On The Composition Of Security Properties

by

Aris Zakinthinos

A thesis submitted in conformity with the requirements for
the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

This thesis presents a general theory of system composition for possibilistic security properties. It is shown that possibilistic security properties can be viewed as a predicate over the traces that are consistent with a low level observation τ_{low}. We provide a uniform framework for analyzing and comparing these properties. We demonstrate how to determine what security property a system satisfies given the security properties satisfied by its constituent components. Also, we show how to construct a system that satisfies a desired security property. This analysis yields a condition that can be used to determine how a property may emerge under composition. We examine the reasons for the failure of feedback composition and provide necessary and sufficient conditions for determining when feedback composition will fail for all properties based on Generalized Noninterference. Unwinding theorems are given for a large class of security properties.

Acknowledgments

A large number of persons were important in the development of this work. First and foremost, I would like to thank my supervisor, Professor E. S. Lee, for his support, encouragement and patient guidance throughout my graduate studies.

My university friends who helped make being a graduate student fun.

I would like to acknowledge the support that I have received from the Natural Sciences and Engineering Research Council of Canada (NSERC) during the course of my graduate studies.

My parents who without their support, dedication and love I would not be here today.

Last, but certainly not least, I would like to thank my wife Keri. Her understanding and love have gotten me through the rough times. It is to her I dedicate this work.

Table Of Contents

List of Figures

List of Definitions

Glossary of Symbols

Notation	Meaning	Page
HLU	A user with high level clearance.	17
LLU	A user with low level clearance.	17
LI	Low Level Input Events.	18
HI	High Level Input Events.	18
LO	Low Level Output Events,	18
HO	High Level Output Events.	18
C	Communication or Internal Events.	19

Traces

Notation	Meaning	Example	Page
<>	the empty trace		15
<a>	a trace containing only a		15
<a,b,c>	a trace with three symbols		15
^	trace concatenation	<a,b>^<>^<c> = <a,b,c>	15
t|*A*	*t* restricted to events in the set *A*	<b,c,d,a>|{a,c} = <c,a>	15
Λt	alphabet of a trace *t*	Λ< a_1, a_2, a_1, a_3, a_2> = {a_1, a_2, a_3}	15
A^*	set of traces with elements in *A*	A^*={s | s|*A* = s }	16
interleave(t_1,t_2)	interleaving of the bunch of traces t_1 and t_2		16
$trace_s(\tau)$	is τ a trace of system S		17
trace(S)	bunch of traces of system S		17
$B_{low}(\tau,S)$	bunch of traces that are consistent with observing τ by the low level user	§s:traces(S)·τ|L=s|L	27
ε	Null Event	p^< ε >^s = p^s	31
$G_p(\tau,S)$	bunch of traces property *P* guarantees are consistent with the observation of τ by the LLU		34
S\α	Removes the event α from the System S		57

Logic

Notation	Meaning	Example
$=$	equals	$x=x$
$\neq$	is not equal to	$x\neq x+1$
$P\wedge Q$	P and Q are both true	$x\geq x-1\wedge x\leq x+1$
$P\vee Q$	one or both are true	$y\leq x \vee y\geq x$
$\neg P$	P is not true	$\neg 6>8$
$P\Rightarrow Q$	if P then Q	$x\leq y\Rightarrow x<y$
$\forall x:A\cdot P$	for all x in bunch A P is true	
$\exists x:A\cdot P$	there exists a x in bunch A such that P is true	
$\S x:A\cdot P$	those x in A such that P is true	

Sets and Bunches

Notation	Meaning	Example
$\in$	is a member of	$a\in\{a,b,c,d\}$
$A \cup B$	A union B	$A\cup B=\{x\|x\in A\vee x\in B\}$
$A \cap B$	A intersect B	$A\cap B=\{x\|x\in A\wedge x\in B\}$
$A\backslash B$	Set restriction	$A\backslash B=\{x\|x\in A\wedge\neg x\in B\}$
:	is a member of	a : a,b,c,d
A, B	A union B	a,b,c,d

1. Introduction and Overview

Security is a game in which the final goal is never quite in reach.
Laurence Martin *(b. 1928)*
British author, academic

1.1. Introduction

Computers have proliferated throughout every aspect of today's society. Our reliance on computers to maintain and store everything from our nation's secrets to the number of items an individual purchases is growing rapidly. Information is becoming the single most important commodity. It is being bought and sold to the highest bidder. Information about individuals, corporations and governments must not be allowed to be inappropriately disclosed, maliciously altered, or destroyed. The threats that must be protected against are diverse, such as the disgruntled employee, the corporate spy, even hackers whose only desire is to inflict hardship on unsuspecting people. A means for controlling these and all other attacks is required. Without one the damage to society could be immense. To provide adequate security it must be possible to construct secure computer systems.

The ability to derive the properties of an assembly of components from the properties of its individual constituents is central to being able to design secure computer systems. A system for which this capability exists is said to be a composable system. Unfortunately, an understanding of how to construct such secure computer systems has been unavailable. The goal of this work is to provide the necessary foundations so that secure computer systems can be built.

The remainder of this introductory chapter will provide some background information and state the thesis of the work.

1.2. Security Properties and Systems

The goals of computer security are easily stated; prevent users from accessing or acquiring information they are not authorized to access or acquire. Realizing this goal has been difficult. To accomplish this goal systems have to be constructed that satisfy security properties. Security properties specify the types of acceptable behaviours of a system. There has been much work in developing formal models of information flow in the hopes of understanding how to prevent these flows. Despite the advances made to date a complete theory of information flow in secure systems is still missing.

1.3. Composability

For many years engineers have been designing systems using standard, pre-designed components to achieve an economy of design effort and multiple other benefits. The system is composed out of components that have been designed individually and separately. This means that the designer need not start with a blank sheet every time a system is to be composed, because a repertoire of components exists that can be incorporated into a new system.

Unfortunately for secure systems this type of system design has not been possible. The critical missing technology is the ability to create a composite trustworthy system, using as components a heterogeneous collection of existing products [Hemenway & Gambel91]. The following quotation from John McLean clearly states the problem:

> A general ability to build composite high-assurance systems presupposes a general theory of system composition. Such a theory provides insight into why certain properties are preserved or not perserved by certain forms of compostion. More importantly, for a large class of properties and a variety of composition constructs, it answers questions of the form: "if a system satisfying property X is composed with a system satisfying property Y using composition construct Z, what properties will the composite system satisfy?" [McLean94]

In this work we examine security properties under composition and propose a solution to the problem stated by McLean.

1.4. This Thesis

This thesis is concerned with a theory of the composition of components into systems such that the information flow properties of the composed system can be predicted from the information flow properties of the components and from the nature of the interconnections. As can be seen from the review in Chapter 2, the problem has a considerable history that has not always resulted in a useful outcome. Our objective has been to present a new view of this problem that develops new insight into the control of information flows by composed systems.

The thesis of this work is that we can predict the security property satisfied by a composed system from those of its constituent components. We determine conditions for a property to emerge under composition. We also demonstrate for a large class of security properties how to determine if a state machine representation of the system satisfies the security property.

1.5. Overview

Chapter 2 provides a brief review of the relevant related research, including security modeling, trusted systems and composability.

Chapter 3 introduces the model of the systems that we will be considering in this work. We also formally introduce the notion of composition of components and other notation that will be used throughout the work.

In Chapter 4 we define security properties. We investigate the nature of security properties and provide a method of specifying security properties. We also demonstrate how to compare security properties. Finally, we demonstrate that the current approaches to composition of systems are not applicable to security properties.

Chapter 5 presents our theory of secure composition. We investigate compositions with and without feedback. We also provide a means of determining under what conditions a property may emerge under composition. That is, when a composed system satisfies a property that not all of its constituent parts satisfy.

Chapter 6 compares our work and results to the other major work on the composition of secure components, McLean's Selective Interleaving Functions (SIF).

Chapter 7 presents a method of determining if a non-deterministic state machine satisfies a particular property. We will show that the class of properties for which we can provide such a result encompasses most of the security properties presented in the literature.

Finally, Chapter 8 summarizes the work and provides a number of conclusions. It also gives a number of directions for future work in the area.

2. Previous Work

> No great man lives in vain. The history of the world is but the biography of great men.
>
> ***Thomas Carlyle*** *(1795-1881)*
> *Scottish writer*

2.1. Introduction

In this chapter we introduce some of the related research in the area of composability and formal modeling of systems. The emphasis will be placed on the research upon which this thesis was directly based.

2.2. Event Systems

The purpose of our work is to predict the effects of interconnecting systems. These systems will be modeled using discrete event systems. The idea of a discrete event system is to describe the possible observed behaviour of the system rather than the way it works. At the heart of all event system models is the concept of a trace. A trace is the temporally ordered series of events that represent one possible execution of the system. We defer a formal introduction to event systems until Chapter 3.

2.3. Confidentiality Models

We present some of the previous work in confidentiality. Confidentiality is the property that information should not be made available or disclosed to an unauthorized user. This work is not intended to be an exposition of security properties. Therefore, we will only present some of the relevant previous work and defer introducing the security properties until they are needed.

2.3.1. Lattice Approaches to Security

Bell and LaPadula [Bell & LaPadula75] introduced a model of security policies for military systems. Their model (BLP) is one of the earliest successful treatments of confidentiality, and was subsequently the basis of U.S. Department of Defense criteria to be discussed in the next section.

The BLP model is based upon requirements for access controls. Bell and LaPadula formalized their model by the axioms of *simple security* and the **-property* (star property). These axioms define which subjects (active entities) are permitted to read which objects (passive, data-storage entities).

Simple security states that a process may not have read access to any object unless the security class of the object dominates that of the process. The *-property states that a process may not write to an object unless the security class of the object is dominated by that of the process.

Denning [Denning76] extended the Bell and LaPadula model by pointing out that the classification of subjects and objects form a lattice of security levels. The two properties of the BLP model have straightforward extensions to the lattice model.

Since the introduction of these models, several shortcomings have been noted in the literature [McCullough87] [McCullough88]. These systems required trusted subjects to perform vital functions of the system. Also, not all information can be easily described using the object representation. The simple notions of read and write operations do not adequately represent the often complex behaviour that occurs in real systems.

2.3.2. Formal Criteria

The formal criteria are the security policies by which security of systems are established and verified. The issue of composability has not been addressed by any of the formal criteria release by the United States government, the European Community or the Canadian government. The *Canadian Trusted Computer Product Evaluation Criteria* [CTCPEC], released in January 1993 mentions the issue of composability. It indicated that the state of current research has not yielded adequate advances to include any composability requirements in the criteria.

> "Efforts have begun to work out methods of evaluation based on composable products. As research continues, composable evaluation of properly defined composable products will enter the mainstream from the research arena. Composable products and evaluation would allow Vendors to modify existing trusted products and ratina or improve their ratings without having the *entire* product re-evaluated" [CTCPEC]

A detailed discussion of the underlying requirements is not warranted here. However, the production of these criteria has had a significant input on the development of security policy and practice in various sectors.

2.3.3. Possibilistic Security Properties

In 1982, Goguen and Meseguer introduced the notion of noninterference [Goguen & Meseguer82] as the basis for confidentiality. They proposed the following definition:

> "One group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see." [Goguen & Meseguer82]

Noninterference was the first possibilistic security property. The idea behind all possibilistic properties is that if information of a given security level interferes with information of a different security level, the interference should be attributable to more than one possible cause.

Numerous other security properties have been proposed. Each new proposed security property was an attempt to correct a flaw with previous models. Some of the more popular properties are Generalized Noninterference [McCullough87], Restrictiveness [McCullough88], Noninference [O'Halloran90] and n-Forward Correctability [Johnson & Thayer88]. These properties will be examined in detail in this section 4.4.

2.3.4. Sutherland's Deducibility

In 1986, David Sutherland [Sutherland86] took a different approach to solving the security modeling problem. Sutherland attempted to quantify what it means for information to flow from one user to another. Each distinct execution of the system can be considered an element of the set of *possible* worlds. A piece of information about the

system is represented by an *information function* whose domain is the set of possible worlds. Sutherland then defines information flows in terms of these concepts:

> "Given a set of possible worlds Ω and two [information] functions f_1 and f_2 with domain Ω, we say that information flows from f_1 to f_2 if and only if there exists some possible world ω and some element z in the range of f_2 such that z is achieved by f_2 in some possible world, but in every possible world ω' such that $f_1(\omega') = f_1(\omega)$, $f_2(\omega') \neq z$." [Sutherland86]

Information flows from f_1 to f_2 if knowing the value of f_1 rules out (or eliminates from consideration) even a single possible value of f_2. Sutherland has an intended interpretation for f_1 and f_2. This interpretation is not relevant for our work.

Sutherland's definition implies that information flow is a symmetric relationship. If there is a flow from low level users to high level users then there must exist a flow from high level users to low level users. This is an undesirable implication because security is an asymmetric relation; High level to low level information flows are not allowed while low level to high level information flows are.

Another consequence of the definition of information flow is that security is dependent on all high level activity that is consistent with each possible low level observation. It will be demonstrated in section 4.3 that this requirement can be met in an insecure system.

2.4. Composability

The composition of two components can be thought of as interconnecting the components in some fashion. There have been numerous formalisms and frameworks for reasoning about composition. In section 4.7 we demonstrate that most of these are not applicable to security properties. In this section we review the relevant work in the area of composing secure systems.

2.4.1. Hook-up Security

Until 1987 not much work went into examining the effects of interconnecting secure components. McCullough demonstrated that Bell and LaPadula's access controls and its extensions, Sutherland's Deducibility model, and Goguen and Meseguer's

Noninterference requirement are inadequate as composable security properties. It was possible to interconnect components that satisfied these properties in such a way that the resulting system did not satisfy the property. McCullough introduced the idea of *hook-up security* as a solution to this problem. McCullough proposed the following circular definition:

> "A system is hook-up secure if it is deducibility secure and if, when it is hooked up with a second hook-up secure system, the result is a hook-up secure composite system" [McCullough87]

Millen [Millen90] argued that "hookup safety is not just a frill" and is "an essential property of a definition of information security."

McCullough also proposed a new definition of security he called Restrictiveness [McCullough88]. Restrictiveness had the desirable property of composability. Unfortunately, Restrictiveness is not a natural way of expressing security [Rushby91].

2.4.2. Safety and Liveness

In 1985, Alpern and Schneider [Alpern & Schneider85] proposed a formal definition for properties. They express properties as a combination of a *safety property* and a *liveness property*. Informally, a liveness property stipulates that "something" must eventually happen during the execution of a system, while a safety property requires that "something bad" never happens.

In the Alpern and Schneider model a property and a system are both sets of traces. A property holds for a system if and only if the set of traces exhibited by the system is a subset of the set of traces of the property.

The safety/liveness model has some appealing consequences. Since properties are a set of traces every property is the intersection of a safety property and a liveness property. Also, the notion of refinement is very intuitive. Unfortunately, we will demonstrate that this model of properties is not applicable to secure systems.

2.4.3. Composing Specifications

Abadi and Lamport [Abadi & Lamport90] defined a modular specification method for composition. Their formalism is based on an extension of the Alpern and Schneider

notion of safety and liveness properties. Their extension includes a notion of an environment in which the system is going to operate. Their goal is to provide a means to "prove that a composite system satisfies its specification if all its components satisfy their specification" [Abadi & Lamport 90].

The behaviour of components is represented by an infinite sequence of states and "agents" that cause the changes of states. Specifications are a set of behaviours. In Abadi and Lamport's framework the first step is to define the behaviour of the system and then compose the systems. There is also an assumption about the behaviour of the environment. The behaviour of the environment must be an assumption because the environment cannot be controlled. The Composition Principle put forward in this work applies to specifications of the form $E \Rightarrow M$, where E is a property and M is a machine property. This expression asserts that the system will satisfy property M as long as the environment satisfies the property E.

Abadi and Lamport define a composition principle and a set of proof rules. These proof rules can be used to prove that components behave correctly when their environments behave correctly. The proof rules and composition principle are not relevant to this work because in section 4.7 we demonstrate that the Alpern and Schneider safety/liveness framework cannot be used to specify security properties. Therefore, the Abadi and Lamport composition principle cannot be applied to secure systems.

2.4.4. Selective Interleaving Functions

One of the first theories to attempt to address the composition of security properties was presented by McLean in "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions" [McLean94]. McLean noted that security properties do not fall within the Alpern and Schneider categorization of properties and therefore cannot be handled by the Abadi-Lamport Composition Principle (see also section 4.7).

McLean defines a Selective Interleaving Function that is used to define security properties. This function is defined such that, given two traces τ_1 and τ_2, it will produce a third trace $f(\tau_1, \tau_2) \rightarrow \tau$. The trace, τ, is an interleaving of the two given traces. A

component satisfies a security property if it is closed under *f*. Different interleaving functions can be used to generate traces satisfying different possibilistic properties. For example, one can define *f* such that *high_inputs*(τ) = *high_inputs*(τ_1), *low_inputs*(τ) = *low_inputs*(τ_2), and *low_outputs*(τ) = *low_outputs*(τ_2). This interleaving function defines Generalized Noninterference.

A complete discussion of Selective Interleaving Functions is given in Chapter 6.

2.5. Bunch Theory

Bunch theory is not itself a method to model systems. Bunch theory will be used throughout our formalism. In this section we present an introduction to bunch theory. A complete description of bunch theory can be found in "A Practical Theory of Programming" [Hehner93].

A bunch is an unpackaged collection of objects. Contrast this with a set, a packaged collection of objects. A bunch is the contents of a set. This point might seem trivial, but it is essential to presenting a consistent theory. An elementary bunch, or element, is any number, character, string, etc. For example, the number 2 is an elementary bunch, so is the character 'c' and the string 'abba'. In this work the most common elementary bunch will be a trace of a system. The axioms of bunch theory that are relevant to this work are:

If *A* and *B* are bunches, then

A, *B* "*A* union *B*"

is a bunch, and

A:*B* "*A* is included in *B*"

is **true** iff all the elements of *A* are included in *B*.

An important bunch is the empty bunch. That is a bunch with no elements. This will be expressed as *null*. The *null* bunch satisfies the following identity property,

A, *null* = *A*

For a complete list of the axioms of Bunch Theory see "A Practical Theory of Programming" [Hehner93].

Quantifiers will be use throughout this document. The quantifiers ∀ and ∃ have their standard meaning but will be augmented by including with each variable of quantification the bunch of elements that quantification is over. For example,

$\forall i{:}int{\cdot}\exists r{:}rat{\cdot}$ i<=r<=(i+1)

says for all integers there exists a rational number between it and the next higher integer.

The solution quantifier § ("solution of") gives the bunch of solutions of a predicate. For example,

$(\S i{:}int{\cdot}\ i^2 = 4) = -2, 2$ "those *i* in *int* such that …"

The axioms for § are (*v* is a name, *A* and *B* and bunches, *b* is a Boolean expression and *x* is an element):

$(\S v{:}null{\cdot}$b$) =$ null

$(\S v{:}x{\cdot}$b$) =$ **if** b(*x*) **then** *x* **else** *null*

$(\S v{:}A, B{\cdot}$b$) = (\S v{:}A{\cdot}$b$), (\S v{:}B{\cdot}$b$)$

2.6. Unwinding Theorems

Most of the security properties presented in the literature have been trace based. That is, the security condition is expressed over the set of traces of the system. However, most formal specification approaches are based on a state transition model and specify individual state transitions. A theorem stating the equivalence between a trace based security condition and a transition based security condition is called an unwinding theorem.

All of the unwinding theorems presented in the literature have dealt with specific security properties [Goguen & Meseguer84] [McCullough90] [Bevier & Young94] [Millen94]. The specific details of these are not important at this time. Generally, an unwinding theorem takes the following form. Given a system and a sensitivity level, an equivalence relation is imposed on the system states. Then a condition is given on how users of different sensitivity levels can move from equivalence class to equivalence class. For example the unwinding theorem for Noninterference can informally be expressed as follows:

> "Two states are equivalent if they are indistinguisable through system output to a user at level *s* or below, either now or after futher inputs. The noninterference policy is satisfied if and only if high-level inputs have no apparent effect on a low user's view, because they cause transitions to states in the same equivalence set." [Millen94]

2.7. Summary

In this chapter we have presented some of the relevant research. In future chapters we will build upon this work to provide a general framework for the analysis of security properties.

3. Components and Systems

> Mathematics possesses not only truth, but supreme beauty – a beauty cold and austere, like that of sculpture.
>
> ***Bertrand Russel*** *(1872-1970)*
> *British philosopher, mathematician*

3.1. Introduction

A discrete event system is a dynamic system that evolves according to the occurrence of physical events. The system modeler first decides which events are important to the system being modeled. These events correspond to the primitive actions done to or done by the system. Then the modeler describes the interaction between these events. Once the system has been adequately described, predictions about expected behaviour can be made and analyzed.

Example 3.1: In modeling a communication network the relevant events might be:

{ packet_sent, packet_lost, packet_received, time_out }

and the trace

$$s = <a_1, a_2, a_3>$$

may represent the behaviour

packet_sent, packet_lost, time_out ❑

To say that $s=< a_1, a_2, a_3,...>$ is a trace means that a_1 is the first event and the event a_{i+1} occurs after event a_i. The trace contains no information about the real time at which an event occurs. We may say that the trace describes only the logical behaviour. This relativistic timing notion corresponds closely to the practical operation of most real computing systems.

An event trace places no constraints on the absolute timing of particular events. All executions with the same relative ordering of events are captured and represented by a single event sequence. This is represented graphically in Figure 3.1. In this case, the labeled circles represent different events, and their absolute timing displayed on the scale

from left to right. The first two timing sequences have identical event trace representations <A,B,C>, while the third sequence differs.

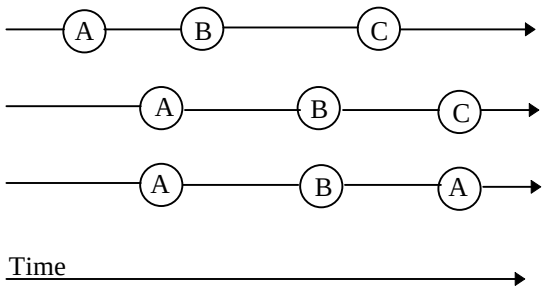


Figure 3.1: Relativistic Timing of Event Traces [Nestor93]
The first two sequences have identical event trace representations, while the third sequence differs.

