-interference was introduced by Goguen and Mesequer [5] to provide a foundation for the specification and analysis of security policies. Although this approach was quite successful in handling multilevel security policies, a number of practical security problems were beyond the scope of their formulation. In particular, Boebert and Kain [1] introduced "type enforcement" and "assured pipelines" as a means to handle specific information flows. The classic example is the "labeler" problem, where the user sends a file that he is allowed to access to the printer application. The system policy requires that the file cannot be printed unless it has gone through the labeler first. Earlier to this effort, Rushby explored the notion of "channel control" [15, 16].

Figure 1 shows two examples of such problems.



END-TO-END ENCRYPTION

CONTROLLED DOWNGRADING

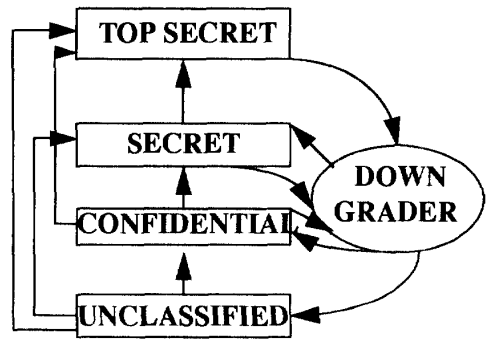


FIGURE 1

In the "controlled downgrader" example, information can flow directly from a lower classification to a higher one; however, information can only flow to a lower level by first passing through a "trusted" downgrader. This problem as well as the labeler example represents an intransitive policy, since secret information **cannot flow directly** to confidential; but can flow from secret to the downgrader and then from the downgrader to confidential. Neither assignment of secret ~> confidential or secret ~\> confidential appropriately describes the required information flow policy.

In end-to-end encryption systems, plaintext messages enter the Red side of the controller. Since network switches need to be able to read header information to perform routing functions, the header goes through the Bypass and arrives at the Black side of the controller. The body of the message is encrypted by the Crypto unit before arriving at the Black side. The Red side must interfere with the Black side, and the challenge is to formulate an intransitive policy which allows interference to occur only through the mediation of the Crypto or the Bypass. The construction of the purge function is at the heart of the problem. Although Goguen and Meseguer extended their work [6] to address these concerns, the first really complete description of intransitive non-interference was done by Haigh and Young [7, 8]. They applied the notion of type enforcement and gave a convincing argument that it was

necessary to consider the complete sequence of actions when forming purge sequences.

Rushby [14] introduced intransitive non-interference as a model for channel-control policies and as a generalization of non-interference and type enforcement. He derived an unwinding theorem for the channel-control case and showed that conventional multilevel policies are a special case of channel-control policies when the interferes relation is transitive. He discussed the weaknesses in earlier formulations and developed a thorough specification and formal verification of his results.

We unify the concepts of standard and intransitive non-interference and present a method which determines when a system satisfies non-interference by examining the security policy and properties of special sets called basis elements generated from the relation ~>. We introduce the notion of absorbing covers which is associated with the standard unwinding theorems for non-interference. Our approach separates the equivalence relation arguments from the non-interference properties, and as a by product, we develop a decision procedure for non-interference. An upper bound on the number of iterations needed for termination of the procedure is provided.

2 System Model

In this section, we describe the underlying framework for formulating non-interference. We unify the concepts of standard and intransitive non-interference through the introduction of a family of purge functions which include both types of non-interference. The notion of basis elements and beta families is described. The section concludes by showing that the existence of a beta-family is a sufficient condition for non-interference to hold.

We assume a standard deterministic state machine and use notation similar to Rushby [14] and Young and Bevier [17]. McLean [12] represents non-interference as an interleave function where purgeable actions are replaced by an action λ_A which causes no state transition and produces no output. Computer scientists have applied the concept of λ_A and stuttering steps to modeling computer systems in CSP [4, 9, 17]. We adopt McLean's approach to purged action sequences.

Definition 1: A ***system*** consists of

a set of ***states***, S
a set of ***actions***, A
a set of ***outputs***, O
a set of ***domains***, D

and functions

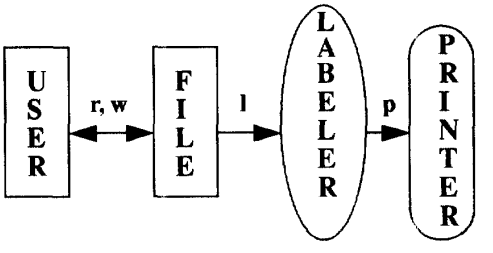
next: S x A → S
out: S x A → O
dom: A → D

We assume that A contains an action λ_A which causes no change in state and O contains an output λ_Φ which produces no data; i.e., $\forall s \in S$, $next(s,\lambda_A) = s$ and $out(s,\lambda_A) = \lambda_\Phi$.

Security is defined in terms of allowable flows of information among action domains as represented by a reflexive relation ~> on D and its complement, denoted by ~\>. A security domain u is non-interfering with domain v if no action performed by u can influence subsequent outputs seen by v. Rushby points out that if u ~\> v, then the requirement of deleting all actions performed by u is too strong if u ~> w and w ~> v. He suggests that only actions of u which are **not followed** by actions of w should be deleted.

Example 1: Consider the labeler example:

A = { r, w, l, p }, D = { {r, w} , {l} , {p} } , where r and



w are the read and write commands and l and p are respectively the actions needed to transfer a file to the labeler and generate the appropriate label, and transfer the labeled file to the printer spool for printing.

The policy is {r, w} ~> {r, w}, {r, w} ~> {l},
{l} ~> {l} , {l} ~> {p} , {p} ~> {p}.

In the following example, ipurge(f,u) denotes the intransitive purge of the sequence f with respect to the domain u:

1). ipurge((r,w,r),{p}) = (λ_A,λ_A,λ_A) since the labeler is not invoked.

2). ipurge((r,w,l,w),{p}) = (r,w,l,λ_A) since the labeler connects r, the first w, and l to the printer.

3). ipurge((w,r,l,p,w,l,w),{p}) = (w,r,l,p,w,l,λ_A) since the last write action was not followed by a labeler request and cannot be sent to the printer.

The purgeability of an action within a sequence requires the examination of the sequence from that point on. An action is purgeable with respect to a domain u only if there is no subsequence of actions beginning at that point consisting of allowable flows which connect to u. The following notation and definitions are used to formalize the concept of an intransitive purge.

