Properties and Meta-Properties of Secure Composable Systems

Heather Maria Hinton

Technical Report CSRI-343
March 1996

Computer Systems Research Institute
University of Toronto
Toronto, Canada
M5S 3H5

The Computer System Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an institute withi the Faculty of Applied Science and Engineering and the Faculty of Arts and Science at the University of Toronto. CSRI is supported in part by the Natural Sciences and Engineering Research Council of Canada.

PROPERTIES AND META-PROPERTIES OF SECURE COMPOSABLE SYSTEMS

by

Heather Maria Hinton, P.Eng.

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

Properties and Meta-Properties of Secure Composable Systems

Doctoral Dissertation, June 1996

Heather Maria Hinton, P.Eng.
Department of Electrical and Computer Engineering,
University of Toronto

The behaviour of a system is defined as secure based on the desired security properties of the system. The assessment of computer security includes the verification that the behaviour of a system implies these desired security properties. This verification should allow us to create hierarchically complex, secure systems from the interconnection of simpler, secure components without the need to re-verify the composite system. We know that many secure behaviours become insecure on composition yet we cannot always predict why or when this will be the case.

Security must be based on a set of goals, clearly defining what is required of any secure system. We define a set of safety and progress properties that are necessary for the secure behaviour of a system and are common to all systems.

Knowledge of the properties of these properties (their meta-properties) is required to ensure that the overall system will behave as expected. The relations of reflexivity, transitivity, and symmetry of properties and the closure of properties on composition describe the meta-properties of safety and progress.

We define and examine the compatibility of properties. Compatibility ensures that different properties do not have goals that are mutually exclusive.

We introduce a systematic means to define and assess emergent properties, properties that are not applicable to the behaviour of individual components but are relevant to the behaviour of interconnections of these components. It is shown that many desired security properties will act as emergent properties.

These safety and progress properties, together with the knowledge of their meta-properties, compatibility, and emergence, are used to assess and to predict the composability of security characteristics of a system. We incorporate these characteristics and criteria in a Composition Principle, providing guidelines for the assessment of properties of composite systems. The Composition Principle provides a more comprehensive approach to composite system security than was previously possible. This in turn will allow us to compose secure systems in a manner consistent with the goals of the secure behaviour of these systems.

ACKNOWLEDGEMENTS

During the last four years my work has benefited greatly from discussions with many different people. In particular, the following people provided especially useful comments and insights: Peter Boulton at University of Toronto, Mark Lomas at Cambridge University, John Rushby at SRI International, David Bonyon, Sandy Sandberg-Maitland of CGI and Milan Kuchta at Canadian Security Establishment

I would like to especially thank my supervisor, E. Stewart Lee, for his support and advice.

I am also grateful to my committee for their helpful comments and insights, especially Professors Boulton, Kschichang, Zaky, Andrews and my external examiner, Sandy Sandberg-Maitland.

I am grateful to the Natural Science and Engineering Research Council for the financial support of a PGS B scholarship, the University of Toronto for the financial support of a University of Toronto Open Fellowship, and the Information Technology Research Centre for the equipment used.

I would also like to thank my friends in the Computer and Electronics Groups, for support in the good times and the bad. In alphabetical order they are: Vaughn (Betz), Naveen (Bewtra), John (Chappel), Kevin (Chung), Bob (Francis), Qiang, Yaska (Sankar), Steve (Wilton), and Aris (Zakinthinos).

David H. gets a big thank-you for helping me to (boldly) un-split infinitives, where none had been split before.

There is no way to thank my husband Philip for all of his support and encouragement, or my family, for their love, understanding, and patience. Thank you very much.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICIES

LIST OF APPENDICIES

LIST OF APPENDICIES

1. Introduction

...He thought he saw an Argument
That proved he was the Pope:
He looked again, and found it was
A Bar of Mottled Soap.
'A fact so dread,' he faintly said,
'Extinguishes all hope!'
LEWIS CARROLL: Mad Gardener's Song

Superstition breeds bad luck.
RAYMOND SMULLYAN: 5000 B.C.

In many respects, computer security is a very mature area of research. For example, encryption is used to send data across insecure or hostile networks, and firewalls are used to protect systems from attack by hackers across the Internet. Secure systems are required to protect the sensitive information stored on computers from banking records, to medical information, to military data. Secure composite systems are required so that this information can be used by people across a diverse organisation, such as a bank or a hospital, without risk to the information or the system itself.