3.2. Traces

As described in the previous section a trace of a system indicates the behaviour of a process up to some moment in time. A trace will be denoted by an ordered bunch of events enclosed in angular brackets:

<x,y> is a trace of two events. The event x followed by the event y.

<x> is the trace containing only the event x.

<> is the empty sequence containing no events.

Traces play a central role in our theory of composition. We will need operations on traces.

Definition 3.1: Trace Concatenation
The notation s^t will refer to the trace formed by putting together traces s and t in that order. We will use st to denote concatenation if s and t are obvious from the context. Formally, if X and Y are an order bunch of events then <X>^<Y> = <X,Y>

The expression t|A denotes the sub-trace of t containing only events in A.

Example 3.2: Let $t = \langle a_1, a_2, a_1, a_3, a_2 \rangle$. Then $t|\{a_1,a_3\} = \langle a_1,a_1,a_3 \rangle$

The operator Λ will return the set of events present in a bunch.

Example 3.3: Let $t = \langle a_1, a_2, a_1, a_3, a_2 \rangle$. Then $\Lambda t=\{a_1, a_2, a_3\}$

If A is a set of events then A^* is the set of all finite traces (including <>) which are formed from symbols in the set *A*. The following axioms exactly define this set.

1. $<>\in A^*$
2. $<x>\in A^*$ if and only if $x\in A$
3. $(s\hat{}t)\in A^*$ if and only if $s\in A^* \wedge t\in A^*$

It will be required to determine all the possible interleavings of two traces. A trace *t* is an interleaving of two traces *s* and *u* if it can be split into a series of subsequences, with alternate subsequences extracted from *s* and *u*. [Hoare85].

Definition 3.2: Interleave of two bunches of traces

The interleave of two bunches *s, t* written *interleave(s,t)* is defined as:

$$\S r:(\Lambda s \cup \Lambda t)^* \cdot r|\Lambda s:s \wedge r|\Lambda t:t$$

Recall that a single trace is also a bunch. Therefore, if the arguments to interleave are single traces then the resulting bunch will contain all interleavings of the traces.

3.3. Discrete Event Systems

The framework for our investigation into composability will be event systems as given by McCullough [McCullough87] and Johnson and Thayer [Johnson and Thayer88]. McCullough's definition derives from the work on modeling concurrency of Hoare [Hoare85]. We will define a discrete event system as follows:

Definition 3.3: Event Systems.

An event trace system is a 4-tuple:

$$S = <E,I,O,T>$$

where

E is the set of events
I, the input events, $I \subseteq E$
O, the output events, $O \subseteq E$ and $I \cap O = \varnothing$
$T \subseteq E^*$ is the set of traces

The set of events corresponds to the primitive actions done to or by the system. The set of traces of an event system must satisfy the following property. It must always be possible for the system to accept an input event. This condition is called *input totality*

[McCullough87a] [Johnson & Thayer88]. This modeling abstraction simplifies the proof of the theorems. The need for input totality is examined in section 5.3.2.

In this work we draw no inference from the likelihood that certain members of *T* are more probable than others; we are interested only in possible traces. The definition of an event system does not include a means to generate the set *T*. In this work it is not important to have a means to generate *T*. However, it must be remembered that the sequences in *T* do have some order. There are conditions for when events can occur and what conditions are effected after the occurrence of an event.

Since we have augmented all quantification with a bunch to quantify over we define the following:

Definition 3.4: Bunch Notation For the Set of Traces
For a system *S* the function *traces(S)* returns a bunch such that exactly all traces of *S* are elements of the bunch.

Definition 3.5: Verifying a trace of a System
The predicate $trace_s(\tau)$ is true if and only if τ is a trace of system S.

The standard set operators of union, $\cup$, and intersection, $\cap$, will be used to combine the various sets of the event trace system. The set difference operator, \, will also be used. The set A\B, for example, contains all elements in the set A that are not in the set B.

The specification of security properties usually requires a distinction between high level (trusted) and low-level (untrusted[1]) users. We will refer to these categories as HLU and LLU respectively. This division is accomplished by dividing *E* into the disjoint subset *L* and *H*, such that every event is in exactly one of *L* or *H*. These are, respectively, the sets of low- and high-level events. Assuming two comparable levels simplifies the presentation of the results without altering the results; the generalization to an arbitrary lattice of levels is straightforward but notationaly cumbersome.

---

[1] Or less trusted.

The following definition gives some notation for commonly used classes of events.

Definition 3.6: Event Classes.

The following notation will be useful in specifying security properties:

HI	$= H \cap I$	high level input events,
LI	$= L \cap I$	low level input events,
HO	$= H \cap O$	high level output events and
LO	$= L \cap O$	low level output events.

Throughout this document *component* and *system* will be used interchangeably. The following convention will be used when discussing composition. Components will be interconnected to yield systems. It is equally valid to interconnect systems to yield larger systems. However, we feel that by explicitly referring to the sub parts of a system as components emphasizes that they are part of a larger system.

3.4. Composition

It is considered good engineering practice to build large and complex systems from smaller independently verified components. This leads to cheaper and better designed systems. Figure 3.2 demonstrates a model of a system consisting of two CPUs and a bus arbitrator controlling access to shared memory. It is easier for each component to be independently designed and verified then interconnected, rather than attempting to design it as one monolithic system.

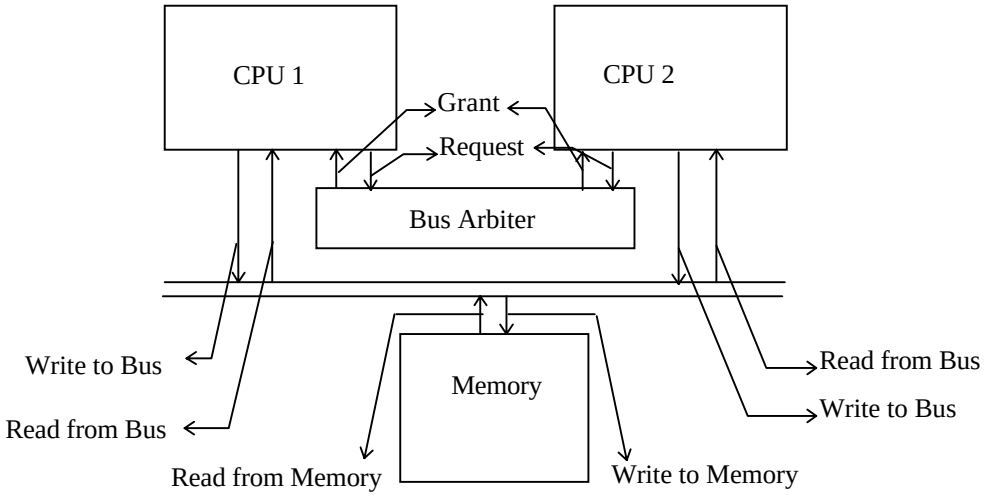


Figure 3.2: An Example of a Composed System.
An example of how components are composed to form a complex system. This system is modeling the interactions between two CPU's and a bus arbiter to access shared memory.

The definition of an event system presented above does not require $E = I \cup O$. The events $E \cap (I \cup O)$ are *internal events*. That is, neither input nor output events. These events arise in one of two ways. The system designer could explicitly use such an event for some internal purpose or through the interconnection of components (see below). Since the users of a component are only interested in the external behaviour of the component we assume internal events are caused by the interconnection of components. To simplify the identification of internal events we will use the following notation:

Definition 3.7: Communication Events.
The set of internal events (communication events) C in an interconnection of two systems S_1 and S_2 is defined as $C=(O_1 \cap I_2) \cup (O_2 \cap I_1)$

If the system designer wants to be able to specify internal events that are not communication events then another class of events can be added to the definition of an event system. Extending the definition of an event system to include another class of events has no effect on any of the results presented in this work.

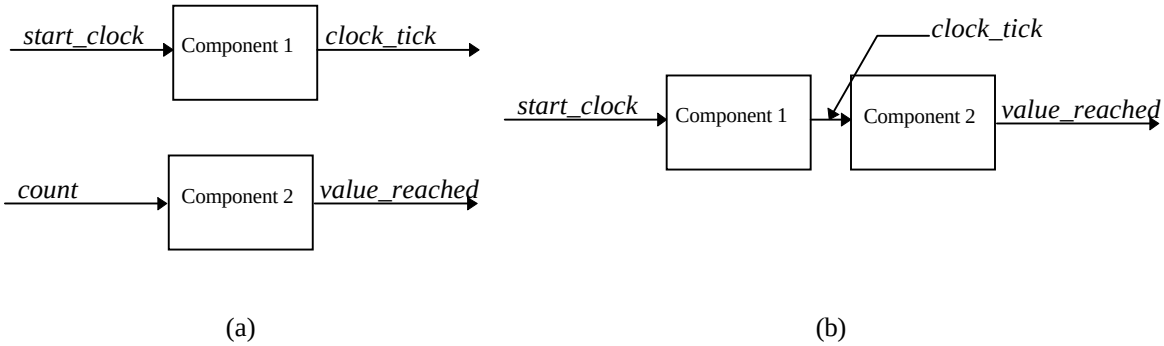


Figure 3.3: Interconnecting Components
(a) Two components that will be interconnected. (b) the input event of the second component is renamed which indicates that the output of the first is to be connected to it.

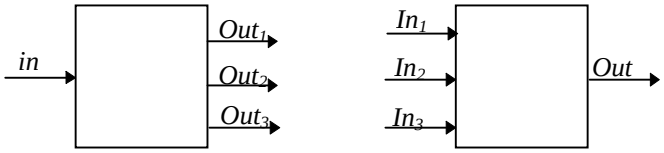
The composition of two components can be thought of as directing output events from one component to become input events at the other. For example, Figure 3.3(a) demonstrates two components. Component 1 is a clock component that produces *clock_tick* events once a second. Component 2 is a counter that produces a *value_reached* event after receiving ten inputs. If a system designer wanted a component that caused something to happen every 10 seconds, he could interconnect these two components by joining the output of the clock component to the input of the counter (Figure 3.3(b)). This combined system accomplishes the desired behaviour. Examining how the above composition accomplished the desired behaviour will be used to motivate our composition operation.

Notice that the communication event (*clock_tick*) is an output of component 1 and an input to component 2. This differs from Figure 3.3(a) where *clock_tick* was only an output of component 1 while component 2's input was called *count*. Therefore, the first step in any composition is to rename the events that will be connected to have the same name.

After this renaming has occurred the events of the composed system are a union of the events of each individual component. Also, the inputs to the system are all the inputs to each component except those connected to the outputs of the other component. Similarly, for the output events. The only remaining aspect of the event system to consider, are the traces of the system. Clearly, a trace of the system restricted to the events of a component must be a trace of that component. We have considered all effects

of composition on an event system and can now define the composition of two components. Before we present the definition we will present some additional requirements that simplify the notation.

To simplify the notation and presentation of the theorems some requirements are required on the interconnection of components. The only valid connection is from an output of one component to the input of another. To achieve another type of connection the event may be routed through a splitter or a merger component. Figure 3.4 demonstrates these components. If a component contains internal event these events may not be used in further interconnections. This requirement makes sense because in a real system only the externally visible behaviour is known to the designer.



(a) (b)

Figure 3.4: Splitting or Merging Event Sequences
(a) The occurrence of the *in* event causes three output events to occur. These events can be used in further connections to simulate connecting the *in* event to three components. (b) After all three *in* events have occurred an *out* event is emitted. This component can be used to merge events.

In the definition of composition we assume that the event renaming mentioned above has already been done.

Definition 3.8: Composition of Components
Given $S_1=<E_1,I_1,O_1,T_1>$ and $S_2=<E_2,I_2,O_2,T_2>$ that satisfy

$$I_1 \cap I_2 = \varnothing$$
$$O_1 \cap O_2 = \varnothing$$
$$(E_1 \setminus (I_1 \cup O_1)) \cap E_2 = \varnothing$$
$$(E_2 \setminus (I_2 \cup O_2)) \cap E_1 = \varnothing$$

then the compostion of S_1 and S_2 produces a new component $S=<E,I,O,T>$ such that:

$$E = E_1 \cup E_2$$
$$I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$$

$$O = (O_1 \setminus I_2) \cup (O_2 \setminus I_1)$$

and $T = \{ a \in E^* \text{ such that } a|E_1 \in T_1 \wedge a|E_2 \in T_2 \}$

The definition of composition implies that the outputs from one component immediately become inputs at the other component. It can be argued that in real systems there always exists some propagation delay and hence our requirement is too strong. It would seem that a better approach would be to allow some time between the occurrence of an output event and its receipt as an input at the other component.

This analysis, though correct, is overly simplistic. The difference between the two is better characterized as synchronized versus non-synchronized communication. An example of synchronized communication might occur in the *Request* event in Figure 3.2 (page 19). In this case it would not be unrealistic that when the CPU generates this event it immediately appears at the bus arbiter. This exchange happens so quickly it can be though of as an atomic event. The *Write_to_Bus* event, however, is a good example of where non-synchronized behaviour is desirable. In this case it is not unreasonable to assume that the *Write_to_Bus* and the *Write_to_Memory* event do not occur simultaneously. In this case other system events may occur between the occurrence of the two events.

Section 5.5 will examine the effects of synchronized communication on composability. It will be shown that the type of synchronization is a factor when the composition of two components fails to preserve a property. In section 5.5.3 we introduce a delay component that can be used to model non-synchronized communication. We defer the introduction until then because the delay component has implications on the composability of properties.

Typical research on composability has proceeded on the basis that a system can be constructed two components at a time. First, two components are interconnected. These are then considered one new component and another component is added. This procedure is repeated until the desired system has been constructed. In this work, however, we will show that the structure of the system is an important consideration for composition. We therefore require a way of expressing this structure. This will be done through the use of a system graph.

Definition 3.9: System Graph.

For a system composed of *k* components construct the following digraph G=<V,D>. Let the set of vertices V be the set of numbers 1 to *k*. An edge (i,j)$\in$D iff there exists an α such that $\alpha \in O_i$ and $\alpha \in I_j$. The above graph is called the *system graph*.

Figure 3.5(b) is an example of a system graph for the system in Figure 3.5(a). Observe that the system graph captures the possible flow of events. The system graph has a cycle if and only if the system involves feedback.

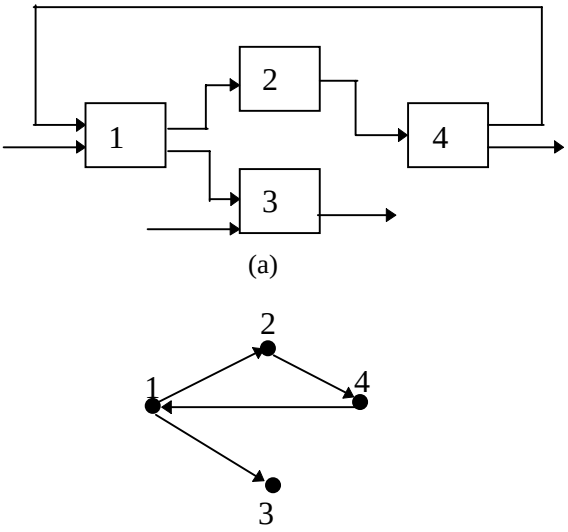


Figure 3.5: An Example of a System Graph
Figure (b) is the system graph for the system in part (a).

Definition 3.10: Feedback Path.

The feedback path from component *i* is a path that starts at vertex *i* and ends at vertex *i*. It is possible that a component is not part of any feedback path.

Definition 3.11: Number of Components in the Feedback Path.

The number of components in the feedback path is defined as the smallest number of vertices visited in the path that starts at vertex *i* and ends at vertex *i*.

For example, in Figure 3.5(b) it can be seen that the path 1-2-4-1 is a feedback path and the number of components in that path is 3.

3.5. Summary

In this chapter we presented the event system formalism that will be used throughout this work. In the next chapter we introduce security properties.

4. Security Properties

> The vanity of being known to be trusted with a secret is generally one of the chief motives to disclose it.
>
> ***Dr. Samuel Johnson*** *(1709-1784)*
> *English author, lexicographer*

4.1. Introduction

Each researcher that has proposed a new security property has constructed his own notation and formalism. Every new security property proposal must be accompanied with a proof of composability. With different notations and assumptions about the model of components, comparing the strengths and weaknesses of the various security properties has been difficult. In this chapter we examine security properties in general. We then present a unified framework for the specification and analysis of security properties. This framework is used throughout this work in proving properties about the composition of components that satisfy security properties.

There have been various frameworks presented for analyzing properties of components [Hoare85] [Abadi & Lamport90] [McLean94] [Nestor93] [Hinton96]. Why is another framework required? In Section 4.7 we show that security properties do not fall within the Alpern-Schneider [Alpern & Schneider85] safety/liveness framework presented in Chapter 2. Therefore, the Abadi and Lamport [Abadi and Lamport90] composition principle cannot be applied to security properties.

One of the first attempts to provide a general theory of security properties was the use of Selective Interleaving Functions [Mclean94]. McLean's framework as will be demonstrated is only applicable to a subset of security properties. Possibly its greatest weakness, however, is that it does not allow for an obvious specification of security. Our framework captures the intuitive notion of security properties and can be used to determine the composability of components that satisfy security properties. A comparison of selective interleaving functions and our framework is presented in Chapter 6.

In this chapter we introduce and motivate security properties. Chapter 5 presents composition theorems that can be used to determine the composability of these properties.

4.2. Properties of Secure Systems

A secure computer system is one that has the properties of confidentiality, integrity and availability [McLean94]. There is no clear distinction between these properties since they are not independent. A Trojan Horse that corrupts files makes these files unavailable. Also, a Trojan Horse that makes a system unavailable can be used to transmit data to low level users. Therefore, confidentiality cannot be assumed without some degree of integrity and availability. Each of confidentially, integrity and availability is useful, however, because each category has its own set of issues.

In this work we are interested in confidentiality properties. The goal of confidentiality is to prevent low level users from deducing anything about high level activity. The security policy defines exactly what low level users are forbidden to discover. For example, it may be considered desirable to ensure low level users cannot determine which high level inputs have occurred. Or we may say, information about high level inputs may not flow to LLU. The security policy dictates what flows are permissible and which are not. A security property is an instantiation of a policy. There may be more than one property that satisfies a given policy. In this work we do not advocate any specific security policy. We consider security properties in general[2].

As with all of the work on confidentiality, we take an optimistic view of the LLUs abilities and a pessimistic view of the intent of the HLUs. For example, we assume LLUs have complete knowledge of the construction of the system and that HLUs, when confronted with a choice, will make the one that compromises system security the most. Taking this approach we get a lower bound on the security of the system being considered.

---

[2] In this work when referring to security we mean confidentiality.

4.3. Inference

To understand what a security property is, we must first understand how low level users can infer information about high level users' activities. Information will be deemed to flow from high level users to low level users when the low level users observe something they believe is connected with high level user activity:

> "Information is transmitted along an object when variety in the events engaged by a [high level] user can be conveyed to a [low level] user as a result of [the high level users] interaction with the object." [Foley87]

There are two types of inferences that can be made:

1. possibilistic
2. probabilistic

In the possibilistic case, one is interested in the possibility of certain events. In the probabilistic case the probability of the events is also considered. Nearly all work on secure systems has been on possibilistic properties and systems. Some work on probabilistic properties has been presented [Gray90] [McLean90] [Gray92].

In this work we are interested in possibilistic properties and inferences. Johnson and Thayer argued that "possibilistic specifications for computer systems [are] inadequate for addressing the main problems of computer security" [Johnson & Thayer88]. While this might be the case we cannot hope to understand the dynamics of a probabilistic system without first understanding the possibilistic case.

In the following discussion it will be useful to examine the bunch of traces that are consistent with a given low level observation. The definition of a low level equivalent bunch captures this notion.

Definition 4.1: Low level Equivalent Bunch.

Given a trace τ and a System *S*, $B_{low}(\tau,S)$ is the bunch of traces that have the same low level events as τ in the same order. We will write $B_{low}(\tau)$ when referring to an arbitrary system *S*. Formally,

$$\S s:traces(S)\cdot \tau|L=s|L$$

When a low level user observes a sequence of events τ_{low} he knows there exists a τ such that $\tau|L=\tau_{low}$. Since we assume that the low level user knows the architecture of the

component he can determine the low level equivalent bunch corresponding to τ_{low}. The question that we wish to address is what can the user infer about $\tau|H$?

One of the first attempts to model the flow of information in a secure system is due to Sutherland [Sutherland86]. His theory of information flow is based on logical deduction. Sutherland argued that if there were high level event sequences such that no element of $B_{low}(\tau_{low})$ had this high level sequence then the low level user would have inferred something about high level behaviour. Formally, information does not flow if and only if $\forall\tau_{low}$:traces(S)|L·$\forall\tau_H$:traces(S)|H·τ_H:($B_{low}(\tau_{low}$,S)|H). Thus a system has no undesirable information flows if all possible high level event sequences are consistent with every possible low level event sequence.

This definition of inference seems reasonable but it is neither complete nor useful. We will not cover all the details of why this definition of inference is not acceptable for secure systems but we will cover some of the major issues below. For an excellent discussion on the subject see "A Trusted Network Architecture" [Thompson et. al 88] or "Information Flow in Nondeterministic Systems" [Wittbold and Johnson90].

One of the biggest problems with Sutherland's theory is that it allows systems with undesirable information flows to be called secure and systems that do not have undesirable flows to be called insecure. Sutherland's assertion that all possible high level activity must be compatible with every low level sequence does not encompass the notion of security correctly.

Consider a system whose only function is to copy all low level inputs to high level outputs. This system is clearly secure. However, Sutherland's theory indicates that a flow from high to low level users to low level users exists. It is true that the low level user has knowledge of high level events but he has not gained any new information. In this case Sutherland's theory is too strong. Sutherland's definition of information flow is symmetric. If there is a flow from *A* to *B* then there must be a flow from *B* to *A*. Security, however, is an asymmetric property. Information flows from LLU to HLU are allowed but information flows from HLU to LLU are not.

Sutherland's theory also allows the construction of a system that hsa real but unacceptable flows. This happens because not all possible methods of transmitting

information have been considered. Only the existence of one interleaving of a high level event sequence and a low level observation is required. The low level user in examining $B_{low}(\tau_{low})$ can however make the following observations:

1. A particular high level input sequence is not consistent with τ_{low}.
2. A particular interleaving of high level event sequences is not possible.
3. A high level output event that does not depend on any high level input events is not consistent with τ_{low}.