For any set X, the set of finite sequences in X is denoted by Seq_X ; in particular,

Seq_A = the set of finite sequences in A .

For $\alpha \in Seq_A$, $||\alpha||$ denotes the number of elements in the sequence; i.e., α: { 1, ... , $||\alpha||$ } $\rightarrow$ A.

We frequently represent α by an n-tuple $(a_1,...,a_n)$, where $n = ||\alpha||$ and $a_i = \alpha(i)$ for $i \in \{1,...,||\alpha||\}$.

For $a \in A$, (a,α) denotes the sequence $(a,a_1,...,a_n)$.

Bool = the set of boolean values: { True, False } .

Definition 2: For γ , $\alpha \in Seq_A$, γ ***is a subsequence of*** α, if each element of γ is in α and the ordering is preserved; i.e.,

subsequence: Seq_A x $Seq_A \rightarrow$ Bool

subsequence(γ,α) =

$\exists$ N : { 1, ... , $||\gamma||$ } $\rightarrow$ { 1, ... , $||\alpha||$ } such that

$\forall$ i , j $\in$ { 1, ... , $||\gamma||$ } , γ(i) = α(N(i))

& i < j $\Rightarrow$ N(i) < N(j) .

Definition 3: A sequence $\alpha \in Seq_A$, ***is an interference path***, if the domains of successive elements of α interfere; i.e.,

interference_path: $Seq_A \rightarrow$ Bool

interference_path(α) =

$\forall$ i < $||\alpha||$, dom(α(i)) ~> dom(α(i+1)) .

Definition 4: For $\alpha \in Seq_A$, $i \in$ { 1, ... , $||\alpha||$ } , $u \in D$, the sequence α ***connects the index i to domain u***, if there exists a subsequence of α initiating at α(i) which forms an interference path and the domain of its last action interferes with u; i.e.,

connects: Seq_A x D $\rightarrow$ Seq_{Bool}

For i $\in$ { 1, ... , $||\alpha||$ }, connects(α,u)(i) =

$\exists$ $\gamma \in Seq_A$ such that subsequence(γ,α), γ(1) = α(i),

interference_path(γ), & dom(γ($||\gamma||$)) ~> u.

We want to construct a purge function that describes both standard and intransitive non-interference. To do so, we first examine properties of the connects function with respect to the interferes relation ~> . We begin by noting that **connects reduces to ~>** when applied to a **singleton action** ; that is, the connects predicate when applied to a single action a, is true **if and only if** dom(a) ~> u. Also, if a sequence cannot connect an action to a domain, that element can't interfere with the domain. More specifically, for $\alpha \in Seq_A$, $i \in$ { 1, ... , $||\alpha||$ }, and $u \in D$, **if α does not connect the index i to u, then dom(α(i)) ~\> u.** The proof of this statement is by contradiction. If dom(α(i)) ~> u, we use the singleton subsequence (α(i)) to violate the assumption that connects(α,u)(i) is false.

Intransitive and standard non-interference are described by:

Intransitive non-interference (π_1)

π_1 : Seq_A x D $\rightarrow$ Seq_{Bool}

π_1 (α,u) = connects(α,u) ; that is,

for i $\in$ { 1, ... , $||\alpha||$ } ,

π_1(α,u)(i) = connects(α,u)(i)

Standard non-interference (π_2)

π_2 : Seq_A x D $\rightarrow$ Seq_{Bool}

For i $\in$ { 1, ... , $||\alpha||$ } ,

π_2(α,u)(i) = dom(α(i)) ~> u .

These two are examples of a mapping π : Seq_A x D $\rightarrow$ Seq_{Bool} which has the property:

$\forall$ $\alpha \in Seq_A$, $u \in D$, and $i \in$ { 1, ... , $||\alpha||$ },

not(π(α,u)(i)) $\Rightarrow$ dom(α(i)) ~\> u .

In considering functions of the form π : $Seq_A \times D \rightarrow Seq_{Bool}$, it is useful to understand the relationship between $\pi((a,\alpha),u)$ and $\pi(\alpha,u)$. Since the (i+1)st element of $(a,a_1,...,a_n) = a_i$, we note the properties:

$\pi_1((a,\alpha),u)(i+1) = \pi_1(\alpha,u)(i)$

(since $connects((a,\alpha),u)(i+1) = connects(\alpha,u)(i)$)

and

$not(\pi_1((a,\alpha),u)(1)) \Rightarrow not(\pi_1((a),u)(1))$

(since $(a,\alpha)(1) = a$) .

π_2 also satisfies these properties since $dom((a,\alpha)(i+1)) = dom(\alpha(i))$. These observations form the basis for defining a family of purge functions which include intransitive non-interference(π_1) and standard non-interference (π_2).

Definition 5: A mapping π : $Seq_A \times D \rightarrow Seq_{Bool}$ is called ***a pi-mapping*** if

$\forall \alpha \in Seq_A$, $u \in D$, the following conditions are satisfied:

a). sequence lengths are invariant:

$\|\pi(\alpha,u)\| = \|\alpha\|$

b). reduce index property:

$\pi((a,\alpha),u)(i+1) = \pi(\alpha,u)(i)$,

$\forall a \in A$, $i \in \{ 1, ... , \|\alpha\| \}$

c). singleton constraint:

$not(\pi((a,\alpha),u)(1)) \Rightarrow not(\pi((a),u)(1))$, $\forall a \in A$.

Definition 6: ***The purge function for a pi-mapping*** π , denoted by $purge_\pi$, is the mapping:

$purge_\pi$: $Seq_A \times D \rightarrow Seq_A$ given by

$\forall i \in \{ 1, ... , \|\alpha\| \}$,

$$purge_\pi(\alpha,u)(i) = \begin{cases} \alpha(i) , & \text{if } \pi(\alpha,u)(i) \\ \lambda_A , & \text{otherwise .} \end{cases}$$

Rushby [14] purges actions based on a function called sources that serves the same purpose as the connects function. Except for the distinction that we replace purgeable actions with λ_A and he deletes them, the formulations are equivalent. He points out that intransitive non-interference is a generalization of standard non-interference since the intransitive purge reduces to the Gougen & Meseguer purge function when the policy ~> is transitive. To understand this, consider the interference path $(a_1,...,a_n)$ when $n > 1$. Since $a_1 \sim> a_2$, $a_2 \sim> a_3$, ..., $a_{n-1} \sim> a_n$, we apply the transitivity of ~> n-1 times to conclude that $a_1 \sim> a_n$.