There remain several fundamental, unsolved problems within computer security: the requirements for security have not been comprehensively defined, and we are unable to predict when security will be maintained by composite systems. This is limiting our ability to create complex systems that are provably, or even demonstrably, secure.

The behaviour of a component or system is defined as secure (or insecure) based on the desired security properties of the system. The assessment of computer security includes the verification that the behaviour of a system implies the desired security properties. This may take the form of an informal argument or demonstration, or a formal, rigorous proof, or a combination of the two. Experience shows that informal arguments are often flawed or too general to include the specific, relevant (insecure) details of a system.

The verification of security for components and systems should allow us hierarchically to create complex secure systems from the interconnection of simpler, secure components without the need to re-verify the composite system. The secure behaviour of the simpler components must be retained by the component within the composite system. We know that many secure behaviours become insecure on composition [McC87, McC88], yet we cannot always predict why or when this will be the case.

It is the goal of this work to examine the secure (and insecure) behaviour of components on composition. We examine properties defining the desired security characteristics and the properties of these properties (meta-properties). These meta-properties are used to help predict the properties of a complex, composite system, given the structure of the composite system and the properties of the system's components. This in turn will allow us to compose secure systems in a manner consistent with the goals of the secure behaviour of these systems.

This work introduces a set of safety and progress properties of system behaviour necessary for security. These properties are common to all secure systems, regardless of the overall goal of the system. We formally specify these properties and investigate their characteristics. In particular, we examine when (if) they are preserved when systems exhibiting these properties are composed. We are concerned primarily with physical computer systems. We believe, however, that the results of this work can be equally applied to a study of software systems.

1.1 Introduction to Systems Security

Previous approaches to systems and information security have been tailored to the system and information under consideration. Security is defined by the goals of behaviour that a system must exhibit, where the exact nature of security is dependant upon the system in question. Military-type approaches are concerned with multi-level security, where users with security-level clearances may or may not access information based on the security-level clearance of the information. Commercial approaches are concerned with unauthorised modification of information. This approach has the added complication that users may have different roles as part of their job description; the information that may be accessed will often depend on the current role of the user. Safety-critical systems, such as control systems for nuclear power plants, are concerned with the correctness of these control systems and their ability to withstand penetration by external (malicious) users. Dependable systems, such as fly-by-wire controls on aircraft, must be shown to be fault-tolerant, that is, to continue to operate correctly in the presence of unwanted stimuli.

All of these different systems have the same basic goal: to ensure that the system, and the information it contains, will behave in a manner that is consistent with a set of predefined goals. Even though many of the goals of these diverse systems are common, each has been developed in isolation.

In this work, we do not consider any one type or approach to security. The basic security properties introduced in this work apply to all types of security. They represent a set of common security goals that must be defined and understood before we can go on to examine the complex behaviours of interconnected systems.

1.2 Security Properties

Properties define the desired behaviour of a system. Security properties have the goal of specifying the types of acceptable behaviours of a system. Security properties have been represented as possibilistic properties (where it is possible that something may happen) and probabilistic properties (where the probability of different events is explicitly considered). Represented formally, security can be represented as a safety property or a progress property. Security therefore encompasses both the safety and progress aspects of system behaviour. Safe behaviour is always acceptable (never unacceptable) to the goals of the system. Progress requires that actions will eventually occur and that the goals of system behaviour are eventually fulfilled. We believe that security depends on safety and progress and cannot be partitioned into a consideration of one without the other.

It is our goal to be able to define theoretically and to quantify those composable safety and progress properties that can be included in formal specification requirements.

1.3 Composition of Systems and Properties

The composition of systems is the simple process of interconnecting individual systems, or components. Unfortunately, properties of individual systems are not always retained as properties of composite systems. The problem statement is clearly articulated in the following quotation from John McLean:

> 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 preserved by certain forms of composition. 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?" [McL94]

Where McLean refers to the composition of systems, we refer to the composition of properties. We contend that systems can always be composed; it is the identification of the properties that continue to be satisfied that is of interest. In this work we examine security properties on composition and propose a solution to the problem as stated by McLean.