No other observations can be made. High level outputs that depend on input events are not a factor because if something can be inferred about these events then something can be inferred about high level inputs. We do not want to imply that the LLU will not know that a high level output has occurred. But, we do imply that the knowledge of the occurrence gives no information about the activities of the high level users.

The first item above has been addressed by many researchers but the second has received little attention. Guttman and Nadel [Guttman & Nadel88] mentioned it as a problem they were trying to address in the presentation of their security property. Their exposition of the problem does not adequately address the issue. Furthermore, their examples are not convincing enough to demonstrate the problem with interleavings. It was hypothesized by Lee [Lee et. al 92] that the interleavings problem might be connected to the issue of nondeterminism. We agree with this and will demonstrate why interleavings need to be considered.

Example 4.1: Machine *A* has one high level input *in*, and one high-level output *out* which is caused by *in* after some processing. There is a low-level *cancel* input, which cancels any high-level processing that is underway, and a low-level *ack* output that acknowledges the *cancel* input after some time interval. If there is high-level processing at the time of the *ack*, that is, if the number of *out* events is less than the number of *in* events, all high-level processing is terminated, and no *out* will occur until after the next uncancelled *in*. If there is no high-level processing at the time of *ack*, then a low-level *error* output may be produced at some time following the *ack*; however, the *error* output is not guaranteed to occur.

It is easy to see that for any sequence of low level events every high level input sequence is possible. However, consider the low level observation:

<cancel, ack, error>

This low level observation precludes the following interleaving of the high level sequence <*in, out*>:

<*cancel, in, ack, out, error*>

The *out* event must come before the *ack* event. Therefore, the low level user knows that at the time of the *ack* no high level events are present. This information can be used to transmit information from the high level user to the low level user. ❑

In light of the above discussion we propose the following definition for information flow in a secure system:

Definition 4.2: Information Flow.
Information flows from high level users to low level users if and only if the low level user's observation of τ_{low} implies that at least one high level event sequence or interleaving is not possible.

Information flows from high level users to low level users if there exists a high level trace or interleaving such that if it had occurred then τ_{low} could not have occurred. Care must be taken in interpreting this statement. If low level actions influence high level behaviour then it is possible for a particular sequence not to be possible because the low level influence precludes it. However, in this case no inference is possible. For example, the low level user may know his influence could not possibly result in a particular high level output, or in the extreme case, may know exactly what the output must be. But what is the inference? The low level user is precluding high level events from occurring. Therefore, he can communicate with the high level user through a covert channel *but* low level to high level communication is already allowed.

4.3.1. The Perfect Security Property

Separability is an example of perfect security [McLean94]. This is because no interaction is allowed between high level and low level events[3]. It is like having two separate systems, one running the high level processes and one running the low level

[3] We must stress that this is in the possibilistic sense. It is possible to construct a Separability secure system that has covert channels.

processes. Separability can be defined as follows. For every pair of traces τ_1 and τ_2 the trace τ such that $\tau|L=\tau_1|L$ and $\tau|H=\tau_2|H$ is a valid trace.

The problem with Separability is that it does not allow low level users to influence high level activity. For example, a computer system that keeps a journal of all low level user activity on a high level device would not be considered secure.

We will present a security property that is the weakest property that does not allow a flow from high level users to low level users. Our property will allow low level users to influence high level user activity. The Perfect Security Property (PSP) will be proven to be the weakest property that does not allow a flow from high level users to low level users.

The idea behind PSP is the same as that behind Separability. All possible high level activity and interleavings must be possible with all low level activity. The difference is that PSP allows high level outputs to be dependent on low level events. The choice of output event for any given interleaving can depend on low level events. This implies that not all interleavings of high level events are possible. This, as will be shown, does not reduce security because the low level user will not know how he has influenced high level outputs.

The traces of the system are constructed from the set of events of the system. The set of events defines all the events that the system can engage in. The definition of PSP requires the insertion of events in traces. To simplify the presentation of PSP we need to represent the insertion of no event. To accomplish this we introduce a special event the use of which has no effect on the set of possible traces. It is merely a placeholder.

Definition 4.3: Null Events

The symbol ε will be used to represent an event that is governed by the following axiom:

$$p^{\wedge}<\varepsilon>^{\wedge}s = p^{\wedge}s \qquad p^{\wedge}s \in T$$

The following function gives all the possible high level events that may occur after a prefix of a given trace. This function will be used to construct all the interleavings of high level sequence with low level events. Notice that since we will be using a function

that gives all possible events after a trace, it is possible that the low level activity in the trace can influence the possible events.

Definition 4.4: Possible Event Function.

Given a trace τ, Let $\nu(\tau)=\{e:H\cup\varepsilon\cdot trace_s(\tau^\wedge<e>)$. This function returns the bunch of all possible high level events that can occur after τ. The function $\nu(\tau)$ is called the *possible event* function.

The definition of the possible event function requires ε to be a possible event. This will ensure that there will be no case where an event *must* occur.

The following defines PSP. The idea behind the property is that for any low level observation the following must be true:

1. All interleavings of high level input sequences must be possible.
2. High level outputs can be inserted anywhere in the trace (assuming they are possible) and can depend on low level activity.

If all high level input sequences are possible and high level outputs can be inserted anywhere then the low level user cannot determine anything about high level activity. This observation will be proven below.

Definition 4.5: The Perfect Bunch

Given an event system S and a low level observation τ_{low}, if the bunch $B_{low}(\tau_{low},S)$ contains the following traces then the bunch is *perfect*.

$$\forall p,s:E^*\cdot traces_s(p^\wedge s)\wedge s|H=<>\wedge p^\wedge s\,|L=\tau_{low}\Rightarrow\forall\alpha:\nu(p)\cdot p^\wedge\alpha^\wedge s:\ B_{low}(\tau_{low},S)$$

Definition 4.6: The Perfect Security Policy.

If for all τ_{low} the bunch $B_{low}(\tau_{low},S)$ is *perfect* then the system satisfies PSP.

The expression of the property might seem complicated but fortunately there exists a simple procedure to determine if a component satisfies PSP (see Chapter 7). We will use this property to determine the strength of the properties presented in the literature (see section 4.5).

The definition of PSP can be transformed into a definition for Separability by defining the possible event function as follows:

$$\nu(\tau)=\S e:H\cup\varepsilon\cdot trace_s(\ (\tau^\frown e)\ |H\ )$$

The only difference between this definition of the possible event function and the one given in Definition 4.4 is that this possible events only depend on the preceding high level events, not the whole trace. We defer the proofs that PSP allows no information flow and is the weakest such property until section 4.6 because we require concepts that have not yet been introduced.

4.4. Security Properties

In the previous section we defined a security property that does not allow any information to flow from high level users to low level users. It would appear that with this property no other property is required. There are many reasons why other properties are required. For example:

1. The risk analysis of the system indicates little threat of Trojan horses. In this case a security property with the possibility of some unauthorized flows might be acceptable.
2. A desired component does not satisfy this property and a weaker property must be used.
3. The flow that PSP objects to might, under further analysis, not be a threat to the system.

The definition of PSP and information flow gave a hint at what a security property is. The definition of PSP was done by indicating what elements must be present in the low level equivalent bunch for a low level observation τ_{low}. We can generalize this to cover all security properties: A security property indicates what traces must be consistent with a low level observation τ_{low}. In other words, the low level user observing τ_{low} can determine the low level equivalent bunch of traces. The security property ensures that certain traces are present in this bunch. Therefore, the low level user cannot distinguish which of these traces has occurred.

Definition 4.7: Security Properties

A system satisfies a security property if and only if all low level equivalent bunches satisfy the security property predicate *P*. Formally,

$\forall t:traces(S)|L\cdot P(B_{low}(\tau,S))$

We will write *P(S)* to indicate that system *S* satisfies property *P*.

A security property will ensure that certain traces are in the low level equivalent bunch. This is not to imply that other traces may not be present in this bunch. For example, if property P_1 implies property P_2 and the component is known to satisfy P_2, then it may also satisfy P_1, but this is not guaranteed. The property specifies what must occur. All other traces are coincidental and can vary from component to component.

For each security property there exists a bunch of traces that are guaranteed to be consistent with τ_{low}. As will be shown, constructing this bunch is a simple procedure once the property has been expressed. The following definition will be used to identify this bunch.

Definition 4.8: Guaranteed low level equivalent bunch

We will write $G_P(\tau,S)$ to identify the bunch of traces that property *P* guarantees will be present for trace τ in System *S*. We will write $G_p(\tau)$ if the system to which we are referring to is obvious from the context.

This bunch should not be confused with the bunch $B_{low}(\tau,S)$. $B_{low}(\tau,S)$ gives the bunch of all traces with the same low level events. $G_p(\tau,S)$ is the bunch of traces that are required to be present in the set *T* of the system for the system to satisfy *P*. Clearly, if a system *S* satisfies a property *P*, $G_p(\tau,S) : B_{low}(\tau,S)$

We have been deliberately vague about how security properties can be expressed. Beginning in the next section we examine some of the security properties that have been presented in the literature. This discussion will be used to demonstrate how security properties can be expressed. The security properties that we shall present here and analyze in Chapter 5 are intended to illustrate the power of our framework and how other frameworks and *ad hoc* approaches to security fail.

4.4.1. Noninference

Noninference was introduced by O'Halloran [O'Halloran90]. It attempts to separate the low level activity from the high level activity. Informally, Noninference

requires that for any trace of the system removing all high level events results in a trace that is still valid.

Noninference is too strong for systems that have a high level output without a high level input. As an example consider a system the only function of which is to keep a journal of all low level events on a high level device. This system is secure. The low level user does not know anything about what high level users are doing. This system, however, does not satisfy the Noninference property.

McLean [McLean94] extends Noninference as follows. For any trace τ, it must be possible to find another trace σ such that the low level events of τ are equal to σ and σ has no high level inputs. McLean calls this stronger property Generalized Noninference.

Consider the following:

$$\forall \tau : traces(S) | L \cdot GN(B_{low}(\tau, S))$$
$$GN(B) \equiv \exists t : B \cdot t | HI = <>$$

This satisfies the definition of a security property. The GN predicate ensures that the trace without any high level inputs is always possible for any low level observation. Therefore, for all possible low level observations a trace can be found with the same low level events but with no high level inputs.

In the chapters that follow it will be required to determine all the traces that are guaranteed to be indistinguishable to the low level user. For Generalized Noninference this bunch can be expressed as:

$$G_{GN}(\tau, S) = \S s : traces(S) \cdot s | L = \tau | L \wedge s | HI = <>$$

4.4.2. Noninterference

Noninterference is a security property introduced by Goguen and Meseguer [Goguen & Meseguer82] [Goguen & Meseguer84]. It captures the attractive notion that system security is preserved whenever high level users are prevented from influencing the behaviour of low level users. Goguen and Meseguer's original definition of Noninterference was only applicable to deterministic systems. McCullough [McCullough87] [McCullough88] extended the definition to encompass non-deterministic systems.

McCullough's definition of Generalized Noninterference (GNI) can be informally defined as follows: Given a trace τ, modifying it by inserting or deleting high level inputs results in a sequence σ, which is not necessarily a valid trace. This is referred to as a *perturbation* of τ. It must be possible to construct a valid trace τ' from σ by inserting or deleting high level outputs. This is called a correction to the perturbation.

We will now formally define Generalized Noninterference.

$$\forall\tau{:}\text{traces}(S)|L\cdot\ \text{GNI}(B_{low}(\tau))$$

$$\text{GNI}(\ A\ ) \equiv \forall t{:}\textit{interleave}(\text{HI}^{*},\tau_{low})\cdot\exists s{:}\ A\cdot t=s|(L\cup HI)$$

To simplify the presentation of the GNI predicate, we used τ_{low} to represent the low level trace. This can be extracted from any member of *A*.

Once the security predicate has been written the bunch of traces that are guaranteed to look like a particular trace τ_{low} to the low level user can be formed:

$$G_{GNI}(\tau,S) = \S s{:}\text{traces}(S)\cdot\exists t{:}\textit{interleave}(\text{HI}^{*},\tau|L)\cdot\ s|(L\cup HI) = t$$

4.4.2.1.Forward Correctability

The above definition of Generalized Noninterference is different than McCullough's original definition. The above definition allows a correction to a perturbation to occur at any point in the trace, even before the perturbation. McCullough called the possibility of correcting before the perturbation a "violation of causality." We will show in section 7.5.1 that this violation can only occur in a non-deterministic system. We can also define a causal or Forward Correctable version of GNI where corrections can only occur after the perturbation. Unless otherwise specified when referring to Generalized Noninterference we will refer to the one defined in the previous section.

4.4.3. Non-Deducible Output Security

The previous two examples were of security properties are founded on the notion of preventing a LLU from deducing anything about high level inputs. Our definition of security is not limited to this type of security. To illustrate a different form of security we present Guttman and Nadel's Non-Deducible Output Security [Guttman & Nadel88]. In this example we start with the formal description of the property and demonstrate how it can be analyzed. Non-Deducible Output Security can be expressed as:

$$\forall\tau{:}traces(S)|L\cdot NDO(B_{low}(\tau))$$

$$NDO(A) \equiv \forall t{:}traces(S)\cdot t|LI=\tau_{low}|LI \Rightarrow \exists s{:}\ A\ \cdot s|(H\cup LI)=t|(H\cup LI)$$

Once again we have used τ_{low} to simplify the notation. This can be extracted from any trace of the bunch *A*.

The analysis of a property begins by writing the security property in a form where it is obvious which traces look like a trace τ to a LLU.

$$G_{NDO}(\tau,S) = \S s{:}traces(S)\cdot\exists t{:}traces(S)\cdot t|LI=\tau|LI \wedge s|L=t|L \wedge s|(H\cup LI) = t|(H\cup LI)$$

If the LLU sees a trace τ_{low} he can determine the bunch $B_{low}(\tau_{low},S)$. All of these traces are indistinguishable to a low level user from a trace *s* that has the same low level events as τ but the high level events come from another trace that has the same low level input events. Since the low level user cannot determine which high level events were chosen the observation of τ_{low} gives the user no new information about high inputs or outputs. Furthermore, since the merging was performed arbitrarily, the low observation is also compatible with all interleavings and so give no information about which interleaving occurred.

Output Non-Deducibility cannot be expressed in McLean's Selective Interleaving Framework. This property is not the interleavings of two traces (see section 6.2).

4.4.4. Separability

Separability is an example of perfect security [McLean94]. Separability is perfect security because no interaction is allowed between high level and low level events. It is like having two separate systems, one running the high level processes and one running the low level processes. Separability can be defined as follows. For every pair of traces τ_1 and τ_2 the trace τ such that $\tau|L=\tau_1|L$ and $\tau|H=\tau_2|H$ is a valid trace.

No matter what the low level user observes, every possible sequence of high level events is possible. Therefore, the low level user cannot gain any new information.

This property can be formalized as:

$$\forall\tau{:}traces(S)|L\cdot\ SEPARABILITY(B_{low}(\tau))$$

$$SEPARABILITY(\ A\ ) \equiv \forall t{:}traces(S)|H\cdot \textit{interleave}(t,\tau_{low}){:}A$$

$$G_{SEPARABILITY}(\tau,S) = \S s{:}traces(S)\cdot\exists t{:}traces(S)|H\cdot s|L=\tau_{low}\wedge s{:}\textit{interleave}(t,\tau_{low})$$

4.5. Comparing Security Properties

Before we can compare security properties we must decide what it means to compare them. Consider any component C_1 that satisfies property P_1 and any component C_2 that satisfies P_2. We can ask does C_1 always satisfy P_2? If it does then property P_2 is weaker than P_1. If C_2 always satisfies P_1 then P_1 is weaker than P_2[4]. If neither is true then P_1 and P_2 are not comparable. By performing the above comparison between all properties a partial ordering of properties can be constructed.

Our formalism provides a mechanical method of evaluating the relative strengths of security properties. Since we have a logical expression for our properties the comparison is simple. To compare properties *P* and *Q* evaluate $P \Rightarrow Q$ and $Q \Rightarrow P$. If the first statement is true then *P* is stronger then *Q*. If the second statement is true then *Q* is stronger than *P*. If both are true the properties are equal and if neither is true they are not comparable.

Example 4.2: We will compare Generalized Noninference to Generalized Noninterference:

We reproduce the definition of Generalized Noninference and Generalized Noninterference here:

$$GN(B) = \exists t:B \cdot t|HI = <>$$

$$GNI(B) \equiv \forall t:\mathit{interleave}(HI^*, \tau_{low}) \cdot \exists s:B \cdot t = s|(L \cup HI)$$

First we will show that GNI implies Generalized Noninference:

	$\forall \tau$:traces(S)|L·GNI($B_{low}(\tau,S)$)	Definition of GNI
=	$\forall \tau$:traces(S)|L·$\forall t:\mathit{interleave}(HI^*,\tau_{low})\cdot\exists s:B_{low}(\tau,S)\cdot t=s\|(L\cup HI)$	Specialization with $t=\tau$
$\Rightarrow$	$\forall \tau$:traces(S)|L·$\exists s:B_{low}(\tau,S)\cdot\tau=s\|(L\cup HI)$	Distributive
$\Rightarrow$	$\forall \tau$:traces(S)|L·$\exists s:B_{low}(\tau,S)\cdot\tau\|L = s\|L \wedge \tau\|HI = s\|HI$	Definition of $B_{low}(\tau,S)$
=	$\forall \tau$:traces(S)|L·$\exists s:B_{low}(\tau,S)\cdot\tau\|HI = s\|HI$	τ has no high level events
=	$\forall \tau$:traces(S)|L·$\exists s:B_{low}(\tau,S)\cdot<> = s\|HI$	Definition of GN
=	$\forall \tau$:traces(S)|L·GN($B_{low}(\tau,S)$)	

Now we show that Generalized Noninference does not imply GNI:

$\forall \tau$:traces(S)|L·GN($B_{low}(\tau,S)$) $\Rightarrow$ $\forall \tau$:traces(S)|L·GNI($B_{low}(\tau,S)$)

Definition of GNI & GN

---

[4] If both cases are true then P_1 is equal to P_2.

$= \quad \forall\tau{:}traces(S)|L\cdot\exists s{:}B_{low}(\tau,S)\cdot s|HI = <> \Rightarrow$

$\forall\tau{:}traces(S)|L\cdot\forall t{:}\textit{interleave}(HI^{*},\tau_{low})\cdot\exists s{:}B_{low}(\tau,S)\cdot t{=}s|(L\cup HI)$

Specialization such that $t|HI\neq<>$

$\Rightarrow \forall\tau{:}traces(S)|L\cdot\exists s{:}B_{low}(\tau,S)\cdot s|HI = <> \Rightarrow \forall\tau{:}traces(S)|L\cdot\exists s{:}B_{low}(\tau,S)\cdot t = s|(L \cup HI)$

Distributive, Definition of $B_{low}(\tau,S)$ and the Specialization condition

$\Rightarrow \forall\tau{:}traces(S)|L\cdot\exists s{:}B_{low}(\tau,S)\cdot s|HI = <> \Rightarrow \forall\tau{:}traces(S)|L\cdot\exists s{:}B_{low}(\tau,S)\cdot\neg s|HI = <>$

$= \quad \perp$

Therefore, GN does not imply GNI and GN is a weaker property than GNI. ❑

By applying the above technique to the security properties presented above the following lattice can be constructed.

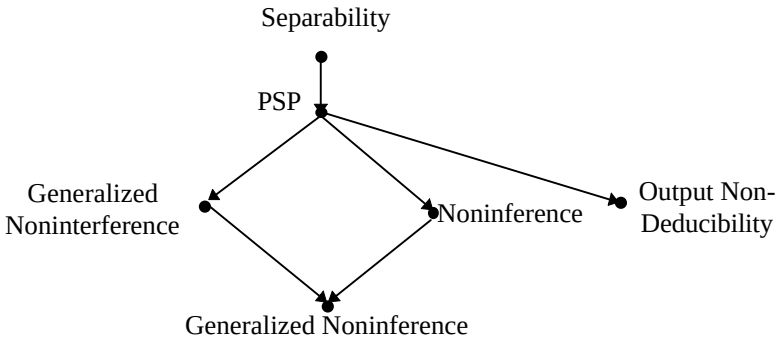


Figure 4.1: A Partial Ordering of Security Properties

The arrows in the lattice indicate which property implies which other. For example PSP implies Generalized Noninterference and by transitivity Generalized Noninference. An instructive way to represent part of the above lattice is to only consider the elements that can be totally ordered.

Figure 4.2 shows the ordering of most of the security properties that have been presented in the literature. First notice that, for example, Separability secure systems are both GNI secure and Generalized Noninference secure. Therefore, if a system designer wishes the system to be Generalized Noninference secure and it is known that it is GNI secure then it is also Generalized Noninference secure. Also notice that PSP, defined in section 4.3.1, partitions the figure into two. This can be used to determine the strength of

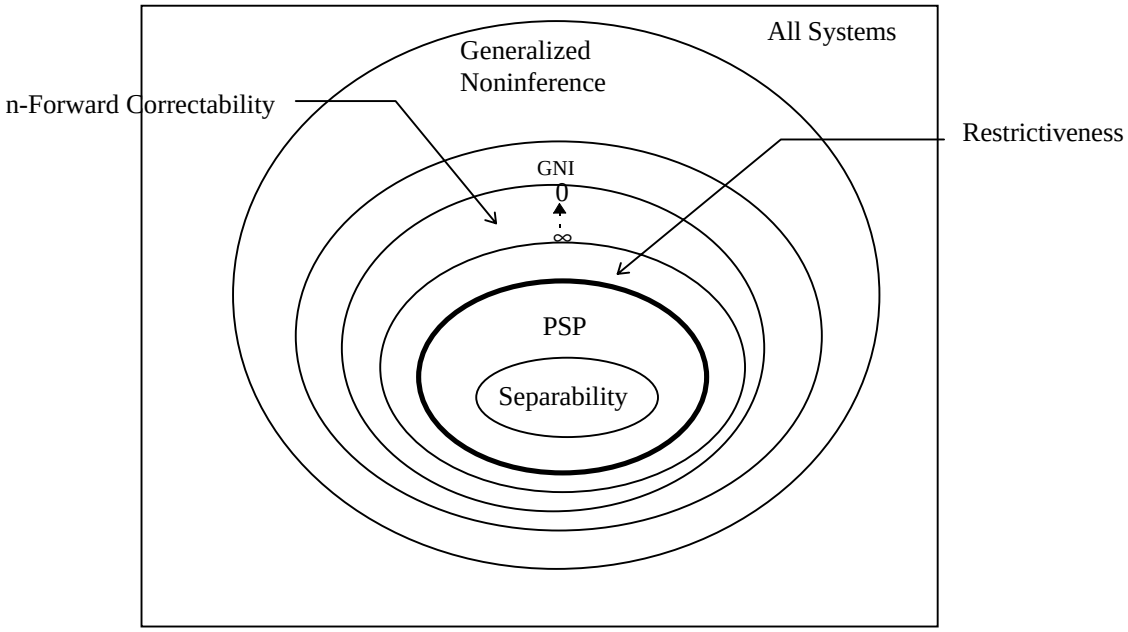


Figure 4.2: A Total Ordering of Most Possibilistic Properties

the properties. We can see that Separability is a stronger property than PSP. Therefore, systems with no information flow are being unnecessarily rejected.