An output from the system as defined by the out function requires both a state and an action (as does the next function), so we maintain a coupling of the state reached by applying a sequence of actions to an initial state and an action by defining the two projection functions state and action and the updating function step. Several authors [7, 14, 17] produce this state by defining a function run, and then add an action to formulate non-interference. We choose to bundle the state and action via the function state_action. These functions are given by:

state: $S \times A \rightarrow S$ action: $S \times A \rightarrow A$

$state(s,a) = s$ $action(s,a) = a$

step: $(S \times A) \times A \rightarrow S \times A$

$step(z,a) = (next(state(z),a),action(z))$

state_action: $(S \times A) \times Seq_A \rightarrow S \times A$

$state\_action(z,\alpha) = (next^*(state(z),\alpha),action(z))$,

where next* is the usual extension of next to a sequence of actions.

Security is defined in terms of the pi-mapping π and the purge function. For all action domains, we require that the system produces the same output after processing a sequence of actions as it does when processing the purged actions.

Definition 7: A system ***satisfies non-interference with respect to the pi-mapping*** π, if

$\forall z \in S \times A$, $\alpha \in Seq_A$

$out(state\_action(z,\alpha)) =$

$out(state\_action(z,purge_\pi(\alpha,dom(action(z)))))$.

In characterizing when a system satisfies non-interference, it is useful to consider those state-action pairs that share a common output value:

view: $S \times A \rightarrow \mathcal{P}(S \times A)$ ($\mathcal{P}$ denotes the power set)

$view(z) = \{ v \in S \times A \mid out(v) = out(z) \}$.

Our analysis begins with the examination of singleton action sequences and observing that a necessary condition for non-interference is that step(z,a) ∈ view(z) for purgeable actions a.

Remark 1: For the pi-mapping π, a necessary condition for non-interference is that $\forall z \in S \times A$ and $a \in A$,

$$not(\pi((a),dom(action(z)))(1)) \Rightarrow \mathbf{step(z,a) \in view(z)}.$$

Proof: If non-interference with respect to π is to hold, we require $out(state\_action(z,(a))) = out(state\_action(z,purge_\pi((a),dom(action(z))))$. By definition, $purge_\pi((a),dom(action(z))) = (\lambda_A)$ and we note that $step(z,\lambda_A) = (next(state(z),\lambda_A),action(z)) = (state(z),action(z)) = z$. Therefore, $\mathbf{out(step(z,a))} = out(state\_action(z,(a))) = out(state\_action(z,purge_\pi((a),dom(action(z))))) = out(state\_action(z,(\lambda_A))) = out(step(z,\lambda_A)) = \mathbf{out(z)}$.

This important property is the motivation for defining basis elements and establishing that non-interference reduces to finding an appropriate grouping of the basis elements. The desired characteristics of the grouping are given in the definition of a beta-family.

Definition 8: For the pi-mapping π and $z \in S \times A$, the ***basis element for z with respect to*** π is given by $basis_\pi(z) = \{ z \} \cup \{ step(z,a) \mid a \in A$ and $\pi((a),dom(action(z)))(1) = False \}$.

Definition 9: The ***action image*** is the mapping:

$$T: \mathcal{P}(SxA) \times A \rightarrow \mathcal{P}(SxA)$$

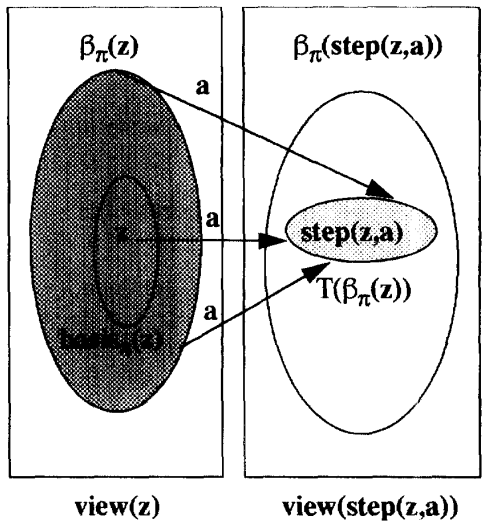
$$T(B,a) = \{ step(z,a) \mid z \in B \}.$$

Definition 10: For the pi-mapping π, a mapping $\beta_\pi: S \times A \rightarrow \mathcal{P}(SxA)$ is a ***beta-family with respect to*** π if $\forall y, z \in S \times A$ and $a \in A$:

Property 1. $basis_\pi(z) \subseteq \beta_\pi(z) \subseteq view(z)$

Property 2. $T(\beta_\pi(z),a) \subseteq \beta_\pi(step(z,a))$

Property 3. $\beta_\pi(y) \cap \beta_\pi(z) \neq \emptyset \Rightarrow \beta_\pi(y) = \beta_\pi(z)$ (β_π sets are disjoint).



Pinsky [13] showed that the existence of a beta-family is a sufficient condition for standard non-interference to hold. We will use the following results to establish the equivalence of non-interference and the existence of beta-families.

Lemma 1: If β_π is a beta-family with respect to the pi-mapping π and $z \in S \times A$, then $z \in \beta_\pi(z)$.

Proof: If follows from the definition of $basis_\pi$ and Property 1 of a beta-family that $z \in basis_\pi(z) \subseteq \beta_\pi(z)$.

Lemma 2: Suppose that β_π is a beta-family for the pi-mapping π, $y, z \in S \times A$, and $a \in A$ such that $y \in \beta_\pi(z)$ and $\pi((a),dom(action(z)))(1) = False$, then $\mathbf{step(y,a) \in \beta_\pi(z)}$.

Proof:

If follows from the definition of $basis_\pi$ and Property 1 that $step(z,a) \in basis_\pi(z) \subseteq \beta_\pi(z)$. Since $step(z,a) \in \beta_\pi(step(z,a))$ (from Lemma 1), we have that $step(z,a) \in \beta_\pi(z) \cap \beta_\pi(step(z,a))$ and $\beta_\pi(step(z,a)) = \beta_\pi(z)$ (from the disjoint property for beta-families). Property 2 is used to conclude that $step(y,a) \in T(\beta_\pi(z),a) \subseteq \beta_\pi(step(z,a)) = \beta_\pi(z)$.

The next lemma establishes the central property that is used in Theorem 1 to prove that the existence of a beta-family implies that non-interference holds.