1.4 Proposition

Security must be based on a set of goals, clearly defining what is required of any secure system. These goals are formally represented as properties of the behaviour of the system. Knowledge of the properties of these properties (the meta-properties) is required to ensure that the overall system will behave as expected, both in isolation and when connected with other components.

We define a set of safety and progress properties that are necessary for the secure behaviour of a system. These properties must be considered in conjunction with one another to ensure that we have comprehensively defined the security requirements of a system. The relations of reflexivity, transitivity, and symmetry of properties and the closure of properties on composition describe the meta-properties of the safety and progress properties. These meta-properties indicate the characteristics of the properties for different structures of composite systems.

We define and examine the compatibility of properties to ensure that different properties do not have goals that are mutually exclusive when considered with respect to the behaviour of a system. It will be shown that properties may be incompatible in terms of a set of goals. Such incompatibility may be conditional on another property.

We introduce emergent properties, properties that are not applicable to the behaviour of individual components but are relevant to the behaviour of interconnections of these components. It will be shown that many desired security properties will act as emergent properties and that existing properties must not act as de-emergent properties.

These necessary safety and progress properties, together with the knowledge of their meta-properties, compatibility, and emergence, are used to assess and to predict the composability of security characteristics of a system. We incorporate these characteristics and criteria in a composition principle used to prove properties of composite systems.

It is the proposition of this work that a complete definition of security requirements requires the consideration of both safety and progress aspects of security, together with knowledge of the meta-properties, compatibility and emergent characteristics of properties.

1.5 Contributions of This Work

This dissertation presents an integrated approach to the composite of secure systems. This is a necessary step towards designing and building large, integrated, secure systems.

The basic safety and progress properties form the basis of a secure system, from which we define security properties that are relevant to that particular system. These basic properties clarify the minimum goals required for all systems, regardless of the level or type of security required for the system.

We show that the consideration of the meta-properties offers the possibility of a complete definition and modelling of the security requirements of a system. The detailed examination of the meta-properties of properties simplifies the evaluation of the satisfiability of properties of composite systems. The knowledge of how and when properties are composable, given the composite structure in question, allows us to define systems and properties that are consistent with the required system goals. This cannot be achieved by the independent examination of the individual aspects of security. The organised treatment of the meta-properties is novel and new.

It is necessary to consider the compatibility of properties. This provides us with insight into when and why properties are not satisfied. This is a critical aspect of security that is required so that we do not inadvertently define a secure system with incompatible properties, resulting in an insecure system.

It is necessary to consider the emergence of properties when examining the security of a composite system. Emergent properties must be examined to ensure that these new behaviours do not violate the overall goals of the system's behaviour. This is critical to the accurate assessment of the composite system's security. The systematic approach to the evaluation of emergent properties introduced in this work allows us to accomplish this goal.

We introduce a Composition Principle that encompasses these aspects of security. Using this principle, we are able to assess the security of a composite system, given the security of the system components. One of the major differences between this principle and previous approaches is the comprehensive consideration of the different aspects of security. This approach is more general than previous approaches and can be applied to the different types of security requirements and properties.

1.6 Structure of this Document

Chapter 2 provides a brief review of the development of the composition problem for systems and security properties. We also introduce some of the different approaches to safety and progress. This is a very brief introduction to some of the many issues of computer security.

Chapter 3 is a brief introduction and summary of the language and temporal logic used as the basis of the model of the system and its behaviour.

We introduce in Chapter 4 the notation used for the specification of component systems and composite systems based on the temporal language. In this chapter we also define the composition constructs that we will consider. This notation will be used throughout this dissertation to describe systems and sequences and the composition of systems.

In Chapter 5 we define and specify properties and meta-properties of properties. We introduce a means to determine if a property is satisfied by the behaviour of a component or system. The completeness of the set of properties and their fulfilment of overall security goals is also discussed.

Chapter 6 addresses those properties classified as safety properties. We introduce and define a set of basic safety properties. These properties describe the safety goals common to all secure systems. These properties are shown to be dependent on the environment in which a system exists. This is reflected in their meta-properties, as demonstrated in this chapter. We discuss how to identify additional properties that are relevant to a given system (system-specific properties).

Progress properties are discussed in Chapter 7, where the basic progress properties and their meta-properties are introduced. These properties are common to all secure systems. The process of identifying and defining system-specific progress properties based on the classification of the formal structure of progress properties is introduced.