Most security properties defined in the literature are weaker than PSP. This may be surprising but can be explained because high level interleavings are not considered by any of the weaker properties. Example 4.1 on page 29 was used to demonstrate that the interleavings of traces can be used to transmit information from high level users to low level users. It can be shown that the component of Example 4.1 is Restrictiveness secure [Lee et. al 92]. From Figure 4.2 it can be seen that this component also satisfies most of the other properties presented in the literature. Therefore, all weaker properties than PSP allow systems to be called secure which are not.

4.6. PSP Security Proofs

In this section we prove that PSP allows no information flow between high level users and low level users. We also prove that it is the weakest such property.

Theorem 4.1: PSP does not allow any information to flow from high level users to low level users.

Proof:

Assume that there is a system *S* that satisfies PSP and allows high level information to flow to low level users. By the definition of information flow there must be some high level sequence that is not possible. By construction all possible high level input sequences and high level input sequence interleavings are possible. Furthermore, all possible interleavings of high level outputs are present. The low level equivalent bunch for τ_{low} contains all sequences that could give the low level users any information about high level activity.

❑

Theorem 4.2: PSP is the weakest security property that does not allow information flow from high level users to low level users.

Proof:

Theorem 4.1 proved that PSP does not allow information to flow from high level users to low level users. We must therefore prove that any weaker property must allow flows from high level users to low level users.

Assume that there exists a property, *P*, that is weaker than PSP and that does not have any unauthorized information flows. Let *S* be a system that does not satisfy PSP but satisfies *P*. Let μ be a trace such that $\neg\mu{:}G_P(\mu,S)$. Such a trace exists because *P* is weaker than PSP.

When the low level user observes τ_{low} he knows that the trace μ is not possible. We now show that the absence of this trace gives the low level user additional knowledge about high level activity. Since μ is not a possible trace one of the following must be true:

1. The high level input sequence of τ is not consistent with the τ_{low} observation
2. A high level output event that does not depend on input events must occur before some low level trace τ_{low} because it influences the subsequent behaviour of the trace.
3. The interleaving of high level events given by μ is not possible with the observation of τ_{low}. By the construction of μ the sequence of events $\mu|H$ is a valid for some τ_{low}. The

absence of this interleaving with the observed τ_{low} gives the low level user the knowledge of some aspect of high level state.

In all the cases the absence of the trace μ gives the low level user additional knowledge about high level activity. ❑

4.7. Security Properties vs. Safety/Liveness Properties

In section 2.4.2 we presented the Alpern and Schneider safety/liveness model of properties. This model is currently the dominant model in the specification of analysis of programs [McLean94]. Properties are regarded as sets of traces and a component satisfies a property if its set of traces is a subset of the property's set. With this notion of refinement and Abadi and Lamport's composition principle it would be desirable to be able to express security properties in this manner. Security properties, however, are not preserved by this type of refinement [McLean92b] [McLean94].

In this section we demonstrate that security properties cannot be expressed in the Alpern and Schneider framework. McLean has demonstrated this in "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions" [McLean94] but for a different model of components. We will prove this for the event systems we are considering.

Before we can prove the required result we must define the notion of one component being a subset of another.

Definition 4.9: Event System Space

An event system space is a 4-tuple <E,I,O,T> where E, I, O, T are defined as in the definition of an event system (see Definition 3.3 page 16) with $T=E^*$. We will write Š for the event system space.

Definition 4.10: An Element of a System Space.

A system $S=<E_1,I_1,O_1,T_1>$ is a subset of the system space $Š=<E,I,O,T>$ if and only if $E_1 \subseteq E$, $I_1 \subseteq I$, $O_1 \subseteq O$, $T_1 \subseteq T$.

Theorem 4.3: Security properties are not expressible as sets of traces.

Proof:

Let *T* be the subset of the set of traces of Š that satisfy a security property P. Any subset *S* of Š whose set of traces are a subset of *T* satisfies P. The satisfaction of a security property ensures that a system has certain behaviours. A security property is defined as all low level equivalent bunches of a system satisfying a predicate. For a property to be satisfied the required traces of the bunch must be present. Construct a system S whose traces are a subset of *T* and the security property predicate is false for some low level observation τ_{low}. Such a system exists because removing one of the traces required to make *P* true will still result in the set of traces being a subset of *T*. The set of traces of S does not satisfy the security property *P* but is a subset of *T*. This yields a contradiction. Therefore, P cannot be expressed as a set of traces. ❑

The proof demonstrates that the refinement step may eliminate some possible behaviours of the system. Eliminating these behaviours means that the security property might no longer hold.

4.8. Conclusions

In this section we have presented the notion of security properties. This definition is general and intuitively appealing. We also demonstrated that security properties do not fall within the safety/liveness framework of Alpern and Schneider. In the next section we begin our discussion on the composition of components that satisfy security properties.

5. Composition And The Emergence Of Security Properties

> Others find their intellectual pleasure lies in the theory, not the practice.
>
> ***Patrick White*** *(1912-1990)*
> *Australian novelist*

5.1. Introduction

The purpose of modeling a system is to be able to predict its behaviour. To be able to predict the behaviour of a system, rules for the effects of interconnecting components are required. These rules should allow the system designer to know what property the system satisfies given the properties of each component. If the property of interest falls within the safety-liveness framework then the Abadi and Lamport composition principle may be used. If not, the system designer must evaluate the system to determine what properties it enforces.

In the previous chapter we demonstrated that security properties do not fall within the safety-liveness framework. Therefore, Abadi and Lamport's composition principle cannot be applied. In this chapter we present composition results for security properties. This gives the system designer the ability to predict the resulting security property of a composition given the property of each of the components.

There are two different approaches a system designer can take. The system designer may want to know what property two or more of the components must satisfy so that when they are interconnected the system satisfies a property *P*. The other approach is to determine what properties are satisfied by the system that results from the composition of two (or more) components with given properties.

The approaches can be seen to be duals of each other. In the first approach the system is *decomposed* to determine what its constituent parts must satisfy. In the other approach the system is *composed* to determine what the resulting system satisfies.

Both approaches are required because they satisfy different needs. If a desired property of the system might not be preserved under composition then it is required to decompose the system to determine what each component must satisfy. If the system designer is composing several components then he wishes to know what properties the resulting system will satisfy. Notice that if a property is always preserved under composition then both approaches will uncover it.

5.2. Classification of Properties

When several components that satisfy a particular property *P* are composed one of three things may happen:

1. The resulting system will satisfy the property *P*.
2. The resulting system might satisfy the property *P*.
3. The resulting system will never satisfy the property *P*.

The distinctions have important implications for the system designer.

It is desirable to identify properties such that the composition of several components that satisfy a property always result in a system that satisfies that property. We will call such properties *component independent properties*. With components that satisfy component independence the system designer is free to interconnect them and need not be concerned about the property not holding. Unfortunately, not all properties are component independent.

Components that satisfy some property may be composed so that the resulting system might not satisfy the property. Special attention is required from the system designer to ensure the resulting system satisfies the desired property. If no theory of component composition were available the system designer would have to reevaluate the system after every newly added component. Fortunately, we can show that this is not required. In the following sections we present criteria that will allow the system designer to know if the composition will preserve the property or not. We will call properties that might not be preserved by composition *component dependent properties.*

The last possible behaviour of a property is such that the composition of components that satisfy a property invariably results in a system that never satisfies the

property. In this case the system designer knows that these components must never be interconnected.

The system designer also needs to be able to determine under what circumstances a property emerges on composition. An emergent property is one that is not satisfied individually by every constituent component but is satisfied by their composition. We will provide criteria to determine when and how a property may emerge on composition.

5.3. Interconnections of Components

In this work we examine what effect the interconnection of systems has on security properties. We are interested in two types of interconnections: cascade composition and feedback composition. It can be shown that these are sufficient to perform general composition[5].

Cascades are formed by taking two components S_1 and S_2 and passing some of S_1's output events to S_2's input events (see Figure 5.1). We assume that S_1's output meets any environment restrictions expected by S_2's input. That is, S_1's outputs are acceptable inputs for S_2. The resulting system can now be considered a new component and another component can be added. In this fashion larger and larger cascade systems can be constructed.

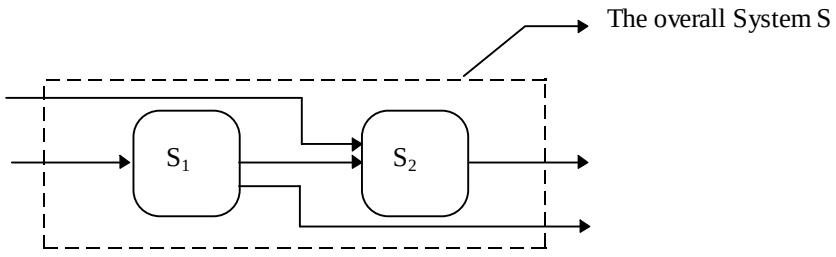


Figure 5.1: Cascade Composition
The cascade composition of components S_1 and S_2. Some of S_1's output events are fed into S_2 as inputs.

---

[5] McLean [McLean94] demonstrates this with product composition (cascade composition with no internal events) and feedback composition. The ability to perform general composition from cascade and feedback composition was noted by Millen [Millen90] who attributes it to Rushby.

The other type of composition involves feedback. In a system that is composed with feedback some of S_2's outputs in Figure 5.1 are directed to become inputs at S_1. See Figure 5.2. With these two types of composition any system can be constructed.

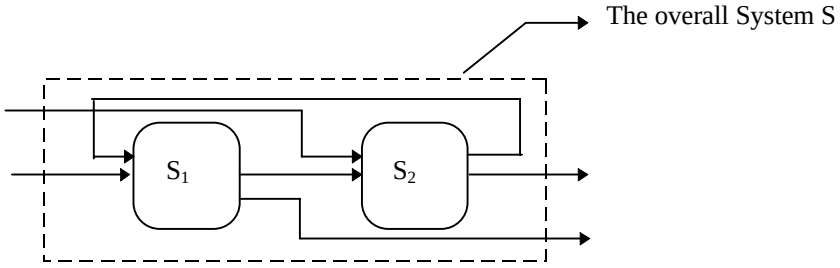


Figure 5.2: Feedback Composition
The difference between cascade composition and feedback composition is that in feedback composition some of S_2's outputs are directed to become inputs at S_1.

5.3.1. Cascade Composition

The definition of composition presented in section 3.4 (page 18) was for general composition. Since we will begin our investigation of composition by examining cascade composition we will formally define the cascade composition operation.

It can be seen from Figure 5.1 that all input events of the first component, S_1, are also input events of the composed system. All of the input events to S_1 are also input events of S. Since all possible inputs can occur for the S_1 component all the traces of S_1 are possible. For the second component (component S_2 in Figure 5.1) not all input combinations are necessarily possible. The outputs of S_1 may not generate all possible input sequences to S_2, therefore, not all traces of S_2 may be possible. This leads us to the following definition of cascade composition.

Definition 5.1: Cascade Composition
Given $S_1=<E_1,I_1,O_1,T_1>$ and $S_2=<E_2,I_2,O_2,T_2>$ that satisfy

$$I_1 \cap I_2 = \varnothing$$
$$O_1 \cap O_2 = \varnothing$$
$$(E_1 \setminus (I_1 \cup O_1)) \cap E_2 = \varnothing$$
$$(E_2 \setminus (I_2 \cup O_2)) \cap E_1 = \varnothing$$

then the compostion of S_1 and S_2 produces a new component S=<E,I,O,T> such that:

$$E = E_1 \cup E_2$$
$$I = I_1 \cup (I_2 \setminus O_1)$$
$$O = (O_1 \setminus I_2) \cup O_2$$

and $T = \{ a \in E^* \text{ such that } a|E_1 \in T_1 \wedge a|E_2 \in T_2 \}$

Notice that the cascade composition is not a symmetric operation. The cascade composition of S_1 and S_2 is not equal to the cascade composition of S_2 and S_1. In this work we will use the convention that component S_1 will be the left component of Figure 5.1 and component S_2 will be the right component.

The definition of cascade composition implies the following:

$$O_1 \cap I_2 = C$$
$$HI = HI_1 \cup HI_2 \setminus HO_1$$
$$HO = HO_1 \setminus HI_2 \cup HO_2$$
$$T_1 = T|E_1$$
$$T_2 \subseteq T|E_2$$

Given components S_1 and S_2 that satisfy properties P_1 and P_2 respectively, we can ask "what property does the cascade composition of S_1 and S_2 satisfy?" The definition of cascade composition yields:

$\forall\tau_1$:traces(S_1)$\cdot P_1(B_{low}(\tau_1,S_1)) \wedge \forall\tau_2$:traces($S_2$)$\cdot P_2(B_{low}(\tau_2,S_2))$ Distributive Laws

$= \forall\tau_1$:traces(S_1)$\cdot \forall\tau_2$:traces(S_2)$\cdot P_1(B_{low}(\tau_1,S_1)) \wedge P_2(B_{low}(\tau_2,S_2))$

traces(S) : *interleave*(traces(S_1),traces(S_2))

$\Rightarrow \forall\tau$:traces(S)$\cdot P_1(B_{low}(\tau|E_1,S_1)) \wedge P_2(B_{low}(\tau|E_2,S_2))$

To proceed further we must substitute the expressions for each of the properties. The resulting expression will indicate the property that the composed system satisfies. A special case of the above is when $P_1=P_2=P$. If after the simplification the composed system can be seen to satisfy P then the property is a component independent property. If the resulting expression does not yield that the composed system satisfies P then the system satisfies a component dependent property. We will continue by demonstrating how to proceed by a series of examples.

The following lemma is useful in determining the effect of cascade composition. It states that if there exists a trace in each component's low level equivalent bunch such

that the communication events are the same and the traces satisfy some predicate then there exists a trace in the low level equivalent bunch of the composite system that satisfies the same predicate.

Lemma 5.1: $\exists t_1{:}B_{low}(\tau|L_1,S_1)\cdot\exists t_2{:}B_{low}(\tau|L_2,S_2)\cdot t_1|C\cap H=t_2|C\cap H\wedge b(t_1,t_2) \Leftrightarrow \exists t{:}B_{low}(\tau,S)\cdot b(t|E_1,t|E_2)$, where b is any predicate

Proof:

$\exists t_1{:}B_{low}(\tau|L_1,S_1)\cdot\exists t_2{:}B_{low}(\tau|L_2,S_2)\cdot t_1|C\cap H=t_2|C\cap H\wedge b(t_1,t_2)$ Definition of B_{low}

$= \exists t_1{:}[\S s_1{:}traces(S_1)\cdot\tau|L_1=s_1|L_1]\cdot\exists t_1{:}[\S s_2{:}traces(S_2)\cdot\tau|L_2=s_2|L_2]\cdot\ t_1|C\cap H=t_2|C\cap H\wedge b(t_1,t_2)$ Definition of §

$= \exists t_1{:}traces(S_1)\cdot\exists t_2{:}traces(S_2)\cdot\ \tau|L_1=t_1|L_1\wedge\ \tau|L_2=t_2|L_2\wedge\ t_1|C\cap H=t_2|C\cap H\wedge b(t_1,t_2)$ Since $t_1|C\cap L = t_2|C\cap L$

$= \exists t_1{:}traces(S_1)\cdot\exists t_2{:}traces(S_2)\cdot\ \tau|L_1=t_1|L_1\wedge\ \tau|L_2=t_2|L_2\wedge\ t_1|C=t_2|C\wedge b(t_1,t_2)$ $t_1|C=t_2|C$ and *interleave*(t_1,t_2):traces(S)

$= \exists t{:}traces(S)\cdot\ \tau|L=t|L\ \wedge\ b(t|E_1,t|E_2)$ Definition of §

$= \exists t{:}[\S s{:}traces(S)\cdot\tau|L=s|L]\cdot b(t|E_1,t|E_2)$ Definition of B_{low}

$= \exists t{:}B_{low}(\tau,S)\cdot b(t|E_1,t|E_2)$ ❑

The first example we present proves that Generalized Noninterference is cascade composable. This result is not new and has already been proven [McLean94] [Zakinthinos & Lee95]. We present it here because Generalized Noninterference has been extensively studied and no work would be complete without demonstrating that it can duplicate known results.

Example 5.1: In this example we will prove that the cascade composition of two components that satisfy the Generalized Noninterference property will also satisfy this property.

$\forall\tau_1{:}traces(S_1)|L\cdot\forall t_1{:}interleave(HI_1^*,\tau_1)\cdot\exists s_1{:}B_{low}(\tau_1,S_1)\cdot t_1=s_1|L_1\cup HI_1\ \wedge$
$\forall\tau_2{:}traces(S_2)|L\cdot\forall t_2{:}interleave(HI_2^*,\tau_2)\cdot\exists s_2{:}B_{low}(\tau_2,S_2)\cdot t_2=s_2|L_2\cup HI_2$
traces(S) : *interleave*(traces(S_1),traces(S_2))

$\Rightarrow\ \forall\tau{:}traces(S)|L\cdot\forall t_1{:}interleave(HI_1^*,\tau|E_1)\cdot\exists s_1{:}B_{low}(\tau_1,S_1)\cdot t_1=s_1|L_1\cup HI_1\ \wedge$
$\forall t_2{:}interleave(HI_2^*,\tau|E_2)\cdot\exists s_2{:}B_{low}(\tau_2,S_2)\cdot t_2=s_2|L_2\cup HI_2$ Distributive Law

$\Rightarrow\ \forall\tau{:}traces(S)|L\cdot\forall t_1{:}interleave(HI_1^*,\tau|E_1)\cdot\ \forall t_2{:}interleave((HI_2\backslash C)^*,\tau|E_2)\cdot\exists s_1{:}B_{low}(\tau_1,S_1)\cdot$
$\exists s_2{:}B_{low}(\tau_2,S_2)\cdot\ s_1|HO_1\cap C=s_2|HI_2\cap C\wedge\ t_1=s_1|L_1\cup HI_1\ \wedge\ t_2=s_2|L_2\cup HI_2$ Lemma 5.1 and Simplification

$= \forall\tau{:}traces(S)|L\cdot\forall t{:}interleave((HI_1\cup HI_2\backslash C)^*,\tau)\cdot\exists s{:}B_{low}(\tau,S)\cdot\ t|E_1=s|L_1\cup HI_1\ \wedge$
$t|E_2=s|L_2\cup HI_2\backslash C$ Cascade Composition and Distributive Law

$= \forall\tau$:traces(S)|L$\cdot\forall$t:interleave(HI*,τ)$\cdot\exists$s:B_{low}(τ,S)$\cdot$ t=s|L$\cup$(HI$_1$ $\cup$HI$_2$\C)

Cascade Composition (HI=HI$_1\cup$HI$_2$\C)

$= \forall\tau$:traces(S)|L$\cdot\forall$t:interleave(HI*,τ)$\cdot$ $\exists$s:B_{low}(τ,S)$\cdot$ t=s|L$\cup$HI

Therefore, the composition of two components that satisfy Generalized Noninterference also satisfies Generalized Noninterference. ❑

Example 5.2: In this example we will determine if Generalized Noninference (section 4.4.1, page 34) is a composable security property. O'Halloran [O'Halloran90] has proven that Noninference is composable and McLean [McLean94] has "proven" using Selective Interleaving Functions that Generalized Noninference is cascade composable. As will be shown McLean's result is wrong. McLean's error comes from the incorrect application of one of his theorems.

$\forall\tau_1$:traces(S$_1$)|L$\cdot\exists$t$_1$:B_{low}(τ_1,S$_1$)$\cdot$t$_1$|HI$_1$=<> $\wedge$ $\forall\tau_1$:traces(S$_1$)|L$\cdot\exists$t$_2$:B_{low}(τ_2,S$_2$)$\cdot$t$_2$|HI$_2$=<>

traces(S) : *interleave*(traces(S$_1$),traces(S$_2$))

$\Rightarrow$ $\forall\tau$:traces(S)|L$\cdot\exists$t$_1$:B_{low}(τ|E$_1$,S$_1$)$\cdot$t$_1$|HI$_1$=<> $\wedge$ $\exists$t$_2$:B_{low}(τ|L$_2$,S$_2$)$\cdot$t$_2$|HI$_2$=<>

Distributive Law

$= \forall\tau$:traces(S)|L$\cdot\exists$t$_1$:B_{low}(τ|E$_1$,S$_1$)$\cdot$ $\exists$t$_2$:B_{low}(τ|L$_2$,S$_2$)$\cdot$t$_1$|HI$_1$=<> $\wedge$ t$_2$|HI$_2$=<>

We cannot progress any further. The high outputs of the first component may not be <>. They may be. However, the property does not guarantee that such a trace exists. Therefore, Generalized Noninference is not a component independent property. With this analysis, however, we can easily determine what conditions are required for it to compose. Since the second component requires the high level communication events to be <> we can require the high level outputs of the first component be <>[6]. Notice that this is Noninference which was proven by O'Halloran to be composable. With this requirement the interface requirement can be satisfied. ❑

In the above example it was easy to see that the composition would not succeed because nothing could be said about the behaviour of the component when the high level input sequence was not <>. In a more complex property this type of observation may not be as obvious. If compatible communication events cannot be guaranteed then the property will not be composable. Therefore, compatible communication events are a

---

[6] Only the high level communication events being <> would suffice.

necessary condition for a property to be component independent. The following steps can be used to determine the conditions that are being imposed on the communication events.