Lemma 3: Suppose that β_π is a beta-family with respect to the pi-mapping π, then

$\forall$ x, y $\in$ S x A and $\alpha \in Seq_A$:

$x \in \beta_\pi(y) \Rightarrow$

state_action(x,α) $\in$

β_π(state_action(y,purge$_\pi$(α,dom(action(y))))) .

Proof: By induction on ||α|| using Lemma 2.

Base Case: ||α|| = 1, α = (a), a $\in$ A

From Property 2 of a beta-family, state_action(x,α) = step(x,a) $\in$ T(β_π(y),a) $\subseteq$ β_π(step(y,a)).

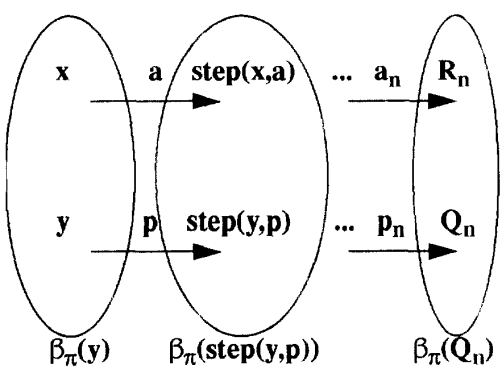
If π(α,dom(action(y)))(1) = True, then purge$_\pi$((a),dom(action(y))) = (a); therefore, state_action(x,α) $\in$ β_π(step(y,a)) = β_π(state_action(y,(a))) = β_π(state_action(y,purge$_\pi$(α,dom(action(y))))).

On the other hand, if π(α,dom(action(y)))(1) = False, then p = λ_A and step(y,a) $\in$ β_π(y) (from Lemma 1 and Lemma 2). Also, y = (next(y,λ_A),action(y)) = step(y,λ_A) = state_action(y,(λ_A)) = state_action(y,purge$_\pi$(α,dom(action(y)))) ; therefore state_action(x,α) $\in$ β_π(step(y,a)) = β_π(y) = β_π(state_action(y,purge$_\pi$(α,dom(action(y))))).

Induction Step: ||α|| = n , α = (a_1,...,a_n) and show $\forall$ x, y $\in$ S x A , a $\in$ A:

$x \in \beta_\pi(y) \Rightarrow$ state_action(x,(a,α)) $\in$

β_π(state_action(y,purge$_\pi$((a,α),dom(action(y))))) .



Induction Hypothesis

step(x,a) $\in$ β_π(step(y,p)) $\Rightarrow$ $R_n \in \beta_\pi(Q_n)$ where

p = purge$_\pi$((a,α),dom(action(y)))(1)

R_n = state_action(step(x,a),α)

Q_n = state_action(step(y,p),purge$_\pi$(α,dom(action(y)))) .

We show that step(x,a) $\in$ β_π(step(y,p)). If purge$_\pi$((a,α),dom(action(z)))(1) = True, then p = a and apply Property 2 of a beta-family to obtain step(x,a) $\in$ β_π(step(y,a)) = β_π(step(y,p)). If purge$_\pi$((a,α),dom(action(z)))(1) = False, then p = λ_A and by Lemma 2, step(x,a) $\in$ β_π(y) and since y = step(y, λ_A) = step(y,p), we have step(x,a) $\in$ β_π(y) = β_π(step(y,p)) .

The state-action and purge functions have the property that $\forall$ z $\in$ S x A , a, b $\in$ A, α, γ $\in$ Seq_A , u $\in$ D: state-action(z,(b,γ)) = state-action(step(z,b),γ) and purge$_\pi$((a,α),u) = (p,purge$_\pi$(α,u)) where p = purge$_\pi$((a,α),u)(1).

We apply the induction hypothesis to conclude: state_action(x,(a,α)) = state_action(step(x,a),α) = $R_n \in \beta_\pi(Q_n)$ = β_π(state_action(step(y,p),purge$_\pi$(α,dom(action(y))))) = β_π(state_action(y,purge$_\pi$((a,α),dom(action(y))))).

Theorem 1: If π is a pi-mapping and there exists a beta family with respect to π , then non-interference with respect to π holds.

Proof: Suppose that β_π is a beta-family with respect to π, z $\in$ S x A , and $\alpha \in Seq_A$, then Lemma 1 and Lemma 3 yields state_action(z,α) $\in$ β_π(state_action(z,purge$_\pi$(α,dom(action(z))))) and by Property 1,β_π(state_action(z,purge$_\pi$(α,dom(action(z))))) $\subseteq$ view(state_action(z,purge$_\pi$(α,dom(action(z))))). Therefore, out(state_action(z,α)) = out(state_action(z,purge$_\pi$(α,dom(action(z))))) .

3 Equivalence Relations and Minimum Cover Relations

This section establishes the equivalence of non-interference and the existence of beta-families. We prove the converse of Theorem 1, namely that if non-interference for a pi-mapping π holds, then there exists a beta-family with respect to π. Our approach separates the equivalence relation arguments from the non-interference properties. Other authors [2, 3, 8, 14] construct equivalence relations for each subject and prove that these equivalence relations satisfy unwinding conditions. We introduce the notion of an absorbing cover which deals with the unwinding condition stated in Property 2 of a beta-family. Our equivalence classes are generated automatically as a natural consequence of the non-interference properties and minimum cover relations developed in this section. We first present some general results on equivalence relations which apply to this problem.

We remind the reader that an equivalence relation $\sim$ on the set X is a binary relation on X which satisfies the properties:

For a,b,c $\in$ X,

1. $a \sim a$	(reflexive)
2. $a \sim b \Rightarrow b \sim a$	(symmetric)
3. $a \sim b \ \& \ b \sim c \Rightarrow a \sim c$	(transitive) .

The relation $\sim$ defines a subset of X x X, where $a \sim b$ is the standard notation for denoting that (a,b) belongs to the relation, and the equivalence class of x, denoted by $[x]_\sim$, is the set $\{ y \in X \mid x \sim y \}$. The set of distinct equivalence classes partitions X into disjoint sets.

Definition 11: A collection C of subsets of X ***is a cover for X*** if for every $x \in X$, there is an element of C which contains x; i.e.,

$$\text{cover}: \mathcal{P}(\mathcal{P}(X)) \rightarrow \text{bool}$$

$$\text{cover}(C) = \forall x \in X, \quad \exists B \subseteq X \text{ such that } x \in B \text{ and } B \in C.$$

For the remainder of this section, we assume that C is a cover for X.