Chapter 8 examines the compatibility of properties. Two compatible properties will not interfere with each other and therefore can be simultaneously enforced for a system. We define how to assess the compatibility of different properties, including the compatibility of a safety property and a progress property.

Chapter 9 addresses the emergence of properties on the interconnection of systems. The interconnection of components results in a system with behaviours that were not possible for the individual components. We describe a procedure that will help identify different types of emergent properties for a composite system.

This leads to an examination, in Chapter 10, of the composability of properties given the interconnection of systems. This includes a brief discussion of the threats to the basic safety and progress properties of a composite system. Knowledge of composability of properties will allow us to simplify the assessment of the safety and progress properties of a composite system. We also introduce a composition principle to guide us in the assessment of the properties of a composite system.

In Chapter 11 we apply the findings of this work to several different systems with different security requirements. These examples illustrate the value of this work in the consideration of security and the composition of security. We show that this approach illustrates the sources of many of the problems experienced on composition. This approach therefore allows us to define and compose composite secure systems.

Chapter 12 concludes this dissertation. We examine this work in relation to real-world applications of formal models and methods and we suggest further work that it may be useful to undertake.

2. Previous Work

But if I don't get from under
pretty damned soon,
there'll be a disaster in the rear
ARISTOPHANES: The Frogs
(tr. Dudley Fitts)

In this chapter we briefly review and highlight the development and issues of the security-on-composition problem. We motivate this problem by considering the formal criteria for secure system evaluation as given by various government agencies. We review the development of security properties and the types of security that can be represented. The problem of assessing security on composition is discussed for several types of security. We also introduce the notion of progress, an aspect of behaviour that we believe is integral to security. The work of this dissertation is based on these various aspects of security and the need to include them in any assessment of the security of systems and composite systems.

2.1 Formal Criteria

The formal criteria, or national computer security policies, are the standards by which security of a system is established and verified. None of the three formal criteria released by the U.S. government, the European Community or the Canadian government deal with composability. The formal criteria of the United States Government are included in the Rainbow books comprising the Trusted Computer System Evaluation Criteria. The Orange Book defines the *Trusted Computer System Evaluation Criteria* [TCSEC]. While the Orange Book establishes guidelines and measures for the level of security confidence, it does not address the question of composability of systems of known confidence levels nor does it address progress. Every system, regardless of its components, must be evaluated in its entirety to determine its security.

The *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* [TNI] (the Red Book), published in July 1987, does deal with the interconnection of network systems. Appendix A of the Red Book "presents a technical basis for assigning a rating to a (sub)system which is composed of more than one component. The rating assigned indicates a minimum level of security as provided by the rated (sub)system as a whole" [TNI]. It does not deal with observations of McCullough that security can be lost on composition, which would violate the security of the composite system, even if created following the composition rules of the TNI.

The *IT Security Criteria: Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems*, known informally as the ITSEC or the Green Book, was produced in 1989 by the German Information Security Agency [ITSEC]. It is implicit in this report that every system must be evaluated, regardless of the security evaluation of its components.

The *Canadian Trusted Computer Product Evaluation Criteria* [CTCPEC], released in January 1993, does not deal with composability. This is because there is no method yet available for the evaluation.

> 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 retain or improve their ratings without having the *entire* product re-evaluated [CTCPEC].

There is an urgent need for a comprehensive theory of composability of systems that can be used in the evaluation of secure systems.

2.2 General System Theory

Systems theory and composition are well established and long studied problems that apply across all disciplines of science. In his 1968 book, *General System Theory*, Ludwig von Bertalanffy [Bert68] discusses the idea of composition of systems and the interactions of these systems as individual systems and as part of the whole system:

> You will always find that the behavior of an element is different within the system from what it is in isolation. You cannot sum up the behavior of the whole from the isolated parts, and have to take into account the relations between the various subordinated systems and the systems which are super-ordinated to them in order to understand the behavior of the parts. The hierarchical nature is characteristic of many systems and is of fundamental importance. [Bert68]

Thus the approaches to the composition of secure systems may very well also be applied to the composition of different types of systems, with requirements other than security.

2.3 Security

Security properties are formal representations of the security-relevant goals of a system's behaviour. These goals are often specific to the type of system under consideration, such as dependable, fault-tolerant or information security. When defined formally, properties can be classed as safety and progress [MaPn92]. We consider security as represented by these two classes of properties.