1. For component S_1 write the expression for all traces that look like a trace τ_{low} of S_1.
2. Take any trace *r* that looks like τ_{low} and restrict the trace to the communication events. Note that if the property does not specify a restriction on a particular class of events, for example high level outputs, then it must be assumed that any sequence of events from this class can occur[7].
3. For component S_2 write the expression for all traces that look like a trace τ_{low} of S_2.
4. Take any trace *s* that looks like τ_{low} and list the properties satisfied by the communication events. The same comment as the one is number 2 applies.
5. Compare the sequences generated by 2 and 4. The comparison will indicate what additional restrictions are required.

Example 5.3: We will apply the above procedure to Generalized Noninference.

1. For the first component we know the bunch of traces that look like a trace τ_{low} of S_1 is: $\S t{:}traces(S)\cdot t|L=\tau_{low}|L \wedge HI=<>$
2. Let *r* be any trace of this bunch. Restricting to its communication events it can be seen that $r|L\cap C=\tau_{low}|L\cap C$. Since nothing is specified about the high level outputs we must assume that they can be anything $r|H\cap C:(H\cap C)^*$.
3. Proceeding in the same fashion for the second component: $\S t{:}traces(S)\cdot t|L=\tau_{low}|L \wedge HI=<>$
4. Let *s* be any trace of this bunch. Restricting it to its communication events results in $s|L\cap C=\tau_{low}|L\cap C$. Since some of the high level inputs of S_2 are now communication events $t|H\cap C=<>$.
5. Comparing the communication events found in 2 and 4 we see that the low level events are compatible[8] but the high level outputs are not necessarily compatible.

The conclusions are the same as those found in Example 5.2. ❑

---

[7] This is a pessimistic view. If other information is known about the class then that can be used instead.

[8] This will always be true of security properties.

This type of analysis performed in the previous example is quick and easy but the benefits may not be obvious. The following conjecture will clarify why we believe this analysis is useful.

Conjecture 5.1: If components S_1 and S_2 satisfy security properties P_1 and P_2 respectively, and P_1 and P_2 guarantee compatible communication events, then the cascade composition of S_1 and S_2 will satisfy all properties P such that $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$.

The following example will illustrate how to apply the techniques presented above to determine what property will be satisfied by the composition of two components that each satisfy different properties

Example 5.4: In this example we will determine what property the resulting system will satisfy if component S_1 satisfies Noninference and component S_2 satisfies Generalized Noninterference.

$\forall\tau_1$:traces(S_1)$|L\cdot\exists t_1:B_{low}(\tau_1,S_1)\cdot t_1|H_1=<> \wedge$

$\forall\tau_2$:traces(S_2)$|L\cdot\forall t_2$:interleave(HI_2^*,τ_2)$\cdot\exists s_2:B_{low}(\tau_2,S_2)\cdot t_2=s_2|L_2\cup HI_2$

traces(S) : *interleave*(traces(S_1),traces(S_2))

$\Rightarrow$ $\forall\tau$:traces(S)$|L\cdot\exists t_1:B_{low}(\tau|E_1,S_1)\cdot t_1|H_1=<> \wedge$

$\forall t_2$:interleave(HI_2^*,τ_2)$\cdot\exists s_2:B_{low}(\tau_2,S_2)\cdot t_2=s_2|L_2\cup HI_2$ Cascade Composition

$=$ $\forall\tau$:traces(S)$|L\cdot\exists t_1:B_{low}(\tau|E_1,S_1)\cdot t_1|H_1=<> \wedge \forall t_2$:interleave($(HI_2\backslash C)^*,\tau_2$)$\cdot\exists s_2:B_{low}(\tau_2,S_2)\cdot$

$s_2|H\cap C=t_1|H\cap C \wedge t_2=s_2|L_2\cup HI_2$ Semicommutative Laws

$\Rightarrow$ $\forall\tau$:traces(S)$|L\cdot\forall t_2$:interleave($(HI_2\backslash C)^*,\tau_2$)$\cdot$ $\exists t_1:B_{low}(\tau|E_1,S_1)\cdot$ $\exists s_2:B_{low}(\tau_2,S_2)\cdot$

$s_2|H\cap C=t_1|H\cap C \wedge t_2=s_2|L_2\cup HI_2 \wedge t_1|H_1=<>$ Lemma 5.1

$=$ $\forall\tau$:traces(S)$|L\cdot\forall t_2$:interleave($(HI_2\backslash C)^*,\tau_2$)$\cdot$ $\exists t:B_{low}(\tau,S)\cdot$ $t_2=s_2|L_2\cup HI_2 \wedge t|HI_1=<>$

Specialization with $t_2|HI_2\backslash C = <>$

$\Rightarrow$ $\forall\tau$:traces(S)$|L\cdot\exists t:B_{low}(\tau,S)\cdot$ $t|HI=<>$

Therefore, the composition of the two components satisfies the Generalized Noninference Property.

We will now do the analysis by comparing communication events and applying Conjecture 5.1.

1. For Noninference the high level communication events output from S_1 are guaranteed to be <>.

2. For Generalized Noninterference the high level communication events input to S_2 can be anything.

Clearly compatible communication events can be found. By Conjecture 5.1 the system satisfies a property *P* such that Noninference $\Rightarrow$ *P* and Generalized Noninterference $\Rightarrow$ *P*. From Figure 4.1 (page 39) we see that *P* can be Generalized Noninference. This is the same conclusion demonstrated above.

Notice that in this example the resulting system satisfies a different property than either of its components. We will discuss this further in section 5.4. ❑

Consider a property, P, that is known to be composable. What can be said about a property P_1 such that $P_1 \Rightarrow P$? The cascade composition of two components that satisfy P_1 will satisfy P. This follows because both components satisfy P, which is composable. But does the resulting system satisfy P_1? We believe that it does.

Conjecture 5.2: Given a composable property P and properties P_1 and P_2 such that $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$ then the cascade composition of components S_1 and S_2 that satisfy P_1 and P_2 respectively will satisfy a property Q such that $P_1 \Rightarrow Q$, $P_2 \Rightarrow Q$ and $Q \Rightarrow P$.

This conjecture follows from Conjecture 5.1 because the compatibility of the communication events is guaranteed by the composable property.

Product composition is a special case of cascade composition. Product composition is cascade composition without communication events (see Figure 5.3). All of the above results can be applied to product composition. For product composition Conjecture 5.1 reduces to:

Conjecture 5.3: Given components S_1 and S_2 that satisfy P_1 and P_2 respectively then the product composition of S_1 and S_2 will satisfy a property P such that $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$.

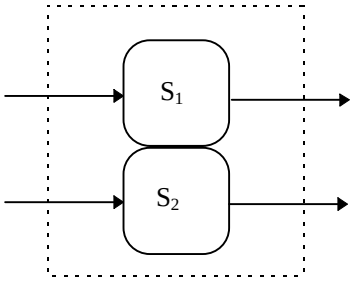


Figure 5.3: Product Composition
Product Composition is cascade composition but without internal events.

5.3.2. Consequences of Input Totality

One of the assumptions in the section Components and Systems (section 3, page 14) was that all of the components must be input total. That is, they must always accept an input. This differs from most other models of event systems. Input totality sounds more restrictive than it is. All that is required is that the input is recorded in the trace. It does not necessarily have any effect on the state of the system; it might be ignored.

Input totality makes the presentation of the cascade results easier. If input totality were not required then it would be possible to find two components such that their cascade composition would not be allowed. Consider the communication events between S_1 and S_2 in Figure 5.1. If the outputs of S_1 were unacceptable as inputs at S_2 or an input event that must occur at S_2 cannot be generated by S_1 then the composition would not succeed. The composition would cause deadlock. Input totality removes this problem.

McLean does not require input totality in his theory of Selective Interleaving Functions [McLean94]. Instead, he has an interface requirement that ensures the composition will succeed. The input totality requirement can be replaced with an interface requirement. This would not change any of our results but would complicate their presentation.

5.4. Emergent Properties

In the previous section we investigated how to determine the effects of composing two components with known security properties in cascade. In this section we will examine emergent properties. As mentioned in section 5.2 an emergent property is one

that is not satisfied by every constituent component but is satisfied by the overall system. One fundamental question that has not been answered is whether emergent security properties exist? As we shall show the answer is yes.

Example 5.4 demonstrated that the composition of a Noninference secure component and a Generalized Noninterference secure component results in a system that satisfied Generalized Noninference. We do not believe that this example proves the existence of emergent properties. In this case the properties of the components both imply Generalized Noninference. In Example 5.2 we demonstrated that Generalized Noninference was not a composable property because it did not guarantee the existence of compatible communication events. However, the existence of the compatible communication events is guaranteed by Generalized Noninterference. Therefore we do not consider this an example of emergence since both components also satisfy Generalized Noninference.

We also want to eliminate from consideration the case where the components satisfy more than one property. For example Generalized Noninterference does not imply nor is implied by Noninference. But a component can be both Generalized Noninterference secure and Noninference secure. In composing two Generalized Noninterference components it might be that the resulting system satisfies Noninference. If both components satisfied Noninference then this would not be a surprise. If one (or both) of the components didn't satisfy Noninference then this would be an example of an emergent property. We will demonstrate that if a composed system satisfies Noninference then each component must also satisfy Noninference.

We will demonstrate that there exists a security property such that two components that both do not satisfy the property when composed result in a system that does satisfy the property. We will not attempt to justify the usefulness of the property. We only want to demonstrate that such properties exist.

Example 5.5: Consider the property EMERGENT:

$$\forall \tau{:}\text{traces}(S)|L \cdot \text{EMERGENT}(B_{low}(\tau,S))$$

$$\text{EMERGENT}(B) \equiv \exists s{:}B \cdot s|HI=\langle\rangle \wedge \neg t|HO=\langle\rangle$$

EMERGENT is Generalized Noninference with the added stipulation that the output sequence cannot be empty. Consider the composition of two components such

that S_1 satisfies Noninference and does not satisfy EMERGENT and S_2 satisfies EMERGENT. Notice that Noninference is neither implied by nor implies EMERGENT. This follows because EMERGENT ensures that the high level output sequence is not empty but Noninference requires it to be empty.

$\forall\tau_1$:traces(S_1)|L·$\exists t_1$:$B_{low}(\tau_1,S_1)\cdot t_1|H_1=<> \wedge$
$\forall\tau_2$:traces(S_2)|L·$\exists t_2$:$B_{low}(\tau_2,S_2)\cdot t_2|HI_2=<>\wedge\neg t_2|HO=<>$
traces(S) : *interleave*(traces(S_1),traces(S_2))

$\Rightarrow$ $\forall\tau$:traces(S)|L·$\exists t_1$:$B_{low}(\tau|E_1,S_1)\cdot t_1|H_1=<> \wedge \exists t_2$:$B_{low}(\tau|E_2,S_2)\cdot t_2|HI_2=<>\wedge\neg t_2|HO=<>$
Distributive Law

= $\forall\tau$:traces(S)|L·$\exists t_1$:$B_{low}(\tau|E_1,S_1)\cdot\exists t_2$:$B_{low}(\tau|E_2,S_2)\cdot t_1|H\cap C=t_2|$ $H\cap C \wedge t_1|H_1=<> \wedge$
$t_2|HI_2=<>\wedge\neg t_2|HO=<>$ Lemma 5.1

= $\forall\tau$:traces(S)|L·$\exists t$:$B_{low}(\tau,S)\cdot t_1|H_1=<> \wedge t_2|HI_2=<>\wedge\neg t_2|HO=<>$ Cascade Composition

= $\forall\tau$:traces(S)|L·$\exists t$:$B_{low}(\tau,S)\cdot t|HI=<> \wedge\neg t|HO=<>$

The resulting system satisfies EMERGENT but both components did not satisfy it. Therefore, there exist emergent properties. This example does not violate Conjecture 5.1 because there does not exist a composable property, *P*, such that Noninference implies *P* and EMERGENT implies *P*[9]. ❑

Now that we have demonstrated the existence of emergent properties we will provide a criterion that allows the system designer to determine if a property might emerge under composition. Before we present the criterion some definitions are required.

Definition 5.2: Event Removing Operator.
Given an event system $S=<E,I,O,T>$ the operation $S\backslash\alpha$, $\alpha\subseteq E$ yields the following system $S'=<E',I',O',T'>$:

$$E' = E\backslash\alpha$$
$$I' = I\backslash\alpha$$
$$O' = O\backslash\alpha$$
$$\text{and } T' = \{\, t \mid t|E'\in T \,\}$$

---

[9] If such a *P* existed then EMERGENT would be a composable property. But, EMERGENT is not composable for the same reasons Generalised Noninference is not composable.

The above operator removes all occurrences of an event from the event system. The following definition will use the above operator to give a condition on properties. We will then show that this condition has an impact on the existence of emergent properties.

Definition 5.3: Stable Property.

A property *P* will be called *stable* if and only if for all systems *S*, $\forall\alpha$:power_set(E)·P(S)$\Rightarrow$P(S\α).

A system that satisfies the stability property is such that removing any number of the events will result in a system that still satisfies the property. How restrictive is the stability requirement? All security properties presented in the literature satisfy this requirement (see Appendix A). The security property EMERGENT of Example 5.5, that was used to demonstrate the existence of emergent properties, does not satisfy the stability requirement. We do not believe that the stability requirement imposes an unduly harsh restriction on security properties. Furthermore, we are not forcing all security properties to satisfy this property. If a desirable property does not satisfy the stability requirement then the previous results can be used to determine composability. Unfortunately, no general comment can be made about how a non stable property might emerge.

Consider the cascade composition of two components S_1 and S_2 such that their composition results in a system *S* that satisfies a property *P*. If the property *P* satisfies the stability requirement then we can conclude $P(S\backslash E_2)$ and $P(S\backslash E_1)$. This follows from the stability condition that any subset of the events can be removed and the property still holds. This result has obvious implications for the ability of the property *P* to emerge under composition.

If a property, *P*, satisfies the stability requirement then a necessary condition for the composition of two components $S_1=<E_1,I_1,O_1,T_1>$ and $S_2=<E_2,I_2,O_2,T_2>$ to yield a system that satisfies *P* is if $P(S_1^*)$ and $P(S_2^*)$ where $S_1^*=<E_1^*,I_1^*,O_1^*,T_1^*>$ and $S_2^*=<E_2^*,I_2^*,O_2^*,T_2^*>$ equal:

$$E_1^* = E_1 \qquad E_2^* = E_2$$
$$I_1^* = I_1 \qquad I_2^* = I_2\backslash O_1$$
$$O_1^* = O_1\backslash I_2 \qquad O_2^* = O_2$$
$$T_1^* = T_1 \qquad T_2^* = T_2$$

This implies that one can determine *a priori* if the composition of two components might result in a system that satisfies *P*. Unfortunately, stability is not a sufficient condition. Generalized Noninference satisfies the stability definition but as demonstrated in Example 5.2 the composition of two Generalized Noninference secure components may not satisfy Generalized Noninference

The stability requirement allows the system designer to determine under what conditions a property may emerge. Consider, Generalized Noninterference. In Example 5.1 it was shown that GNI is a cascade composable security property. Therefore any two systems that satisfy GNI when composed in cascade will result in a system that also satisfies GNI. Since GNI is a stable property (see Appendix A) we can also conclude that the only way a cascade system can satisfy GNI is if the externally visible parts satisfy GNI. By externally visible we mean the system such that all internal events are removed.

5.5. Feedback Composition

In this section we examine systems that contain feedback. Most real systems do exhibit some form of feedback. In the previous sections we have considered security properties in general. In this section we will limit our discussion to a subset of security properties. We do this for two reasons. First, we can provide much stronger results by limiting the class of security properties we consider. Second, the class we are considering encompasses nearly all security properties presented in the literature and appear to contain the best candidates for a formal basis of security.

The class of properties that we will be considering include those that imply the causal variant of GNI. Recall from section 4.4.2.1 on page 36 that the causal variant of GNI requires that all corrections to perturbations occur after the perturbation. In this section when referring to GNI we mean causal GNI. As can be seen from Figure 5.4 most security properties that have been proposed as a basis for the foundations of computer security are included.

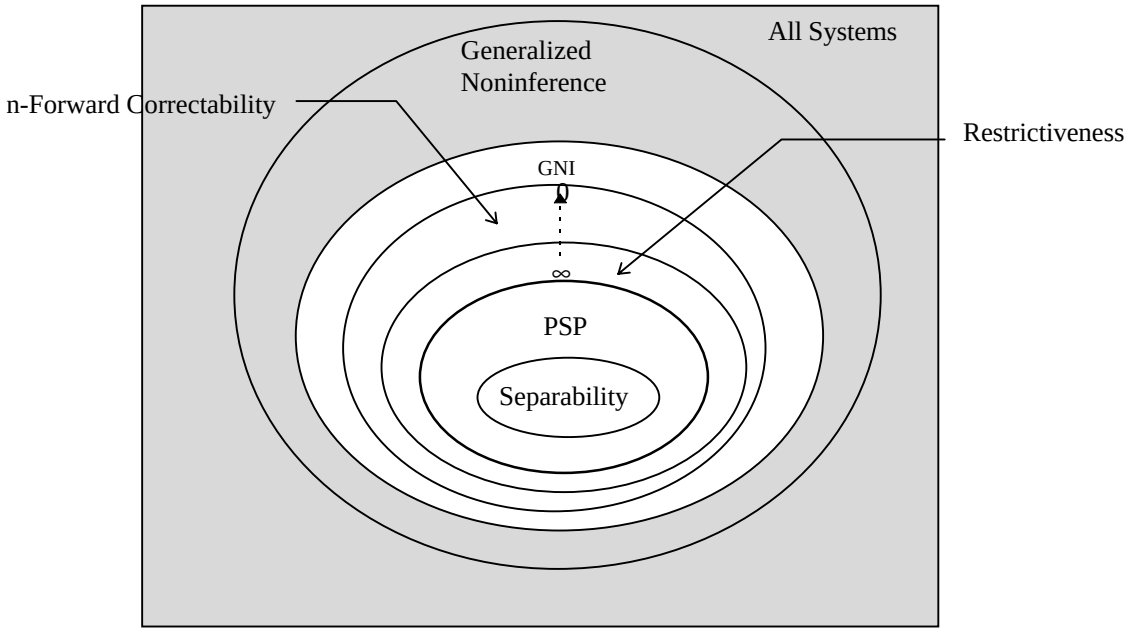


Figure 5.4: Comparison between GNI other Properties.
Generalized Noninterference is implied by most of the security properties presented in the literature.

Until 1987 it was thought that security properties could be composed such that the resulting system also satisfies the property. Then McCullough [McCullough87] [McCullough88] provided an example of two GNI secure systems such that their composition resulted in a system that did not satisfy GNI. We will present a variation of the example McCullough used to motivate our results.

Example 5.6: Machine *A* has one high level input *in*, and one high-level output *out* which is a reply to *in* after some processing. There is a low-level *cancel* input, which cancels any high-level processing that is underway, and a low-level *ack* output that acknowledges the *cancel* input after some time interval. If there is high-level processing at the time of the *ack*, that is, if the cumulative number of *out* events is less than the cumulative number of *in* events, the high-level processing is terminated, and no out will occur until after the next uncancelled *in*. If there is no high-level processing at the time of *ack*, then a low-level *error* output may be produced at some time following the *ack*; however, the *error* output is not guaranteed to occur.

Machine *B* is similar to *A*, but does not have an *ack* output. It cancels high-level processing, if any, at the moment the *cancel* is received. If there is no high-level

processing at the time of the *cancel*, then a low-level *error* output may be produced at some time following the *cancel*; however, the *error* output is not guaranteed to occur.

We will demonstrate that we can compose two components such that the resulting system is not non-interference secure. We will hook up the machines as follows: *A*'s *ack* output feeds into *B*'s *cancel* input, *A*'s *out* feeds into *B*'s *in*, and *B*'s *out* feeds into *A*'s *in*. *A*'s *in* is also available as an external input besides receiving *B*'s *out*. Our model of an event system and composition does not explicitly allow *A*'s *in* event to be both a communication event and an input event. This is not a problem because we can use a multiplexer as describe in section 3.4, page 18. Figure 5.5 shows this interconnection.

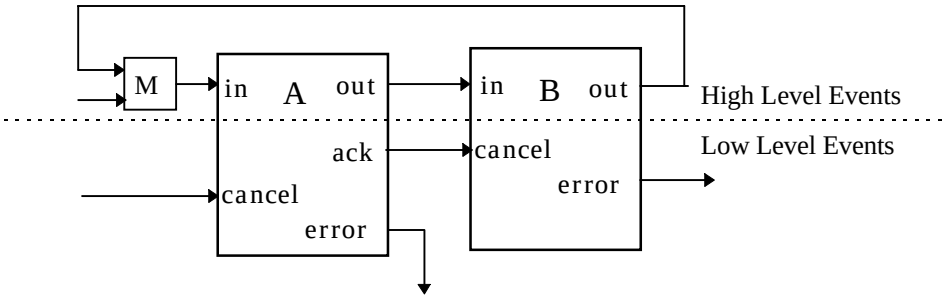


Figure 5.5: Interconnections for Example 5.6**.**
A diagram of the interconnections between the Machine *A* and the Machine *B*. M is a multiplexer required because our model of components does not allow an event to be both an internal event and an input event.

Each trace in Figure 5.6 consists of a time line, running vertically, and events, drawn as labeled arrows along the time line. Time flows *up* the time line; earlier events are nearer the bottom of the time line. Dashed arrows signify high-level events, and solid arrows are low-level events. An arrow directed at the time line is an input event, while an arrow directed away denotes an output event. The composition of Figure 5.5 is not non-interference secure because, for the trace shown in Figure 5.6(a) and the perturbation shown in Figure 5.6(b), there is no correction. Any attempt to correct the trace in one of the components results in the other component requiring a correction also. ❑

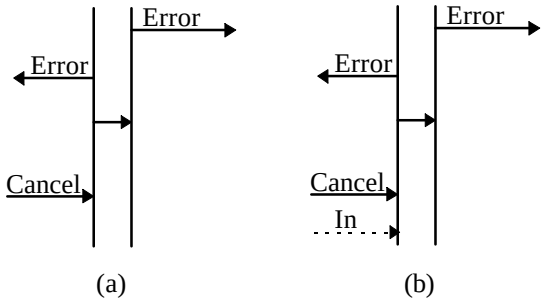


Figure 5.6: Demonstration that the Component of Figure 5.5 does not satisfy GNI**.** Trace (a) is a trace of the composite machine that does not have a correction for the perturbation shown in part (b).