Definition 12: For $x \in X$ and the cover C, the ***cover set for x***, denoted by $U_C(x)$, is the union of all elements of C which contain x: i. e.,

$$U_C: X \rightarrow \mathcal{P}(X) \setminus \{ \emptyset \}$$

$$U_C(x) = \bigcup_{\substack{B \in C \\ x \in B}} B \quad (U_C(x) \neq \emptyset \text{ since C is a cover}).$$

We use U_C to build an equivalence relation on X, where x and y are related if there is a sequence α of elements in X, initiating at x and ending at y where the cover sets corresponding to successive elements of α intersect; i.e. ,

$$\sim C: X \times X \rightarrow \text{bool}$$

$$\sim C(x,y) =$$

$$\exists \ \alpha \in Seq_X \text{ such that } \alpha(1) = x \ , \ \alpha(\|\alpha\|) = y \ ,$$

$$\text{and } \forall \ i < \|\alpha\| \ , \ U_C(\alpha(i)) \cap U_C(\alpha(i+1)) \neq \emptyset \ .$$

Lemma 4: $\sim C$ is an equivalence relation.

Proof:

a). $\sim C$ is reflexive

For $x \in X$, the sequence (x,x) satisfies the definition for $\sim C(x,x)$ (also denoted, $x \sim C \ x$).

b). $\sim C$ is symmetric

For x, $y \in X$, with $\sim C(x,y)$ and corresponding sequence $\alpha = (a_1,....,a_n)$, we reverse the sequence to form $(a_n,....,a_1)$ establishing $\sim C(y,x)$.

c). $\sim C$ is transitive

Suppose x, y, $z \in X$, with $\alpha = (a_1,....,a_n)$ verifying $\sim C(x,y)$ and $\beta = (b_1,....,b_r)$ verifying $\sim C(y,z)$. We merge the two sequences to form $\gamma = (a_1,....,a_n,b_1,....,b_r)$ and note that $a_n = y = b_1$ and γ is the required sequence for verifying that x is related to z under the relation $\sim C$.

Remark 2: The equivalence relation $\sim C$ has the property that $\forall \ x \in X, \ U_C(x) \subseteq [x]_{\sim C}$ since if $y \in U_C(x)$, then the sequence (y,x) verifies $\sim C(y,x)$ because $y \in U_C(y) \cap U_C(x)$. The next lemma shows that $\sim C$ is the **"minimal"** equivalence relation that has the property that every equivalence class contains the cover set for each of its members.

Lemma 5:

Let $E_C = \{ \sim \mid \sim \text{ is an equivalence relation on X with the property, } \forall x \in X, \ U_C(x) \subseteq [x]_\sim \}$, then $\forall x \in X$, $[x]_{\sim C} = \bigcap_{\sim \in E_C} [x]_\sim$.

Proof:

Suppose $\sim \in E_C$ and $y \in [x]_{\sim C}$. Then there exists $\alpha = (a_1,...,a_n)$ such that $\alpha(1) = x$, $\alpha(n) = y$, and $\forall i < n$, $U_C(\alpha(i)) \cap U_C(\alpha(i+1)) \neq \emptyset$.

a). $[x]_{\sim C} \subseteq \bigcap_{\sim \in E_C} [x]_\sim$

For $i < n$, $[\alpha(i)]_\sim \cap [\alpha(i+1)]_\sim \neq \emptyset$ since $U_C(\alpha(i)) \cap U_C(\alpha(i+1)) \subseteq [\alpha(i)]_\sim \cap [\alpha(i+1)]_\sim$. Hence $[\alpha(i)]_\sim = [\alpha(i+1)]_\sim$ (equivalence classes are disjoint) and $y \in [\alpha(n)]_\sim = ... = [\alpha(1)]_\sim = [x]_\sim$. Since y is an arbitrary element of $[x]_{\sim C}$ and $\sim$ is an arbitrary element of E_C, we conclude that

$[x]_{\sim C} \subseteq \bigcap_{\sim \in E_C} [x]_\sim$

b) $\bigcap_{\sim \in E_C} [x]_\sim \subseteq [x]_{\sim C}$ since $\sim C \in E_C$ (from Remark 2).

We now apply these results to the non-interference problem. There is a natural cover for S x A consisting of basis elements and sets obtained by applying arbitrary action sequences to basis elements. For $B \in \mathcal{P}(S \times A)$ and $\alpha \in Seq_A$, let

$$SA(B,\alpha) = \{ \text{state\_action}(z,\alpha) \mid z \in B \}.$$

Definition 13: *The **non-interference cover for S x A with respect to the pi-mapping*** π, denoted by ni_cover_π, is the set

$$\mathbf{ni\_cover_\pi} = \{ \mathbf{basis_\pi(z)} \mid z \in S \times A \} \cup \{ \mathbf{SA(basis_\pi(z),\alpha)} \mid z \in S \times A, \alpha \in \mathbf{Seq_A} \}.$$

Properties of beta-families involve basis elements, views, and the application of actions to sets generated from basis elements. The cover ni_cover_π has the properties:

1). $\forall z \in S \times A$, $basis_\pi(z) \in ni\_cover_\pi$
2). $\forall B \in ni\_cover_\pi, a \in A$, $T(B,a) \in ni\_cover_\pi$.

The first property holds by the definition of ni_cover_π. To understand why the second property holds, first consider $z \in S \times A$ and note that $T(basis_\pi(z),a) = \{ step(x,a) \mid x \in basis_\pi(z) \} = \{ state\_action(x,(a)) \mid x \in basis_\pi(z) \} = SA(basis_\pi(z),(a)) \in ni\_cover_\pi$. Secondly, if $\alpha = (a_1,...,a_n) \in Seq_A$, we form the sequence $\gamma = (a_1,...,a_n,a)$ and note that $T(SA(basis_\pi(z),\alpha),a) = \{ step(state\_action(x,\alpha),a) \mid x \in basis_\pi(z) \} = \{ state\_action(x,\gamma) \mid x \in basis_\pi(z) \} = SA(basis_\pi(z),\gamma) \in ni\_cover_\pi$.

The property $T(B,a) \in ni\_cover_\pi$ is related to Property 2 of a beta-family. The following definitions will be used in constructing beta-families.