Previous analyses of security properties have focused on the so-called "CIA" policies, the safety properties of Confidentiality [BLP75] and Integrity [Biba77, ClWi77] and the progress property of Availability [YuGl88, BaKu91]. These properties are usually addressed in isolation. It has been suggested that these properties are inadequate and oversimplified [Par91] and that these properties cannot be discussed separately [HiLe94].

2.3.1 Lattice Approaches to Security

Bell and LaPadula [BLP75] introduced a model of security policies for military systems, based on the MULTICS system. Security was defined to protect the confidentiality of information. Multi-level security as defined by Bell and LaPadula is formalised by the axioms of *simple security* and the **-property* (the star property).

Simple security states that a process p may not have read access to object x unless the security class of x dominates that of p. The *-property states that a process may not have read access to object x and write access to object y unless the security level of y dominates x.

The lattice model approach to security introduced by Denning [Denn76] is an extension of the Bell and LaPadula model. This approach describes policies and channels of information flow within a system. The lattice (SC, $\leq$) defines a finite set of security classes, SC, and a partially ordered binary relation, $\leq$. If $A \leq B$ then the security class A is lower than or equal to the security class B, or, security class B dominates security class A. Information is therefore permitted to flow between equal classes or from a class to a higher class.

Biba [Biba77] defined a simple integrity policy analogous to Bell and LaPadula's simple property. This property is defined as "A process p may not have write access to object x unless the security class of p dominates x." This policy was intended to represent the modification of information, whereas the BLP approach was concerned with the observation of information.

2.3.2 Safety and Liveness

In the papers "Defining Liveness" and "Recognizing Safety and Liveness" [AlSc85a, AlSc87b], Alpern and Schneider give an automata-theoretic characterisation of safety and liveness. Given this characterisation, they prove that every property can be expressed as a conjunction of safety and liveness. Safety properties are proven using invariance arguments where liveness properties also require well-foundedness arguments.

Liveness was first introduced in 1977 by Amir Pnueli as the idea of eventuality [Pnu77]. At the same time Lamport introduced the classification of safety and liveness properties, where liveness stipulates that a "good thing" happens during execution [Lam77]. Alpern and Schneider formalised liveness by defining a partial execution α to be live if and only if there is a sequence β that extends α such that $\alpha\beta \models P$; the extended sequence $\alpha\beta$ entails P for some liveness property P [AlSc85a].

2.3.3 Possibilistic Security Properties

Possibilistic security is based on the idea that if information of a given security level interferes with information of a different security level, it should be attributed to more than one possible cause. In the terminology of non-interference, "what one group of users does using a certain ability has no effect on what some other group of users sees" [GoMe82].

This has been applied to multi-level security, especially within the context of military-type applications. The properties of generalised non-interference [McC87], restrictiveness [McC88], non-inference [OHal90], non-deducibility on strategies [WiJo90], and forward-correctability [JoTh88] are possibilistic properties.

2.3.4 Additional Security Properties

Parker has introduced and defined a set of security properties aimed at strengthening security by containing and minimising the weak points of security [Par91]. These properties are defined as confidentiality, authenticity, integrity, utility, and availability.

> Although defining security policies for one attribute to be preserved may be useful for theoretical purposes, in practice system security must be seamless in all aspects because adversaries are likely to switch from one goal to another. [Par91]

Parker also highlights the need to consider all of these properties for a system, not just individual properties that appear to be the "most" applicable to a given system.

2.4 Composability

2.4.1 Hook-up Security

In his 1987 paper, "Specifications for Multi-Level Security and a Hook-Up Property", Darryl McCullough [McC87] made the somewhat startling observation that two systems, defined to be secure, were not necessarily secure when composed. McCullough introduced the idea of *hook-up security* as a solution to this problem. Hook-up security is defined by the following closure property:

> Hook-up secure systems are deducibility-secure, and, when hooked up with other hook-up secure systems, the result is a hook-up secure system.

The idea of deducibility is simply that a low-level user, who can view all low-level inputs and outputs, can not deduce anything about any high-level inputs. This uses a trace-based representation of a system. Deducibility can not handle non-deterministic systems[1].