5.5.1. Low Level Preconditions and System State

We have used an event system model for components because it eliminates the need for specifying a model of computation. However, even without specifying a model of computation we can still examine how a system behaves as it processes events. As the system accepts and processes new events the state of the system changes. Events might be dependent on the state of the system. That is the occurrence of an event might be dependent on some condition being true. These conditions are called preconditions for an event. The event system model presented in Chapter 3 does not explicitly contain these preconditions. The preconditions are implicit and embedded in the set of all traces. In proving our results for feedback composition we do not explicitly require these preconditions for each event. However, we will refer to them and therefore deal with the preconditions if they exist.

In this work we are only interested in the preconditions of a low level event that requires a condition on high level state. It might seem strange that low level events can be dependent on high level state but consider the informal definition of precondition given in the preceding paragraph. Since the state of the component is dependent on both the low level users actions in τ and the high level users actions in τ, the condition for λ to occur may be dependent on the actions of high level users.

Definition 5.4: Low Level Preconditions

A low level event requiring condition ϕ to be true for its occurrence means either one (or both) of the following are true:

1. The condition must be true of the high level state for the low level event to occur
2. The condition must be true of the high level state for some future low level event to occur.

The only exception to the above is that a low level input event may never depend on high level state for it to occur.

5.5.2. Theorems on Feedback Composition

Example 5.6 demonstrates that a property that is cascade composable might not be composable in the presence of feedback. The question that has not been addressed in any work on composability is what structures of the system cause this failure to occur. Example 5.6 demonstrates the failure of GNI to compose using two components with feedback. Is it possible to construct such an example with three (or more) components in the feedback path? As we shall prove the answer is no. The only case where feedback composition can fail is in the composition of two components. How this can help the system designer is discussed below.

To prove that the only interconnection that causes feedback composition to fail is in the interconnection of two components, a characterization of why the composition fails is required. The following Lemma proves that the failure is due to a low level event whose preconditions cannot be satisfied. It does not, however, give any indication to which low level event or how the failure will occur. The determination of which event and why it fails is presented below.

Lemma 5.1: For a system composed of GNI secure components, if a trace τ exists such that a perturbation σ has no correction then there exists a low level event such that its conditions for occurring can not be satisfied.

Proof:

Given a trace τ and a perturbation σ such that no correction exists, assume all low level event conditions can be satisfied.

From σ remove all high level non inputs. Apply the following procedure to each low level event beginning with the first low level event in σ.

σ can be written as $\sigma=\alpha\lambda\beta$ where λ is the low level event that we are currently considering. By assumption the conditions for λ to occur can be satisfied. Perform the necessary corrections such that λ can occur in the trace. All remaining corrections can be accomplished after λ.

After all low level events have been handled σ can be written as $\sigma=\alpha\beta$ where β contains only high level events and all required corrections can occur in β. Constructing a valid trace from this point is straightforward. Without a feedback path the system would have a correction to σ (event renaming will be required in general). If this correction is then taken as the new σ and the feedback connections are made again then this will appear as a new perturbation. The correction would not affect any events in α. This procedure should then be repeated until all feedback events are handled. This is a contradiction because it is given that no correction exists.

Therefore, there exists a low level event such that its conditions for occurring cannot be met. ❑

Theorem 5.1: Given a composed system with *k* components that satisfy GNI construct the system graph. If the graph has no 2-cycles then the system satisfies GNI.

Proof: Consider any trace τ of the system and a perturbation σ. By Lemma 5.1 the only way a correction will not exist is if there exists a low level event such that its preconditions can not be satisfied. Consider any low level event, λ, in σ. At worst λ can occur only if all the components to which it is connected to satisfy their respective conditions. Since there is no feedback between any two components that share communication events (there are no 2-cycles) it is always possible to ensure the precondition for λ can be satisfied. This can be accomplished because since there is no feedback between the components they appear in cascade. It has been proven that cascade composition of components satisfies GNI. Therefore, it is always possible to construct a partial correction $\tau'=\alpha\beta$ where β only contains high level events and all corrections can occur in β. The corrections to β can be accomplished by applying the same technique that was used in Lemma 5.1. ❑

Notice that the proof uses a more general definition of composition than the one presented in section 3.4. The definition in section 3.4 only allows an output of one component to be connected to an input of another component. The above proof allows the output to be connected to *n* components. This was done because it is a stronger result. Clearly, this result applies if the definition of composition is that given in section 3.4.

The above theorem allows the system designer to quickly determine if the system under consideration might not satisfy GNI. If the system graph does not contain any 2-cycles then the system satisfies GNI and the system designer is done. The system is guaranteed to satisfy GNI. If 2-cycles exist, then it is possible that the system does not satisfy GNI. Some possible solutions are: 1) Reorganize the system to avoid 2-cycles or 2) insert a dummy component to break all 2-cycles. We will discuss 2-cycles and the dummy component in section 5.5.3.

The above alternatives will work but once again they might cause unnecessary work for the system designer. It is possible that the feedback connection is perfectly safe. What is required are necessary and sufficient conditions for the composition of two GNI secure components to compose.

Theorem 5.2 will give the necessary and sufficient conditions for a security property to be composable. Before presenting the theorem there are some issues that need clarifying. The theorem requires that certain conditions be true at certain times. Since the event system model has no time component this statement may seem strange. The idea in the proof is to capture the notion that two conditions are required to be true simultaneously. The easiest way to capture this notion and to demonstrate that if it isn't true the property would be composable, was to introduce this artificial notion of time into the model. Our use of temporal words is merely for expediency and does not violate the event systems model.

Once again consider the composition that was used to demonstrate the failure of GNI to compose in the presence of feedback (Figure 5.6). McCullough attributed this failure of GNI to compose to the rapid exchange of high level events between the components in the feedback loop [McCullough87]. Lemma 5.1 proved that if GNI fails to

compose it is because the preconditions to some low level event could not be satisfied. This then implies that the rapid exchange of events referred to by McCullough was the attempt by the components to satisfy the preconditions of a low level event. This can be seen from Figure 5.6 by considering the low level trace <*cancel, ack, error*$_1$*, error*$_2$> and the perturbation <*in, cancel, ack, error*$_1$*, error*$_2$>. Each attempt to satisfy the precondition for *ack* in S_1 caused the precondition in S_2's to become false and vice versa.

The most difficult aspect in presenting the necessary and sufficient conditions is to capture the alternation of conditions that was observed in Example 5.6. An exact definition of this alternation would require a model of computation for the components; that is a formal model that can recognize (or generate) the set of traces. One such model has been developed by Nestor [Nestor93]. Since we want the theorem to be as general as possible we do not want to provide a specific model of computation. Also, the alternation of conditions may cause the components to progress through a variety of states in which the alternation may finally stop because the required conditions can be satisfied. For a sufficient complex model of computation determining if the alternation of conditions stops is undecidable[10]. As will be shown the usefulness of this theorem is not diminished without a strict definition of the alternation.

Theorem 5.2: The composition of two GNI secure components S_0 and S_1 will yield a system that is GNI secure if and only if for all low level outputs λ_1 of one component that are connected to low level inputs λ_2 of the other, one or more of the following is false:

1. For event λ_1 to occur requires condition ϕ_1 to be true at the occurrence of λ_1.
2. For event λ_2 to occur requires condition ϕ_2 to be true at the occurrence of λ_2.
3. If ϕ_1 or ϕ_2 become false they cannot both be made true simultaneously.

Proof:

---

[10] It is equivalent to the halting problem.

$\Rightarrow$

We will prove the contrapositive statement. That is, if all of the conditions are false then a correction does not exist.

Since λ_1 and λ_2 are connected rename this composite event to λ.

Let $\tau=\alpha\lambda\beta$ be a trace of the composite system. Where a is a sequence of events λ is the low level event defined above and β is a sequence of events that contains any low level events such that ϕ_1 and ϕ_2 must be true at λ.

Consider a perturbation $\alpha'\lambda\beta$ that makes ϕ_1 false.

Since there is no way to make both ϕ_1 and ϕ_2 true the preconditions for λ cannot be satisfied.

$\therefore$ No correction can be found for the trace

$\Leftarrow$

We will prove each case by contradiction. For each case below assume that for a trace t and a perturbation s no correction exists. If no correction exists then by Lemma 5.1 there must exist a low level event in t such that a perturbation has meant that the precondition for it cannot be satisfied. Assume that the event whose condition can not be satisfied is *f*.

Case 1: *f* is not a communication event.

Assume *f* is an event of component S_0. The following is applicable if *f* is an event of S_1 by changing the appropriate subscripts. Since *f* is not a communication event any correction to *f* need only effect events in S_0. The addition or removal of events might involve communication events. These will appear as a perturbation to S_1. The corrections for these perturbations can be postponed until after *f*. This leads to a contradiction that the condition for *f* could not be satisfied.

Case 2: One (or both) of the conditions required for the low level event to occur are not satisfied at the occurrence of the event but some time before it.

Assume f_1 is to be evaluated before the occurrence of *f*. Once again with the appropriate changes this is applicable to the other case. The correction to make ϕ_1 true must occur before the point ϕ_1 will be evaluated. This correction may make ϕ_2 false but the correction for this can wait until after ϕ_1 is evaluated but before *f*. When ϕ_2 is corrected ϕ_1 may have been made false but ϕ_1 has already been evaluated and hence *f* can occur. All remaining corrections can wait until after *f*. This lead to a contradiction that the condition for *f* could not be satisfied.

Case 3: If ϕ_1 or ϕ_2 is made false it is always possible to make ϕ_1 and ϕ_2 true simultaneously.

The existence of a trace follows immediately since the conditions for *f* are guaranteed to be satisfied.

Since for each case we have reached a contradiction if a trace does not have a correction then the conditions of the theorem must hold. ❑

The system designer now has all the tools required to determine if a system composed of GNI secure components is GNI secure. The following procedure can be used to determine if the system is GNI secure.

1. Construct the system graph.
2. If the graph has no 2-cycles then the system is GNI secure.
3. For each 2-cycle examine the low level connections to see if the low level output event of one component must satisfy a condition at its occurrence while the low level input event of the other must also satisfy a condition at its occurrence.
4. If no such case exists then the system is GNI secure.
5. For all cases that do exist ensure that the conditions required for the low level events can be satisfied.

The checking to ensure that the alternation of conditions does not occur may not be trivial in a complex component. Therefore, we suggest that if it is discovered that the

composition of two low level events requires two conditions to be satisfied for the event to occur, a dummy component be inserted in the feedback path.

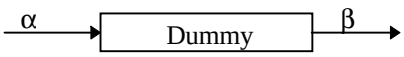
5.5.3. Why Dummy Components?

In this section we wish to investigate why a three cycle in the system graph is sufficient for the system to satisfy GNI. Specifically, why does inserting a dummy component, a component that does nothing except copy its inputs to its outputs, ensures that the system satisfies GNI?

The definition of composition requires that an output of one component immediately becomes an input at the other. Inserting another component between the two components allows other system events to occur between the output event of one component and the corresponding input event. One may argue that the timing of events in the system is important. If the dummy component copies the event as quickly as it can then no other system event would be able to occur. This might be true but since GNI is a possibilistic property the dummy component provides the possibility of a correction. That is not to say that in a system with processing delays the system will have a correction. However, it can be argued that any possibilistic property has this problem since processing delays are not considered in the models of the properties. This leads to the situation where a system is pronounced secure but really is not secure. The pros and cons of possibilistic properties are beyond the scope of this work.

Another way to view the effects of the dummy component is that it breaks the synchronization between the components it is connected to. The definition of composition only allows for synchronized communication. Since the composition with the dummy component allows other system events to occur between the output and the corresponding input event it can be viewed as non-synchronized communication.

As mentioned in section 3.4 both forms of communication might be required. If non-synchronized communication is required, then the component depicted in Figure 5.7 can be used. The benefits of using this form of communication are obvious. If all components use this form of communication and all components satisfy GNI, then the whole system satisfies GNI.



S_D=<{α,β},{α},{β},{αβ}>

Figure 5.7: A Component that can be used to Model Non-Synchronized Communication.

5.5.4. Emergent Properties in the Presence of Feedback

In section 5.4 we discussed the conditions under which a property may emerge under composition. In that section it was shown that if the property satisfied a stability requirement then very specific predictions could be make about its emergent potential. The necessary condition presented in that section applies here also. The externally visible portion must satisfy the property. In the feedback case, however, it is much more difficult to ensure that the communication events are compatible.

5.5.5. Why Certain Properties Compose

With the necessary and sufficient conditions for GNI to compose, some insight can be gained into why certain properties and techniques ensure that the composition of two components preserves the property. This insight will give system designers and researchers a better understanding of what the security property is attempting to accomplish.

After McCullough demonstrated that GNI was not a composable security property he proposed a property he called Restrictiveness. Restrictiveness is GNI with the added stipulation that high level outputs cannot be fixed up arbitrarily soon after a modified high level input. High level output correcting must wait until any immediately following low level inputs. Examining the Restrictiveness condition on GNI it can be seen why it composes. Restrictiveness ensures that low level inputs never require a condition to be true for future low level events to be possible[11]. The condition of the theorem above

[11] The possibility of a low-level input can never be dependent on the high level state. The only condition possible is for future low level events to be dependent on the high level state at the occurrence of the low level input.

requires that an input event have a condition associated with it. Since Restrictiveness does not allow such a condition it is composable.

Johnson and Thayer's [Johnson & Thayer88] n-Forward Correctability composes for the exact same reason as Restrictiveness. This should not be a surprise since Restrictiveness and n-Forward Correctability are so similar. The only difference being that while Restrictiveness must wait until after all low level inputs following a perturbation to begin a correction n-Forward Correctability need only wait *n* low level input events. Notice that 1-Forward Correctability is the weakest condition of any property that solely eliminates the possible of there being a condition on a low level input event. This follows immediately from the theorem because the synchronization affects at most one low-level event at a time.

Both Restrictiveness and n-Forward Correctability rely on eliminating all components that require input conditions on the low level events. With Theorem 5.2 it is possible to construct many security properties that are composable. It is also possible to construct a composable security property stronger than 1-Forward Correctability. GNI secure components can be used, but in interconnecting them a check must be done to ensure that two low level events are not connected such that each satisfies a condition at the time of its occurrence. This property enforces the same rule (albeit differently) as 1-Forward Correctability but only in the cases where a problem may exist.

Zakinthinos and Lee [Zakinthinos & Lee95] proposed a technique that allows GNI secure systems to be composed with feedback. This technique is based on the use of non-synchronized communication for communication events. This is modeled by using a delay component in the feedback path. The reason why this technique works follows from Theorem 5.1. Since the delay component is inserted in the feedback path, the feedback path contains three components. Theorem 5.1 also gives guidance as to how long the delay should be. In Zakinthinos and Lee's paper a feedback event must be delayed until the next low level event. Theorem 5.1 indicates that any (even a fixed) delay is sufficient.

5.6. Summary and Conclusions

We began this chapter by considering how to determine the result of composing two components with known security properties. We demonstrate how to use our formalism to determine what property the resulting system satisfies. Then we considered under what conditions a property may emerge under composition. We showed that if the property satisfies a stability requirement then specific predictions can be made about the emergence of the property.

After considering cascade composition we examined feedback composition. We proved that the only structures of a system that can cause Generalized Noninterference to fail are those that involved feedback between two components. We then presented necessary and sufficient conditions for Generalized Noninterference to compose with feedback. These results were then used to analyzed why certain properties and techniques were composable and others were not.

6. Comparison To Selective Interleaving Functions

Science moves, but slowly slowly, creeping on from point to point.
Alfred, Lord Tennyson *(1809-1892)*
English Poet.

6.1. Introduction

The only other general framework for the specification and analysis of security properties is McLean's Selective Interleaving Functions. In this chapter we compare our framework to McLean's. Specifically, we will compare the expressability of the two frameworks and the results one can obtain from each.

McLean defines a framework for the analysis of Selective Interleaving Functions (SIF). Selective Interleaving Functions can be used to express properties that are an interleaving of two traces of the system. The justification for using SIFs is McLean's observation that certain security properties are "a closure property with respect to some function that takes two traces and interleaves them to form a third trace" [McLean94]. If a security property cannot be so expressed, then the results of McLean's work are not applicable. It may be argued that if the security properties that cannot be handled by SIFs are "uninteresting" then SIFs are all that is required. We will show that at least one of the security properties presented in the literature cannot be handled by SIFs.

Definition 6.1 defines SIF. Our definition is different than McLean's because our model of components is different then McLean's.

Definition 6.1: Selective Interleaving Functions

Let $S=<E,I,O,T>$ be a component. Partition the set of input events I into m disjoint subsets. I_x will be used to refer to the x^{th} subset. Similarly, partition the set of output events O into n disjoint subsets. O_x will be used to refer to the x^{th} subset. Let $i \in \{0,1,2\}^m$, and $j \in \{0,1,2\}^n$, the notation i[x], j[x] will be used to refer to the x^{th} coordinate of i or j

respectively. A function f: $T \times T \rightarrow T$ is a *selective interleaving function* of type $F_{i,j}$ if and only if $f(t_1,t_2) = t$ implies:
for all x such that i[x] = 1 : $t|I_x = t_1|\alpha I_x$,
for all x such that i[x] = 2 : $t|I_x = t_2|\alpha I_x$,
for all x such that j[x] = 1 : $t|O_x = t_1|\alpha O_x$ and
for all x such that j[x] = 2 : $t|O_x = t_2|\alpha O_x$.

The definition of SIFs is intended to be general and encompasses more than security properties. This can be seen from examining the partitioning of *I* and *O*. The definition allows an arbitrary partitioning. In the case of security properties this partitioning will be into high and low level event classes. In demonstrating that SIFs can be expressed in our framework we will partition *I* and *O* into two sets, the set high level events and the set of low level events. This partitioning is also used by McLean in demonstrating the use of SIFs.

6.2. Comparison of Expressability

We will now consider the expressability of our framework versus SIFs. First, consider the following examples of the Separability property (section 4.4.4) as defined in our framework and McLean's:

$$\forall\tau\text{:traces(S)}|L\cdot SEPARABILITY(B_{low}(\tau,S))$$
$$SEPARABILITY(\ B\ ) \equiv \forall t\text{:traces(S)}|H\cdot \textit{interleave}(t,\tau_{low})\text{:B}$$

vs.

$F_{<1^H:2^L>,<1^H:2^L>}$: $T \times T \rightarrow T$, which expands to
interleave: $T \times T \rightarrow T$, such that interleave(t_1,t_2) = t implies
highin(t) = *highin*(t1)
lowin(t) = *lowin*(t_2)
highout(t) = *highout*(t_1)
lowout(t) = *lowout*(t_2)
if a system is closed under *interleave* then it satisfies Separability

Expressing the property as a SIF ($F_{<1^H:2^L>,<1^H:2^L>}$) requires the system designer to take many steps before he arrives at such a definition. Also, the intent of the property is not clear when expressed as an SIF. The intended property is clearer, however, in the expanded version. We believe that neither form captures the notion of a security property.

We believe our formalism, which is based on what the low level user can infer from an observation, is more natural.

We will now demonstrate that SIFs can be expressed in our formalism. We will do this for the subset of SIFs that have the *lowin* and *lowout* from one trace and the *highin* and *highout* from the other. All of the security properties examined by McLean have this property. Furthermore, it is not clear how the events can come from different traces and still have a useful system.

Given a SIF of type $F_{<x^H:2^L>,<y^H:2^L>}$ x, y∈ {0,1} consider the following property:

$$\forall\tau\text{:traces(S)}|L\cdot P(B_{low}(\tau,S))$$
$$P(B) \equiv \forall t\text{:traces(S)}|Z\cdot \textit{interleave(}t,\tau_{low}\textit{)}:B$$

Z	x	y
∅	0	0
HO	0	1
HI	1	0
H	1	1

The implication of the above is that only 4 types of security properties can be achieved using SIF. This is partially true. To achieve others the domain of the interleave can be restricted. For example, Generalized Noninference is the SIF $F_{<1^H:2^L>,<1^H:2^L>}$ restricted to the domain $\{<>\} \times T$. Also, as mentioned above the definition of SIF is more general then that considered here. The partitioning can be more complex then the simple high and low level events. However, all these extensions can be handled in much the same way as above.

The above result demonstrates that all security properties that SIFs can express can also be expressed in our framework. We will demonstrate that there are other security properties that cannot be expressed using SIFs but can be using our framework. The argument that all "interesting" security properties can be expressed is naïve. First, we will show that SIFs cannot represent all security properties presented in the literature. What is considered interesting today may not be interesting the future. A framework should not place limits on what types of properties can be expressed.

Consider Guttman and Nadal's Output Non-Deductibility (section 4.4.3, page 36) reproduced here:

$$\forall\tau\text{:traces(S)}|L\cdot NDO(B_{low}(\tau))$$

$$NDO(B) \equiv \forall t:traces(S)\cdot t|LI=\tau_{low}|LI \Rightarrow \exists s:B\cdot s|(H\cup LI)=t|(H\cup LI)$$

Informally, for every pair of valid traces $a,p \in T$, if event sequence $s \in E^*$ satisfies

$$s|L = a|L$$
$$s|(H\cup LI) = p|(H\cup LI)$$

then *s* is a trace.

This property appears that it should be possible to fit into the SIF format since the final trace *s* has all low level events from one trace and all high level events from the other. A problem, however, is that the interleaving is dependent on the position of low level input events. This type of condition cannot be expressed in SIFs.

Another example of a property that can be expressed in our formalism but cannot using SIFs is the PSP property presented in section 4.3.1, page 30. Recall that PSP is similar to Separability but allows high level outputs to be dependent on low level events. This type of dependency is not expressible using SIFs.

6.3. Comparison of Results

McLean examines three types of composition. Product, cascade and feedback. In this work we consider product composition a special case of our definition of cascade composition. McLean's definition of cascade composition is different than ours. In his definition of cascade composition all of the output events of the first component and all input events of the second are involved in the composition. In this definition the only inputs to the system are the inputs to the first component and the only outputs are the outputs from the second component. To achieve the type of cascade composition that we describe, identity components must be introduced by McLean. An identity component copies its inputs to its outputs. Figure 6.1 demonstrates this type of composition. McLean calls this type of composition general cascade composition.

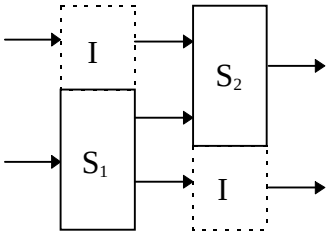


Figure 6.1: General Composition
The two identity components (I) are first composed with S_1 and S_2 using product composition. Then cascade composition is used to connect the composed S_1 and S_2 components. In this fashion general cascade composition can be achieved.