Definition 14: A cover C, ***covers all basis sets with respect to the pi-mapping*** π, if $\forall z \in S \times A$, there exists a set $B \in C$ such that $basis_\pi(z) \subseteq B$.

Definition 15: A cover C, ***absorbs actions***, if $\forall B \in C$ and $a \in A$, there exists $D \in C$ such that $T(B,a) \subseteq D$.

Definition 16: A cover C, ***contains views***, if $\forall B \in C$, there exists $z \in S \times A$ such that $B \subseteq view(z)$.

Definition 17: For the cover C, ***the beta-operator for C***, denoted by $\beta\text{-operator}_C$, associates each element of S x A with its minimum cover equivalence class; i.e.,

$$\beta\text{-operator}_C : S \times A \rightarrow \mathcal{P}(S \times A) \setminus \{ \emptyset \}$$
$$\beta\text{-operator}_C(z) = [z]_{\sim C}.$$

Lemma 6 relates absorbing covers to Property 2 of beta-families and Theorem 2 completes the equivalence of non-interference and the existence of beta-families.

Lemma 6: If the cover C absorbs actions, then $\forall z \in S \times A, a \in A$, $T([z]_{\sim C},a) \subseteq [step(z,a)]_{\sim C}$.

Proof:

Suppose that $y, z \in S \times A$ and $a \in A$ such that $y \in [z]_{\sim C}$. For some positive integer n and $i \in \{ 1, ..., n \}$, let $x_i \in S \times A$, $C_i, D_i \in C$ such that $z = x_1$, $y = x_n$, $x_i \in C_i$, $T(C_i,a) \subseteq D_i$, and for $i < n$, $v_i \in C_i \cap C_{i+1}$. The existence of n, $\{C_1, D_1, ..., C_n, D_n\}$, $\{x_1, ..., x_n\}$, and $\{v_1, ..., v_{n-1}\}$ are guaranteed by the definition of $\sim C$ and the assumption that C absorbs actions. The sequence $(step(x_1,a), ..., step(x_n,a))$ verifies that $step(y,a) \in [step(z,a)]_{\sim C}$ since $step(x_i,a) \in D_i$ for $i \in \{ 1, ..., n \}$ and for $i < n$, $D_i \cap D_{i+1} \neq \emptyset$ because $step(v_i,a) \in T(C_i,a) \cap T(C_{i+1},a) \subseteq D_i \cap D_{i+1}$.

Theorem 2: If π is a pi-mapping and C is a cover for S x A which covers all basis elements with respect to π, contains views, and absorbs actions, then β-operator$_C$ is a beta-family with respect to π.

Proof:

We first show that $\forall z \in S \times A$, $[z]_{\sim C} \subseteq view(z)$. Let $y \in [z]_{\sim C}$ with $x_1, \ldots, x_n \in A$, $C_1,\ldots,C_n \in C$ such that $\forall i \in \{1, \ldots, n\}$, $x_i \in C_i$, and for $i < n$, $C_i \cap C_{i+1} \neq \emptyset$. Since C contains views, there exists $b_1, \ldots, b_n \in S \times A$ such that $\forall i \in \{1, \ldots, n\}$, $C_i \subseteq view(b_i)$. Now $z = x_1 \in C_1 \subseteq view(b_1)$. Then $out(z) = out(b_1)$ and $view(b_1) = \{ w \in S \times A \mid out(w) = out(b_1) \} = \{ w \in S \times A \mid out(w) = out(z) \} = view(z)$. Since for $i < n$, $C_i \cap C_{i+1} \neq \emptyset$, the same argument shows that $view(b_i) = view(b_{i+1})$. Therefore, $view(z) = view(b_1) = \ldots = view(b_n) = view(y)$, and since y is a arbitrary element of $[z]_{\sim C}$, we conclude that $[z]_{\sim C} \subseteq view(z)$.

We apply this result to show that the three properties of a beta-family with respect to π hold:

1. Since $basis_\pi(z) \subseteq B$ for some $B \in C$, we apply Remark 2 and the definition of U_C to obtain $z \in basis_\pi(z) \subseteq B \subseteq U_C(z) \subseteq [z]_{\sim C} \subseteq view(z)$.

2. Since C absorbs actions, we apply Lemma 6 to show that Property 2 holds; i.e., $T(\beta\text{-operator}_C(z),a) = T([z]_{\sim C},a) \subseteq [step(z,a)]_{\sim C} = \beta\text{-operator}_C(step(z,a))$.

3. Property 3 holds since equivalence classes of any equivalence relation (~C in particular) are disjoint.

Corollary: If π is a pi-mapping and non-interference with respect to π holds, then there exists a beta-family with respect to π.

Proof: Let C be any cover for S x A which covers all basis elements with respect to π, contains views, and absorbs actions. Ni_cover_π is one such cover (from remarks and the discussion above and the observation that $basis_\pi(z) \subseteq view(z)$ and $SA(basis_\pi(z),\alpha) \subseteq view(state\_action(z,\alpha))$, $\forall z \in S \times A$, $\alpha \in Seq_A$ whenever non-interference is satisfied). Apply Theorem 2 to conclude that β-operator$_C$ is a beta-family with respect to π.

4 A Decision Procedure

All the sets described in this paper become finite when implementing computer systems. In this section, we formulate an algorithm for determining if non-interference holds, based on Theorem 2. We start with the set of all basis elements and aggregate them to form a partition of S x A based on applying actions. This process is repeated until either a new set does not have a constant view or an absorbing cover is obtained. We present an example before formalizing the decision procedure.

Example 2: This system, devised by Jon Haugsand [11], does not satisfy non-interference.

$S = \{ S_0, S_1, S_2, S_3, S_4 \}$, $A = \{ lo, high, \lambda_A \}$,
$D = \{ \{lo\}, \{high\}, \{\lambda_A\} \}$, $O = \{ O_1, O_2, \lambda_\Phi \}$,
$\{lo\} \sim\!> \{high\}$ and $\{high\} \sim\!\!\!\!\!\!\diagdown\!> \{lo\}$:

next	lo	high	λ_A	out	lo	high	λ_A
S_0	S_0	S_1	S_0	S_0	O_1	O_1	λ_Φ
S_1	S_2	S_1	S_1	S_1	O_1	O_1	λ_Φ
S_2	S_3	S_2	S_2	S_2	O_1	O_2	λ_Φ
S_3	S_4	S_3	S_3	S_3	O_1	O_2	λ_Φ
S_4	S_4	S_4	S_4	S_4	O_2	O_2	λ_Φ

Computations:

1. basis elements for standard non-interference

a). $basis_{\pi_2}((S_0,lo)) = \{ (S_0,lo), (S_1,lo) \}$
b). For $s \in \{ S_1, S_2, S_3, S_4 \}$, $basis_{\pi_2}((s,lo)) = \{ (s,lo) \}$
c). For $s \in S$, $basis_{\pi_2}((s,high)) = \{ (s,high) \}$
d). For $s \in S$, $basis_{\pi_2}((s,\lambda_A)) = \{ (s,\lambda_A) \}$.