McLean's results on product and cascade composition are, not surprisingly, similar to our own. The main difference is our assumption of input totality. McLean requires the *a priori* knowledge of compatible traces for the composition. Our assumption of input totality removes this requirement. See section 5.3.1 for further discussion of the input totality assumption.

Next McLean considers Feedback composition. McLean's definition of feedback is more restrictive than ours. McLean defines an interface condition which is essentially the requirement that both components involved in the feedback agree on the timing of the events. We feel that this requirement is overly restrictive and requires work by the system designer to determine if the required condition exists.

6.4. Summary

McLean's SIFs were a step forward in defining a general framework for the expression and analysis of security properties. However, they can only be used to analyze a subset of possible security properties. One of the greatest weakness, however, is that if the application of the theorem yields that the composition of two components will not satisfy the desired property, the result yields no information as to why.

7. Implementation Issues

> All theory, dear friend, is gray but the golden tree of actual life sprints evergreen.
>
> ***Johann Wolfgang von Goethe*** *(1749-1832)*
> *German poet, novelist and playwright.*

7.1. Introduction

The previous chapters dealt with the issue of composability. With the tools provided the system designer can quickly determine if a property is composable. If it is not composable then he can determine why. What is missing is a procedure or mechanism for the system designer to determine if a component satisfies a particular property.

The framework we presented in chapter 3 was a trace based formalism for components. This formalism allows for the easy expression of security properties but makes the procedure of validating a component more difficult. While some systems may be specified directly in terms of their possible traces (see [Zwiers & Roever89] or [McLean92a] for an example), program code and other formal specification approaches assume a state-transition model and specify individual state transitions [Millen94]. Therefore, it would be helpful if we could transform our trace based approach to security to a condition on individual states. A theorem stating the equivalence of a trace-based security condition with a transition-based security condition is called an unwinding theorem.

To date unwinding theorems have only dealt with specific trace based security specifications [Goguen & Meseguer84] [McCullough90] [Bevier & Young94] [Millen94]. No general approach to constructing an unwinding theorem has been presented. In this chapter we will present an approach to construct an unwinding theorem for a class of security properties.

7.2. Event System Acceptors

In this work we have not relied on any model of computation to generate or recognize the traces of an event system. Thus our composition theorems can be applied to any model of computation. In presenting an unwinding theorem we need to fix the model of computation. All unwinding theorems in the literature have used some form of state machine. The differences between the various models are whether the machines are non-deterministic and whether they must have a finite number of states. A state machine with an infinite number of states may be of theoretical interest but analyzing such a machine is often impractical. Therefore, we have chosen to use non-deterministic finite state automata (NFA) for our model of computation. Using NFAs as the model of computation does limit the class of machines that can be analyzed but we believe that the usefulness of having an understanding of unwinding theorems out weighs any limitation. The analysis of more general models might be possible.

Millen [Millen94] has shown the equivalence of an infinite state machine and an event system as defined in chapter 3. We will also show this equivalence and as corollary how a NFA can be used to represent event systems. The main difference between our construction of the state machine and Millen's is that Millen's machines may require an infinite number of states even though our construction may not. Our construction will never require more states than Millen's. The difference in number of states is a result of Millen's desire to prove the existence of the state machine rather than finding a compact representation.

A NFA defines a language over its symbols. We will interpret this language as the set of traces of a system. To be able to use NFAs as a model of computation we must demonstrate their equivalence to a class of event systems.

We begin by showing how to construct a possibly infinite state machine for a given event system. Given an event system S={E,I,O,T} consider the following equivalence relation (see Appendix B for a proof that it is an equivalence relation) on the set of traces *T*.

$$s \in T,\ t \in T \quad s \equiv t \text{ iff } \forall r:E^{*} \cdot s^\frown r \in T \Leftrightarrow t^\frown r \in T \tag{7.1}$$

The above relation partitions the set *T* into equivalence classes such that two traces are in the same class if and only if they share the same possible futures. These equivalence classes correspond to the states of the event system. An informal argument for this is as follows: Assume the execution of all the traces of an equivalence class arrive at *m* different states such that all possible futures are the same. Since all *m* states have the same possible futures they are indistinguishable and can be considered one state [Wood87, pg128]. The above relation will result in a finite number of equivalence classes if *T* is finite. If *T* is infinite then the number of equivalence classes may be infinite. The number of equivalence classes is finite for the class of machines in which we are interested[12].

The events of an event system are partitioned into two classes. These are the high level events and the low level events. Security policy typically requires that low level users can see only low level events, while high level users can see all events. This classification can be applied to the set Σ of the NFA. When such a classification is used we can also define the views of each user level.

A user's view of the system is that portion of the system state that *could* potentially influence his activities [Bevier & Young94]. The unwinding technique presented below constructs the view of the low level user and the unwinding conditions give the conditions on how high level activity can influence the low level view. An unwinding theorem can be thought of as giving conditions on how high level users can influence a low level user's view.

The following definitions will be central to our presentation of a a state machine that can be used to represent certain classes of event systems.

The projection operator is used to determine the possible low level futures from a state. There are two versions of this operator. The one in Definition 7.1 will be used in those properties where high level outputs can be inserted to ensure that a correction for a perturbation exists (see section 7.3). The version given in Definition 7.2 will be use for those properties were high level outputs cannot be inserted to ensure a correction exists.

---

[12] The existence of event systems that require an infinite number of states follows immediatly from the existstance of languages that can not be recognized by a NFA.

An algorithmic approach to determining the projection of an NFA can be found in [Ginzburg68].

Definition 7.1: Projection Operator.
The projection of a state[13], *q*, where high level outputs do not affect the projection written $\pi(q)$ is the set of possible low level futures from *q* :

$$\pi(q) = \{ t \mid s \in q \wedge s\hat{}t \in T \wedge t|HI=<>\}|L$$

Definition 7.2: Projection Operator II.
The projection of a state, *q*, written $\pi'(q)$ is the set of possible low level futures from *q*:

$$\pi'(q) = \{ t \mid s \in q \wedge s\hat{}t \in T \wedge t|H=<>\}|L$$

Definition 7.3: The After Operator.
The *after* operator returns the tail of a trace after the execution of its prefix. Formally, If $r \subseteq E^*$ is a set of event sequences and $\sigma \in E^*$

$$r/\sigma = \{ p \mid \sigma\hat{}p \in r \}$$

Definition 7.4: λ Events
λ is an event with the following behaviour:

$$r/\lambda = r \qquad \text{For all r.}$$

The following definition describes the equivalence between event systems and an non-deterministic infinite state automata. For the purposes of this work we will only consider state machines that have a finite number of states.

Definition 7.5: Event System Acceptor.
An event system acceptor for an event system S={E,I,O,T} is a quintuple M={Q,Σ,δ,s,F} where:

Q is an alphabet of the state symbols. Each state will correspond to a different equivalence class generated by relation 7.1.

---

[13] By state we mean an equivalence class. This is formalized in Definition 7.5.

Σ is an alphabet of symbols for the NFA. This corresponds to the set of events *E*. Therefore, for each member of Σ there is a corresponding unique member of E.

$\delta \subseteq Q \times (\Sigma \cup \lambda) \times Q$ is a transition relation such that (q,e,q') if and only if $\pi(q)/e = \pi(q')$. The transitions are called *moves* of the state machine.

s in Q is the start state

$F \subseteq Q$ is the set of final states.

In this work the set of final states *F* will be equal to the set of states Q. This implies that the set of traces is prefix closed. The prefix-closedness is consistent with the intuitive interpretation of an event sequence as a temporal ordering of events [Lee et al. 92]. The implication of this is that there are no inseparable events. That is, there is never a state where the system must wait exclusively for a particular event to occur. The assumption of prefix-closure simplifies the presentation of the results. The extension to a non prefix-closed set of traces is straightforward.

Theorem 7.1 If M is the event system acceptor for S, M accepts σ iff $\sigma \in T$.

Proof:

$\Rightarrow$ M accepts σ if $\sigma \in T$

We will prove this by induction on the length of sequences produced by the state machine.

Base Case:

The zero length sequence corresponds to the empty trace. The empty trace is a trace of all event systems.

Induction Hypothesis:

A sequence σ of length *n* is a trace of S.

Induction Step:

Consider a sequence σe where σ is a sequence of *n* symbols and *e* is a possible event after σ. Since σ is a trace of S the relation 7.1 above places σ in an equivalence class. The event *e* is possible iff there is a trace σ' such that $\sigma e \equiv \sigma'$.

Therefore, by induction if M accepts σ then $\sigma \in T$

$\Leftarrow$ if $\sigma \in T$ then M accepts σ

The proof is similar to the above and follows from the construction of M. ❑

7.3. Security Properties

The definition of a security property given in Chapter 4 imposes few restrictions on the form of a property. This lack of structure makes providing a general transformation routine from security property to unwinding theorem difficult. In this work we will provide a set of rules that can be used to transform a class of security properties into unwinding conditions.

To simplify the presentation of the results we will split the unwinding theorems into three classes. The first class we will consider are those expressible in the N-Forward Correctable Hierarchy and Generalized Noninterference. We will define N-Forward Correctability below. We will then show how the unwinding theorem for N-Forward Correctability can be applied to PSP. Finally, we will demonstrate that a special case of the unwinding theorem can be used to handle Noninference and Generalized Noninference. The unshaded area of Figure 7.1 shows the classes of security properties we will provide unwinding theorems for.

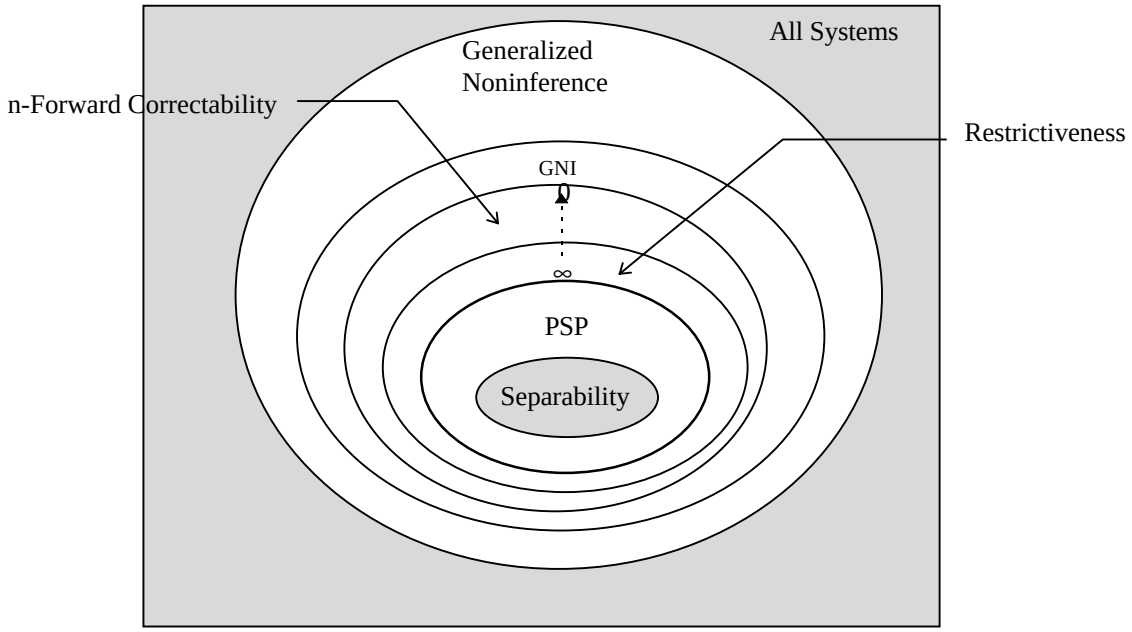


Figure 7.1: The Class of Properties Our Unwinding Theorems Cover.
The unshaded area shows the classes of security properties that our unwinding theorems can handle.

It may seem strange that Separability is not included in these theorems. Our unwinding theorems will be expressed in terms of states that are reachable from any given state. Separability requires all high level traces to be possible with all low level traces. This is an expression over the whole state machine and cannot be expressed in terms of reachable states from a state.

7.4. Unwinding Theorems

Ideally, an unwinding theorem has two properties:

1) The required conditions are an expression between a state and those one move away.
2) The system satisfies the desired property if and only if the unwinding condition is true for all states.

It is not always possible for the unwinding theorem condition to be expressed between adjacent states. For example, the unwinding theorem given by Goguen and Meseguer for non-interference can be expressed between a state and all adjacent states. Millen's unwinding theorem for 1-Forward Correctability, cannot be so expressed. We will not restrict ourselves to those unwinding theorems that are expressible between

adjacent states. We will, however, only want unwinding theorems such that if the conditions are true on all states the system does satisfy the property and conversely. This will ensure our unwinding theorems are sound and complete.

7.5. Unwinding Theorem for GNI and N-Forward Correctability

In this section we will provide an unwinding theorem for Generalized Noninterference and N-Forward Correctability. Before we present the unwinding conditions we will define these properties in terms that will simplify the proofs of the theorems. The following definitions are due to Millen [Millen94].

Definition 7.6: Simple Perturbation.

δ is a *simple perturbation* of τ *before* γ if there exists a high level input event x such that for some β:

1. $\tau=\beta\gamma$ and $\delta=\beta x\gamma$ (x is inserted into α before γ)
2. $\tau=\beta x\gamma$ and $\delta=\beta\gamma$ (x is deleted from α before γ)

Definition 7.7: Correction.

τ' is a *correction* of δ in γ if $\delta=\phi\gamma$ and $\tau'=\phi\gamma'$ such that $\gamma|(L\cup HI)=\gamma'|(L\cup HI)$.

Thus, a correction in γ is a modification of high level non-inputs in γ. We are now ready to define N-Forward Correctability in the above terms.

Definition 7.8: N-Forward Correctability.

An event system, S=<E,I,O,T> is N-Forward Correctable if for all traces $\tau\in$ T and for all $\alpha\in LI^n$, if δ is a simple perturbation of τ before either γ or $\alpha\gamma$, and $\gamma|HI=<>$ (γ contains no high level input events), then there exists a correction τ' of δ in γ such that $\tau'\in$ T.

We note that ∞-Forward Correctability is Restrictiveness and 0-Forward Correctability is forward correctable Generalized Noninterference. We will now prove (in Theorem 7.3) that the non-forward correctable GNI is equivalent to the forward correctable version in deterministic systems. That is, a correction can only occur before a perturbation in a non-deterministic system. Furthermore, we will demonstrate how to

transform the NFA into one such that the forward correctable definition of the security property can be applied.

7.5.1. Forward Correctable versus Non-Forward Correctable GNI

The non-forward correctable version is more difficult to handle than the forward correctable one. For the forward correctable one only the possible futures need to be considered. If the system designer is evaluating state *q* then only the states reachable from *q* need to be considered since a correction to a perturbation can only affect future events. In a non-forward correctable system a change to a trace may change the path through the state machine. Therefore, if one is considering state *q* a change in the path from the start state may result in the system being in a state *q'*, *q≠q'*. The following theorem proves that we can always transform the state machine into one which causal techniques apply.

Theorem 7.2: Given a NFA, M, by

1. replacing all high level output transitions with λ transitions,
2. transforming the NFA to remove λ transitions
3. transforming the NFA to eliminate non-determinism

will result in a deterministic finite automata (DFA), M', such that the causal version GNI may be used to evaluate the system.

Proof:

In M replace high level output transitions with λ transitions and eliminate the λ transitions and the non-determinism. Call this new DFA M'. The transformation can be accomplished and does not change the language of M [Wood87 pg. 118]. We must prove that if M' satisfies the causal version of the property then M satisfied the non-causal version.

Let τ be a trace that will place the system in state q∈Q (of M) such that for a simple perturbation after τ a correction would modify events in τ. After the modification of the events in τ M will be in a state q'. By definition only High Level Output events may be changed. Consider the effects on the states q and q' by replacing in M all High Level Output transitions with λ transitions. The projections from the two states must be equal by definition. If q=q' then we are done since eliminating non-determinism will not affect this equality. If q and q' are different then there exists a non-deterministic choice that causes the path through the state machine to diverge. If this were not the case then the

execution of τ (after replacing the High Level Output transitions with λ transitions) would be in state q. Transforming the NFA to a DFA will result in the non-determinism being removed. Since there is no longer any non-determinism the execution of τ (after replacing High Level Output transitions with λ transitions) will place the system in state q=q' and only future events from q need be considered. ❑

Theorem 7.2 demonstrates that a non-causal property can be transformed into a causal one by eliminating λ transitions and transforming the NFA to a DFA. If the property is causal then the elimination of the λ transitions could be done. It is not required since the calculation of the projection from a state will implicitly remove them. However, the transformation from a NFA to a DFA must *not* be done. In a causal system the nature of the non-determinism is important. This will be demonstrated below. If the non-determinism is eliminated then the property is transformed into its non-causal equivalent.

7.5.2. Unwinding Theorems

Theorem 7.3: If *M* is the event system acceptor for the event system *S*, *S* is N-Forward Correctable if and only if

$$\forall x{:}HI \cdot \forall \alpha{:}LI^{n} \cdot \forall q{:}Q$$

$$\pi(q)=\pi(q/x) \text{ and } \pi(q/y)=\pi(q/x/\alpha)$$

Proof:

There are four parts to this proof corresponding to the two directions of the implication and either the two state equivalencies above, or whether or not the low inputs are present before γ in the definition of N-Forward Correctability.

Suppose *S* is N-Forward Correctable. Assume $x \in HI$ and $q \in Q$. We will show that $\pi(q)=\pi(q/x)$ in two steps. First we will show that $\pi(q/x) \subseteq \pi(q)$. Let $\gamma \in \{ t \mid s \in (q/x) \wedge s\hat{}t \in T \wedge t|HI=<> \}$. By definitions there exists a σ such that $\sigma x \gamma \in T$. So $\sigma\gamma$ is a simple perturbation of $\sigma x \gamma$ before γ. Also γ has no high level input events. By N-Forward Correctability there exists a γ' such that $\sigma\gamma' \in T$ and $\gamma'|L \cup HI = \gamma|L \cup HI$. Therefore $\pi(q/x) \subseteq \pi(q)$. The proof that $\pi(q) \subseteq \pi(q/x)$ are identical to this one since inserting an event

is also a simple perturbation. The proof that $\pi(q/y)=\pi(q/x/\alpha)$ is the same as above. The same proof methods are used.

Assume $\pi(q)=\pi(q/x)$ and $\pi(q/y)=\pi(q/x/\alpha)$. Consider a trace $\tau \in T$ such that $\beta x\gamma \in T$ where $\gamma \in \pi(q/x)$, by definition $\gamma|HI=<>$. From the equivalence $\pi(q)=\pi(q/x)$ and $\pi(q/y)=\pi(q/x/\alpha)$ there exists a $\gamma' \in \pi(q)$ such that $\beta\gamma' \in T$. Therefore, $\beta\gamma$ which is the simple perturbation of $\beta x\gamma$ has a correction $\beta\gamma'$. Similarly we can construct corrections for $\beta x\gamma$ a simple perturbation of $\beta\gamma$, $\beta\alpha\gamma$ a simple perturbation of $\beta x\alpha\gamma$ and $\beta x\alpha\gamma$ a simple perturbation of $\beta\alpha\gamma$. Therefore the state machine satisfies N-Forward Correctability. ❑

We will demonstrate how to apply the above unwinding theorem with an example. Figure 7.2 illustrates the system under consideration. The usefulness or the function of this system in unimportant. We will demonstrate that the state machine is Generalized Noninterference secure but is not 0-Forward Correctability secure.

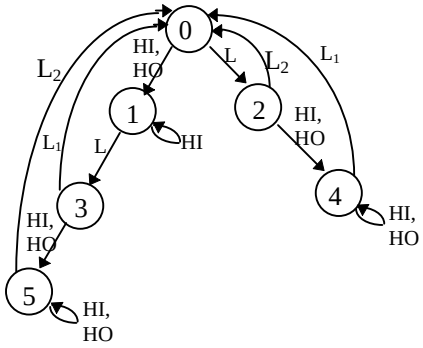


Figure 7.2: A State Machine Used to Demonstrate the Unwinding Theorem.

Since we want to determine if it satisfies both forward correctable and non-forward correctable GNI we must transform the state machine such that the forward correctable version is applicable. Figure 7.3(a) shows the results of replacing the high level output transitions with λ transitions. Figure 7.3(b) shows the transformation to remove the λ transitions and Figure 7.3(c) removes the non-determinism.

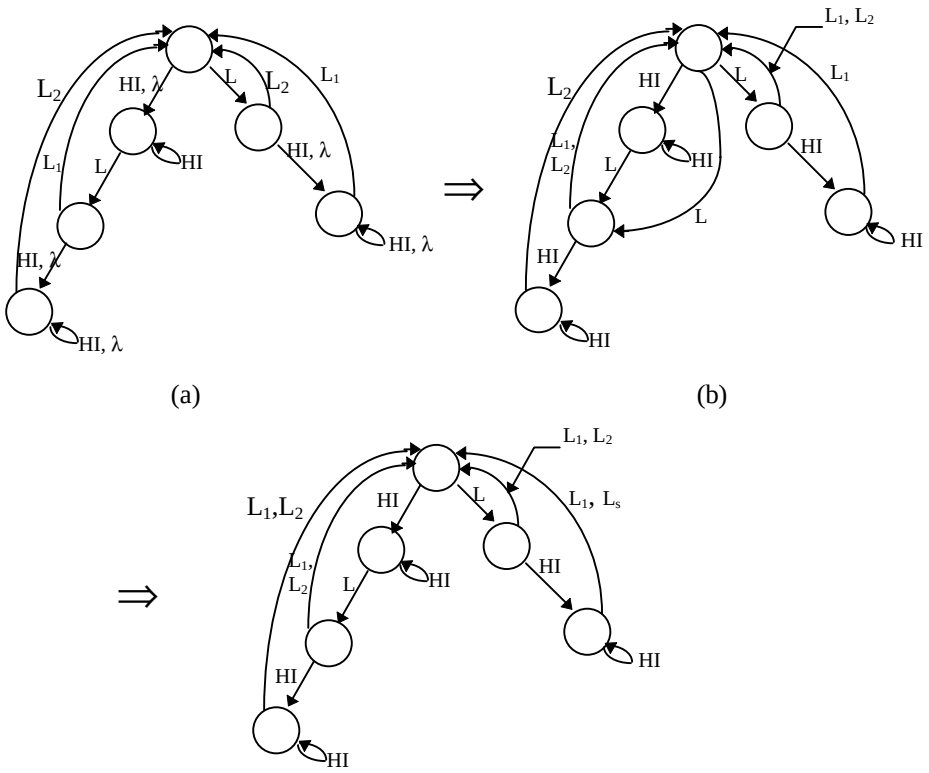


Figure 7.3: Transforming a State Machine.
The state machine is transformed so that forward correctable GNI can be applied to it. Part (a) shows the replacing of high level output transitions with λ transitions. (b) shows the elimination of the λ transitions and (c) shows the elimination of non-determinism.