Step 1: Initial partition

$P_0 = \{ \{(S_0,lo), (S_1,lo)\}, \{(S_2,lo)\}, \{(S_3,lo)\}, \{(S_4,lo)\},$
$\{(S_0,high)\}, \{(S_1,high)\}, \{(S_2,high)\},$
$\{(S_3,high)\}, \{(S_4,high)\},$
$\{(S_0,\lambda_A)\}, \{(S_1,\lambda_A)\}, \{(S_2,\lambda_A)\}, \{(S_3,\lambda_A)\},$
$\{(S_4,\lambda_A)\} \}$.

Step 2: For $B \in P_0$, compute T(B,lo) and union the elements of P_0 which contain elements of T(B,lo).

a). $T(\{ (S_0,lo), (S_1,lo) \},lo) = \{ (S_0,lo), (S_2,lo) \}$; therefore $\{ (S_0,lo), (S_1,lo) \} \cup \{ (S_2,lo) \}$.

$P_1 = \{ \{ (S_0,lo), (S_1,lo), (S_2,lo) \}, \{(S_3,lo)\}, \{(S_4,lo)\}, \{(S_0,high)\}, \{(S_1,high)\}, \{(S_2,high)\}, \{(S_3,high)\}, \{(S_4,high)\}, \{(S_0,\lambda_A)\}, \{(S_1,\lambda_A)\}, \{(S_2,\lambda_A)\}, \{(S_3,\lambda_A)\}, \{(S_4,\lambda_A)\} \}$.

b). repeat Step 1 for $T(\{ (S_0,lo), (S_1,lo), (S_2,lo) \},lo)$ to obtain:

$P_2 = \{ \{ (S_0,lo), (S_1,lo), (S_2,lo), (S_3,lo) \}, \{(S_4,lo)\}, \{(S_0,high)\}, \{(S_1,high)\}, \{(S_2,high)\}, \{(S_3,high)\}, \{(S_4,high)\}, \{(S_0,\lambda_A)\}, \{(S_1,\lambda_A)\}, \{(S_2,\lambda_A)\}, \{(S_3,\lambda_A)\}, \{(S_4,\lambda_A)\} \}$.

c). repeat Step 1 for
$T(\{ (S_0,lo), (S_1,lo), (S_2,lo), (S_3,lo) \},lo)$ to obtain:

$P_3 = \{ \{ (S_0,lo) , (S_1,lo) , (S_2,lo) , (S_3,lo) , (S_4,lo) \}, \{(S_0,high)\}, \{(S_1,high)\}, \{(S_2,high)\}, \{(S_3,high)\}, \{(S_4,high)\}, \{(S_0,\lambda_A)\}, \{(S_1,\lambda_A)\}, \{(S_2,\lambda_A)\}, \{(S_3,\lambda_A)\}, \{(S_4,\lambda_A)\} \}$.

Step 3: Check the view for the set $\{ (S_0,lo), (S_1,lo), (S_2,lo), (S_3,lo), (S_4,lo) \}$. This set does not have a constant view since $out((S_0,lo)) = O_1$ and $out((S_4,lo)) = O_2$. The algorithm terminates with non-interference failing. We confirm this result with the state-action pair (S_0,lo) and action sequence (high,lo,lo,lo):

$out(state\_pair((S_0,lo),(high,lo,lo,lo))) = out(S_4,lo) = O_2$

and

$out(state\_pair((S_0,lo),ipurge((high,lo,lo,lo),\{lo\}))) =$
$out(state\_pair((S_0,lo),(\lambda_A,lo,lo,lo))) =$
$out(state\_pair((next(S_0,\lambda_A),lo),(lo,lo,lo))) = out(S_0,lo) = O_1$.

The following notation and functions are used to formalize the decision procedure:

Let $Z = \mathcal{P}(S \times A)$ and $A = \{ a_1, \ldots , ||A|| \}$.

Z_intersect: $Z \times \mathcal{P}(Z) \rightarrow Z$
$Z\_intersect(V,B) = \bigcup_{D \in B,\ V \cap D \neq \emptyset} D$

Z_disjoint: $Z \times \mathcal{P}(Z) \rightarrow \mathcal{P}(Z)$
$Z\_disjoint(V,B) = \{ D \in B \mid V \cap D = \emptyset \}$.

A cover for S x A can be converted into disjoint sets by repeatedly applying Z_intersect. For $C = \{ C_1, \ldots , C_n \}$, start with $B_1 = Z\_intersect(C_1,C)$. Let $E = Z\_disjoint(C_1,C)$, and if $E \neq \emptyset$, then $E = \{ E_1, \ldots , E_r \}$ and compute $B_2 = Z\_disjoint(E_1,E)$. Since $C_1 \notin E$, E is a subset of C with fewer elements, and we continue the process until $\emptyset$ is reached.

Upd: $Z \times \mathcal{P}(Z) \rightarrow \mathcal{P}(Z)$
$Upd(V,B) = Z\_intersect(V,B) \cup Z\_disjoint(V,B)$.

For the partition $P = \{ P_1, \ldots , P_{||P|} \}$, $||P|| > 1$, and $a \in A$:
Form $\{ T(P_1,a), \ldots, T(P_{||P||},a) \}$ and update as follows:
$U_1(a) = Upd(T(P_1,a),P)$
$U_2(a) = Upd(T(P_2,a),U_1(a))$
...
$U_{||P||}(a) = Upd(T(P_{||P||},a),U_{||P||-1}(a))$.

The final partition is captured in the function update_action:

update_action: $\mathcal{P}(Z) \times A \rightarrow \mathcal{P}(Z)$
$update\_action(P,a) = U_{||P||}(a)$.