After the high level output transitions have been replaced with λ transitions the high level input transitions should be removed from the state machine. The state machine at this point will only have states with low level event transitions. Now the projections from each state must be calculated. The projection indicates the possible futures that the low level user in that state. Figure 7.4(a) and Figure 7.4(b) show this transformation given causal and non-causal GNI respectively.

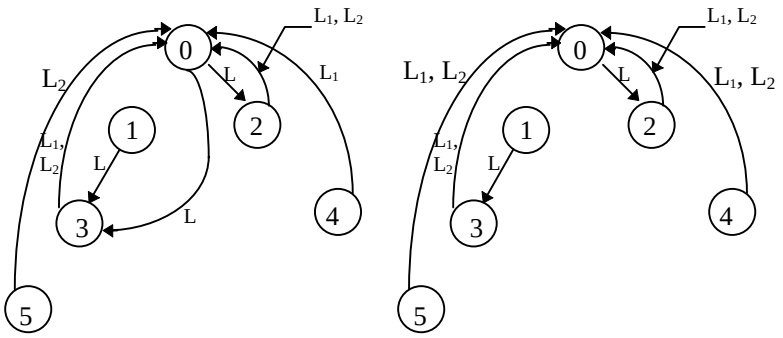


Figure 7.4: State Machines to be used to Calculate Projections.
Figure (a) is the state machine for forward correctable GNI and (b) is the state machine to be used for non-forward correctable GNI.

We will now calculate the projections from each state. For the state machine in part (a) of Figure 7.4 (Note: '+' means OR):

$\pi(q_0) = [L(L_1 + L_2)^*]^*$
$\pi(q_1) = null + L(L_1 + L_2)^*$
$\pi(q_2) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$
$\pi(q_3) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$
$\pi(q_4) = null + L_1[L(L_1 + L_2)^*]^*$
$\pi(q_5) = null + L_2[L(L_1 + L_2)^*]^*$

For the state machine in part (b) of Figure 7.4:

$\pi(q_0) = [L(L_1 + L_2)^*]^*$
$\pi(q_1) = null + L(L_1 + L_2)^*$
$\pi(q_2) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$
$\pi(q_3) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$
$\pi(q_4) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$
$\pi(q_5) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^*$

We are ready to apply the unwinding conditions to see if the state machines satisfy GNI. For the first state machine notice that $\pi(q_3) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^* \neq \pi(q_3/HI) = \pi(q_5) = null + L_2[L(L_1 + L_2)^*]^*$. Therefore this system does not satisfy GNI. Another way of seeing this is to consider the trace $<L,L_2>$ and the perturbation $<L,HI,L_2>$. This perturbation does not have a causal correction. This can be seen by examining state 5 of Figure 7.2. There is no L_2 transition from this state and all high level events do not change the state of the system.

Now we will examine the state machine in part (b) of Figure 7.4:

$\pi(q_0) = [L(L_1 + L_2)^*]^* = \pi(q_0/HI) = \pi(q_1)$
$\pi(q_1) = \pi(q_1/HI) = \pi(q_1)$
$\pi(q_2) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^* = \pi(q_2/HI) = \pi(q_4)$
$\pi(q_3) = null + (L_1 + L_2)\,[L(L_1 + L_2)^*]^* = \pi(q_3/HI) = \pi(q_5)$
$\pi(q_4) = \pi(q_4/HI) = \pi(q_4)$
$\pi(q_5) = \pi(q_5/HI) = \pi(q_5)$

Therefore the state machine in part (b) of Figure 7.4 satisfies Forward Correctable Generalized Noninterference. But, we have transformed the state machine and hence the original state machine satisfies Generalized Noninterference. The trace the gives Forward Correctable Generalized Noninterference problems, namely $<L,L_2>$, and the perturbation $<L,HI,L_2>$ has a correction $<HO,L,HI,L_2>$.

7.6. Unwinding Theorem for PSP

The unwinding theorem for PSP is similar to the one for N-Forward Correctability. The difference being that in PSP high level outputs cannot be freely inserted to ensure a correction exists. Therefore the unwinding theorem for PSP is

Theorem 7.4: If *M* is the event system acceptor for the event system *S*, *S* satisfies PSP if and only if $\forall x{:}H \cdot \forall q{:}Q \cdot \pi'(q) = \pi'(q/x)$

Proof:

This follows immediately from the definition of PSP. PSP is defined by saying that a possible high level event can be inserted or not at any point in the trace and the low level events must stay the same. This is exactly what is happening here. In any state of the system, performing a possible high level event must not change the low level users view. ❑

The unwinding theorem for PSP is simpler than that for the other class of security properties. This should not be a surprise since it is also more restrictive then the other class of properties. The unwinding condition is simple enough that an automated tool can easily take a description of the state machine under consideration and quickly determine if it satisfies PSP.

7.7. Unwinding Theorem for Generalized Noninterference.

The properties considered above ensure that any high level that occurs will not affect the low level users view of the system. Generalized Noninterference requires that for any trace the trace without any high level input events be a trace of the system. The unwinding theorem for Generalized Noninterference is similar to the one for N-Forward Correctability. The difference can be illustrated as follows: If a low level user is in a state of the state machine he sees a projection π_1. If no high level input events occur then all traces in π_1 are possible. Consider a high level input occurring and the system moving to a new state with projection π_2. The property indicates that removing this event must not decrease the possible futures, therefore, $\pi_2 \subseteq \pi_1$. If this is not true then a possible trace after the occurrence of the high level event is not possible if it does not occur. The projections do not have to be equal (i.e. $\pi_2 = \pi_1$) since the property merely requires the removing of high level inputs to result in valid traces. Therefore, the unwinding theorem for properties like Generalized Noninference is:

$$\forall x{:}HI \cdot \forall q{:}Q \cdot \pi(q) \subseteq \pi(q/x)$$

The proof that this is the unwinding theorem for Generalized Noninference is identical to the proof above except the proof of $\pi(q/x) \subseteq \pi(q)$ must be removed.

As a special case of the unwinding theorem for Generalized Noninference we can give an unwinding theorem for Noninference. The difference between the unwinding theorem for Noninference and Generalized Noninference is that high level outputs can not be freely inserted to ensure a trace exists. Therefore we must use the second version of the projection operator. The unwinding theorem becomes: $\forall x{:}H \cdot \forall q{:}Q \cdot \pi'(q) \subseteq \pi'(q/x)$

7.8. Conclusions

In this chapter we have presented a technique to construct an unwinding theorem for a class of security properties. This result and the composition theorems give the system designer all the tools to construct secure systems.

8. Summary and Conclusions

Nothing is worth doing unless the consequences may be serious
George Bernard Shaw *(1865-1950)*
Anglo-Irish Playwright, critic.

8.1. Summary

In this work we have presented and examined a framework for expressing and analyzing security properties. The use of this framework will allow the system designer to reason about security properties both abstractly and in the design of systems. Furthermore, it provides a means to ensure that the system begin designed enforces the desired security property.

We began by examining how low level users can infer possibilistic information from high level activity. We then defined a property that has no possibilistic information flows and is the weakest such property. Examining this property led us to the definition of a security property. A security property enforces the existence of certain high level traces for every possible low level observation.

We then examined the composition of security properties. We began our examination of secure composition with cascade composition. We demonstrate how to determine the effects of interconnecting two components with known properties. We next turned to determining under what conditions a property may emerge under composition. We demonstrated that if a property satisfies a stability requirement then it may only emerge in a very specific fashion. Finally, we turned to composition in the presence of feedback. Our investigation of feedback began by considering what structures of the system caused feedback to fail. We discovered that it was when the system graph had a two cycle. We then presented necessary and sufficient conditions for the feedback composition to succeed.

That last thing we presented was an unwinding theorem. The unwinding theorem can be used to determine if a component expressed as a non-deterministic finite automata satisfies a property.

8.2. Conclusions

The ability to construct large complex systems from smaller independently designed and verified components is a requisite in building affordable secure systems. In this work we have presented the foundations required to build such systems. Even though our results have been applied directly to the design and analysis of secure computer systems we conjecture that what we call a property can be broadly extended beyond security to many other system features. Several such areas are fault tolerance, availability and data integrity. It may be possible in the future to incorporate some of these ideas into the software engineering area.

8.3. Future Work

The previous sections have summarized the results of this thesis. However, as in every research effort, the results have indicated a number of areas where further work could lead to other important conclusions. Some of these areas include:

- Developing a software-base tool that will take a state based definition of a system and a property and determine if the system satisfies the property. Such a tool could also be used to construct systems with a known property from individual components.
- Work must be done on developing a secure refinement theorem. System designers must know at each step of the design process if the system satisfies the desired secure property. Waiting until the design is done to verify the system might result in a large redesign effort if the desired security property is not satisfied.
- It would be interesting to examine what other types of properties can be handled by our approach. In this work a security property was defined as a predicate over a bunch of traces that look the same to the low level user. A fault tolerant property might be defined as a predicate over a bunch of traces with the same external visible behaviour.

Appendix A - Proof of Stability for Various Security Properties

In this Appendix we will demonstrate that the stability requirement is satisfied by most of the security properties presented in the literature. Recall the definition of a stable property:

Definition 5.3: Stable Property.

A property *P* will be called *stable* if and only if for all systems *S*, $\forall\alpha$:power_set(E)·P(S)$\Rightarrow$P(S\α).

Separability

In a Separability secure system all interleavings of high level traces and low level traces are present. Removing an event, be it high level or low level, might reduce the number of traces but all interleavings are still possible.

PSP, GNI and N-Forward Correctability

All of these properties are very similar and hence a similar argument demonstrates that they are stable properties. Consider a system, *S*, that satisfies one of the above properties. Removing a low level event from *S* will result in a system that still satisfies the property since all perturbations of low level traces still have a correction but there are fewer low level traces to consider. Removing a high level input event is not a problem because this can be viewed as a perturbation to the traces *S* . We will now examine removing a high level output event from *S*. We will argue that this results in a system that still satisfies the property. Consider a trace τ of *S* such that a perturbation σ requires some particular high level output event to have a correction τ'. The trace τ' with the high level outputs removed is a trace of the new system and is a correction to the σ perturbation. Therefore all of the properties are stable.

Noninference and Generalized Noninference

Removing low level events and high level input events from a system that satisfies either of these properties will result in a system that still satisfies the property. Also removing high level outputs for a Noninference secure system is acceptable since the

property requires the trace with no high level events to be trace of the original system. By the same argument as that given for the PSP, GNI and N-Forward Correctability class of property Generalized Noninference is a stable property.

Appendix B - Proof that ≡ is an equivalence relation

In this appendix we prove that the following expression is an equivalence relation on the set of traces, *T*, of an event system:

$$s \in T,\ t \in T \quad s \equiv t \text{ iff } \forall r:E^* \cdot s^\wedge r \in \tau \Leftrightarrow t^\wedge r \in \tau \qquad (1)$$

Reflexive

Let *s* be an element of *T*. We must show $s \equiv s$.

Assume $\neg s \equiv s$

$= \neg \forall r:E^* \cdot s^\wedge r \in T \Leftrightarrow s^\wedge \in T$

$= \exists r:E^* \cdot \neg (s^\wedge r \in T \Leftrightarrow s^\wedge \in T)$

$= \bot$

Symmetric

Let $s \in T$ and $t \in T$. We must show $s \equiv t \Rightarrow t \equiv s$

$s \equiv t$

$= \forall r:E^* \cdot s^\wedge r \in T \Leftrightarrow t^\wedge \in T$

$= \forall r:E^* \cdot t^\wedge r \in T \Leftrightarrow s^\wedge \in T$

$= t \equiv s$

Transitive

Let $s,t,u \in T$ we must show $s \equiv t \wedge t \equiv u \Rightarrow s \equiv u$

$s \equiv t \wedge t \equiv u$

$= \forall r:E^* \cdot s^\wedge r \in T \Leftrightarrow t^\wedge \in T \wedge \forall r:E^* \cdot t^\wedge r \in T \Leftrightarrow u^\wedge \in T$

$= \forall r:E^* \cdot s^\wedge r \in T \Leftrightarrow t^\wedge \in T \wedge t^\wedge r \in T \Leftrightarrow u^\wedge \in T$

$= \forall r:E^* \cdot s^\wedge r \in T \Leftrightarrow u^\wedge \in T$

$= s \equiv u$

Therefore, the relation defined in (1) is an equivalence relation and partitions the set of traces of an event system into equivalence classes.

List of References

[Abadi & Lamport90] Martin Abadi and Leslie Lamport. "Composing Specifications," *Technical Report 66*, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1990.

[Allen91] P.G. Allen. "A Comparison of Non-Interference and Non-Deducibility using CSP," *Proceedings of the Computer Security Foundations Workshop IV*, pages 43-54. IEEE Press. June 1991.

[Alpern & Schneider85] Bowen Alpern and Fred Schneider. "Defining Liveness," *Informatin Processing Letters, 21(4),* pages 181-185. October 1985.

[Bell & LaPadula75] D. Elliott Bell and Leonard J. LaPadula. "Secure Computer Systems: Mathematical Foundations," *Technical Report TR-2547*, MITRE Corporation, Bedford, MA, 1975]

[Bevier & Young94] William R. Bevier and William D. Young. "A State-Based Approach to Noninterference," *Proceedings of the Computer Security Foundations Workshop VII*, pages 11-21. IEEE Computer Society, June 1994.

[Brzozowski64] J. A. Brzozowski. "Derivatives of Regular Expressions," *Journal of the ACM*, Vol. 11, No. 4, pages 481-494. October 1964.

[Cover91] Thomas M. Cover and Joy A. Thomas. "Elements of Information Theory." John Wiely & Sons, Inc. Toronto, 1991.

[CTCPEC90] Canadian System Security Centre. "The Canadian Trusted Computer Product Evaluation Criteria Version 2.0," Ottawa, December 1990.

[CTCPEC92] Canadian System Security Centre. "The Canadian Trusted Computer Product Evaluation Criteria Version 3.0e," Ottawa, April 1992.

[Cuppens & Cuppens94] N. Boulahia-Cuppens and F. Cuppens. "Asynchronous composition and required security conditions," *Proceedings of the 1994 IEEE Computer Security Symposium on Research and Privacy*, pages 68-78. IEEE Press. 1994.

[Denning76] Dorothy E. Denning. "A Lattice Model of Secure Information Flow," *Communications of the ACM*, pages 236-242. May 1976, Volume 19, Number 5.

[Foley87] Simon N. Foley. "A Universal Theory of Information Flow," *Proceding of the 1987 IEEE Symposium on Research in Security and Privacy*, pages 116-121. IEEE Press. 1994.

[Garcia89] Alberto Leon-Garcia. "Probability and Random Processes for Electrical Engineering." Addison-Wesley Publishing Company, New York, 1989.

[Ginzburg68] A. Ginzburg. "Algebraic Theory of Automata," Academic Press, 1968.

[Goguen & Meseguer82] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models," *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, pages 11-20. IEEE Press. April 1982.

[Goguen & Meseguer84] Joseph A. Goguen and José Meseguer. "Unwinding and Inference Control," *Proceedings of the Symposium on Security and Privacy*, pages 75-86. IEEE Computer Society, May 1984.

[Gray90] James W. Gray III. "Probabilistic Interference," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. IEEE Press. May 1990.

[Gray92] James W. Gray III. "Mathematical Foundations for Information Flow Security," *The Journal of Computer Science*, Volume 1, Number 3,4. 1992.

[Guttman & Nadel88] J. D. Guttman and M. E. Nadal. "What Needs Securing?", *Proceedings of the Computer Security Foundations Workshop*, IEEE Computer Society, June 1988, pages 34-57. June 12-15 1988.

[Hemenway & Gambel91] Judith Hemenway and Dan Gambel. "Issues in the Specification of Composite Trustworthy Systems," Grumman Data Systems.

[Hinton96] Heather M. Hinton. "Properties and Meta-Properties of Secure Composable Systems," Ph.D. dissertation, University of Toronto, Toronto, Ontario. 1996.

[Hoare85] C. A. R. Hoare. "Communicating Sequential Process." London: Prentice-Hall International, UK, LTD., 1985.

[Jacob88] J. Jacob. "Security Specifications," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pages 14-23. IEEE Press. May 1988.

[Johnson & Thayer88] Dale M. Johnson and F. Javier Thayer. "Security and the Composition of Machines," *Proceedings of the Security Foundations Workshop*, Franconia, NH, pages 72-89. June 1988.

[Lee et al. 92] E.S. Lee, P.I.P. Boulton, B.W. Thomson, and R.E. Soper. "Composable Trusted Systems," *Technical Report 272*, Computer Systems Research Institute, University of Toronto, Toronto, Ontario. May 1992.

[Marcus88] Leo Marcus and Timothy Redmond. "A model-theoretic approach to specifying, verifying, and hooking up security policies," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pages 127-138. IEEE Press. May 1988.

[Mazurkiewicz] Antoni Mazurkiewicz. "Trace Theory," *Lecture Notes in Computer Science*, Vol. 255, pages 279-324. Springer-Verlag.

[McCullough87] Daryl McCullough. "Specifications for Multi-Level Security and a Hook-Up Property," *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*. IEEE Press, May 1987.

[McCullough88] Daryl McCullough. "Noninterference and the Composability of Security Properties," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pages 177-186. IEEE Press, May 1988.

[McCullough90] Daryl McCullough. "A Hookup Theorem for Multilevel Security," *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, pages 563-568. IEEE Press, June 1990.

[McLean87] John McLean. "Reasoning about Security Models," *Proceedings of the 1987 Symposium on Research in Security and Privacy*, pages 123-131. IEEE Press. April 1987.

[McLean88] John McLean. "The Algebra of Security," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*. IEEE Press. 1988.

[McLean90] John McLean. "Security Models and Information Flow," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 177-186. IEEE Press. May 1990.

[McLean92a] John McLean. "Proving Noninterference and Functional Correctness Using Traces," *Journal of Computer Security*, Vol. 1, No. 1, pages 37-58. IOS Press 1992.

[McLean92b] John McLean. "Models of Confidentiality: Past, Present, and Future," *Proceedings of the Computer Security Foundations Workshop VI*. IEEE Press. June 1993.

[McLean94] John McLean. "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions," *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, pages 79-93. IEEE Press. May 1994.

[Millen90] Jonathan Millen. "Hookup Security for Synchronous Machines," *Proceedings of the Computer Security Foundations Workshop III*. IEEE Press. June 1990.

[Millen94] Jonathan K. Millen. "Unwinding Forward Correctability," *Proceedings of the Computer Security Foundations Workshop VII*, pages 2-10. IEEE Computer Society, June 1994.

[Milner80] R. Milner. "A Calculus of Communicating Systems," *Lecture Notes in Computer Science 92*, Springer Verlag, 1980.

[Moore90] Andrew P. Moore. "The Specification and Verified Decomposition of System Requirements Using CSP," *IEEE Transactions on Software Engineering*, pages 932-948. September 1990, Volume 16, Number 9.

[Moore90] Andrew P. Moore. "The Specification and Verified Decomposition of System Requirements Using CSP," *IEEE Transactions on Software Engineering, Volume 16, Number 9*. September 1990.

[Murata89] Tadao Murata. "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, pages 541-580. April 1989, Volume 77, Number 4.

[Nestor93] John Paul Nestor and E. Stewart Lee. "The Composition of Property-Preserving Event Systems," *Technical Report CSRI-290*, Computer Systems Research Institute, University of Toronto, Toronto, Ontario. November 1993.

[O'Halloran90] Colin O'Halloran. "A Calculus of Information Flow," *Proceedings of the European Symposium on Research in Computer Security*. Toulouse, France. 1990.

[Peterson81] J. L. Peterson. "Petri net theory and the modeling of systems," Prentice-Hall, 1981.

[Rushby91] John Rushby. "Composing Trustworthy Systems. A Position Paper," SRI International. 1991.

[Sutherland86] David Sutherland. "A Model of Information," *Proceedings of the 9th National Computer Security Conference*, pages 175-183. 1986.

[TCSEC85] National Computer Security Center. "Department of Defense Trusted Computer Security Evaluation Criteria," DOD 5200.28-STD, December 1985.

[Thomson88] B. Thomson, E.S. Lee, P.I.P. Boulton, M. Stumm and D.M. Lewis. "A trusted Network Architecture." Technical Report CSRI-228, Computer Systems Research Institute, University of Toronto, Toronto, Ontario. October 1988.

[Varadharajan90] Vijay Varadharajan. "Petri Net Based Modeling of Information Flow Security Requirements," *Proceedings of the Computer Security Foundations Workshop III*, pages 51-60. IEEE Press. June 1990.

[Winskel] Glynn Winskel. "Event Structures,: *Lecture Notes in Computer Science 255*.

[Wittbold & Johnson90] J. Todd Wittbold and Dale Johnson. "Information Flow in Nondeterministic Systems," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. IEEE Press. May 1990.

[Wood87] Derick Wood. "Theory of Computation," New York: Harper & Row, Publishers, Inc., 1987.

[Zakinthinos & Lee] A. Zakinthinos and E. S. Lee. "The Composability of Non-Interference," To Appear in the *Journal of Computer Security*. IOS Press.

[Zakinthinos & Lee95] A. Zakinthinos and E. S. Lee. "The Composability of Non-Interference," *Proceedings of the Computer Security Foundations Workshop VIII*. IEEE Press. June 1995.

[Zakinthinos & Lee96] A. Zakinthinos and E. S. Lee. "How and Why Feedback Composition Fails," *Proceedings of the Computer Security Foundations Workshop IX*. IEEE Press. June 1996.

[Zwiers & Roever89] Job Zwiers and Willem-P. de Roever. "Compositionality and Modularity in Process Specification and Design: A Trace-State Based Approach," *Lecture Notes in Computer Science*, Vol. 398, Temporal Logic in Specification, pages 351-374. Springer-Verlag, 1989.