We repeat the update process for all actions using the recursive function:

update_all_actions: $Z \times \{ 1, \ldots , ||A|| \} \rightarrow Z$

$update\_all\_actions(P,1) = update\_action(P,a_1)$.
$\forall\ i < ||A||,\ update\_all\_actions(P,i+1) = update\_action(update\_all\_actions(P,i),a_{i+1})$.

Updating all actions in A is given by:

update: $\mathcal{P}(Z) \rightarrow \mathcal{P}(Z)$
$update(P) = update\_all\_actions(P,||A||)$.

The function fixed_point produces a cover which absorbs actions:

fixed_point: $\mathcal{P}(Z) \rightarrow \mathcal{P}(Z)$

$$fixed\_point(P) = \begin{cases} P, & \text{if } update(P) = P \\ fixed\_point(update(P)), & \text{if } update(P) \neq P . \end{cases}$$

Let P_0 be the partition of basis elements, then $fixed\_point(P_0)$ computes a **partition of S x A** which ***"absorbs actions"***. This is accomplished by using the function update to union the elements of P_0 which intersect sets of the form $\{ T(U,a) \mid a \in A \}$ where $U \in P_0$ and $a \in A$. If $update(P_0) = P_0$, then P_0 ***"absorbs actions"***; otherwise, we let $P_1 = update(P_0)$ and note

that $||P_1|| \leq ||P_0|| - 1$ since at least two elements of P_0 have been merged by update. The partition update(P_1) is checked for equality with P_1. The function fixed_point continues this process and terminates either by finding P_k such that update(P_k) = P_k or reaches the partition that unions all the elements of P_0 ; namely { S x A }.

The maximum number of calls to Upd is

$$||P_0|| \frac{(||P_0||+1)}{2} ||A||$$

since

$$||P_0|| ||A|| + (||P_0|| - 1) ||A|| + ... + 1 ||A||$$

$$= \sum_{i=0}^{||P_0||-1} (||P_0||-i) ||A|| = ||P_0|| \frac{(||P_0||+1)}{2} ||A|| .$$

A decision procedure for non-interference has been developed. This immediately raises the question of its applicability to system designs. How large does $||P_0||$ or $||A||$ have to be before the computations become unwieldy or how well does the procedure scale for larger systems? We do not know the answers to these questions; however, we are automating the algorithms and plan to examine these issues.

5 Conclusion

Although intransitive policies require a much more complex purge function than standard non-interference, our unified approach uses a method to determine when a system satisfies non-interference which has the complexity of the standard case. The system designer can demonstrate that his system satisfies non-interference either by a). satisfying the conditions of Theorem 2, or b). executing the decision procedure, or c). proving that the unwinding theorems hold.

The key to non-interference with respect to a pi-mapping π is captured by the properties of the set of basis elements $\{basis_\pi(z) \mid z \in S \times A\}$. Establishing non-interference reduces to finding an appropriate grouping of the basis elements. From Theorem 2, this occurs if there exists a cover for S x A which covers all basis elements, contains views, and absorbs actions. We have presented an algorithm that determines if such a cover exists, whenever the underlying sets are finite. We begin with the cover consisting of only basis elements and successively apply actions in A to update the cover. This process is continued until there is no change in the cover ; hence producing a cover which absorbs actions. If the resulting cover also contains views, then non-interference holds and the algorithm has produced a beta-family. Checks at intermediate stages of the algorithm could be made to determine if the updated covers satisfy the contains view property.

Whenever the algorithm terminates with a cover which contains views and absorbs actions, the "minimal" beta-family is obtained. By starting with the disjoint collection of all basis elements, we produce the set of equivalence classes corresponding to the minimal equivalence relation that has the property that every equivalence class contains the cover sets of each of its members.

6 Acknowledgments

We are indebted to Will Harkness for conveying the "mathematical essence" of non-interference by generalizing the Haigh-Young View-Identical problem to arbitrary finite sets [10]. We benefited from conversations with Tim Redmond, Jon Millen, and Josh Guttman. We especially thank Bill Young, Tom Haigh, and John Rushby for explaining the subtleties of intransitive purges. We also thank anonymous referees for their suggestions.

References

[1] E. Boebert and R. Kain, A practical alternative to hierarchical integrity policies. In Proceedings of the Computer Security Initiative Conference, 1985.

[2] T. Fine, T. Haigh, D. O'Brien, and D. Toups, Noninterference and Unwinding for LOCK. In Proceedings of the Computer Security Foundations Workshop II, 1989.

[3] T. Fine, Constructively Using Noninterference to Analyze Systems. In Proceedings of the IEEE Symposium on Security and Privacy, 1990.

[4] R. Focardi and R. Gorrieri, A Taxonomy of Trace-Based Security Properties for CCS. In Proceedings of the Computer Security Foundations Workshop VII, 1994.

[5] J. Goguen and J. Meseguer, Security policies and security models. In Proceedings of the IEEE Symposium on Security and Privacy, 1982.

[6] J. Goguen and J. Meseguer, Inference Control and unwinding. In Proceedings of the IEEE Symposium on Security and Privacy, 1984.

[7] J. Haigh and W. Young, Extending the non-interference model of MLS for SAT. In Proceedings of the IEEE Symposium on Security and Privacy, 1986.

[8] J. Haigh and W. Young, Extending the noninterference version of MLS for SAT. In IEEE Transactions on Software Engineering, SE-13(2), 1987.

[9] C. A. R. Hoar, Communicating Sequential Processes. Apprentice-Hall International. London, 1985.

[10] W. Harkness, The General LOCK Model and Unwinding Theorem. R21 Informal Technical Report, Department of Defense, October 1990.

[11] J. Haugsand, Private communication, University of Solo, Norway, 1994.

[12] J. McLean, A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In Proceedings of the IEEE Symposium on Security and Privacy, 1994.

[13] S. Pinsky, An Algebraic Approach to Non-Interference. In Proceedings of the Computer Security Foundations Workshop V, 1992.

[14] J. Rushby, Noninterference, Transitivity, and Channel-Control Security Policies. SRI International, CSL Technical Report, 1992.

[15] J. Rushby, The design and verification of secure systems. In 8th ACM Symposium on Operating System Principles, ACM Operating System Review, Val 15, No. 5, 1981.

[16] J. Rushby, Verification of secure systems. Technical Report 166, Computing Laboratory, University of Unchaste upon Tone, UK, 1981.

[17] W. Young and W. Bevier, A State-Based Approach to Noninterference. In Proceedings of the Computer Security Foundations Workshop VII, 1994.