McCullough states that while deducibility is necessary for hook-up security it is not sufficient. He therefore turns next to non-interference [GoMe82, GoMe84]. Non-interference requires that no high-level inputs can effect low-level input or output events. Unfortunately, the hook-up of non-interference secure systems is not always a non-interference secure system.

This definition of non-interference is modified by McCullough to give "Generalised Non-Interference" (GNI). This modified definition is used in the following theorem:

> If a system is hook-up secure, then it is deducibility secure, and it has the [generalised] non-interference property. If systems A and B are both hook-up secure, then any composite system formed by identifying outputs of A with inputs of B with the same security level, and vice-versa, is hook-up secure. [McC88]

Generalised non-interference is able to handle non-determinism but is not composable with feedback.

2.4.2 Composing Specifications

In the 1990 technical report "Composing Specifications," Martin Abadi and Leslie Lamport defined a modular specification method for composition. Abadi and Lamport state that "the fundamental problem of composing specifications is to prove that a composite system satisfies its specification if all its components satisfy their specifications" [AbLa90].

An interesting result of this work is the notion that an environment assumption is a safety property whereas properties "guaranteed by the system and its components need not be safety properties: they can include liveness properties" [AbLa90].

This approach requires that we attempt to control the system's behaviour and then compose systems. The environment is external to the system and can not be controlled. Thus an environment specification is an *assumption* about its behaviour. The behaviour of a component is represented by an infinite sequence of states and "agents" that cause the changes of states. Abadi and Lamport view a specification as a set of behaviours, where the most general specification is a program. The Composition Principle put forward in this work applies to specifications of the form $E \Rightarrow M$, where E is a safety property and M is a machine property. This form asserts that a system will guarantee a property M as long as the environment satisfies the safety property E.

Given a composite system Π composed of components Π_i, where the composite system satisfies the environment properties E and the machine properties M, and the components satisfy

---

1 Formal language theory uses mathematical structures called nondeterministic automata. In contrast, researchers in formal security models have tended to use the term "nondeterministic" to refer to machines whose operation contains come elements of randomness. "Probabilistic" would be a better name. [LBTS92]

the environment properties E_i and the machine properties M_i, the Composition Principle is given as:

1. Π guarantees M if each component Π_i guarantees M_i
2. E_i of each Π_i is satisfied if the environment of Π satisfies E and every Π_j satisfies M_j (where $\Pi_i \neq \Pi_j$)
3. Every Π_i guarantees M_i under E_i

Using proof rules, Abadi and Lamport prove that components that behave correctly do so only when their environments also behave correctly. When two components are connected, one component and its environment become part of the other component's environment (and vice versa). Thus compositions of components that behave correctly must behave correctly with respect to the other components and their environments.

2.4.3 Composition of Possibilistic Properties

In "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions", John McLean [McL94] examined the composition of possibilistic properties represented as trace sets. Possibilistic properties do not fall within the Alpern and Schneider categorisation of properties and therefore cannot be handled by the Abadi-Lamport Composition Principle. This is because these properties are properties of *sets of trace sets* (trace sets) and not *trace sets* (individual traces). That is, we cannot determine if a given trace obeys a possibilistic property because we must consider that trace in the context of other possible traces. Refinement, a key step used in applying the Composition Principle, does not preserve possibilistic properties.

McLean defines a Selective Interleaving Function, used to define traces that obey different possibilistic properties. This function is defined such that given two input traces, *t1* and *t2*, it will produce a third trace, $f(t1,t2) \rightarrow t$, such that *t* is a desired possibilistic property. Different interleaving functions can be used to generate traces satisfying different possibilistic properties.

2.4.4 An Environment-Based Approach

The Master's dissertation "An Environment-Based Approach to the Composition of Safe Systems" [Hin93] introduced a set of comprehensive safety properties, referred to as confidentiality, availability, authenticity, faithfulness, utility, and provenance. Levels of optimism of the confidentiality property are introduced and motivated by the need to account for the different levels of trust in the environment and components within the system.

The rôle of the environment was included in the definition of the components and of the safety properties. Each component exists within a given environment. The users of a component are part of the environment, as are all events that act on components. Thus during composition, a component's environment must also be considered as it will have an effect on other components in the composition. This approach does not suffer from the same downfall as many previous approaches where the "success" of the safety is limited to a few predetermined environments.

2.5 Progress

Progress properties differ from safety properties in that the formal definition of a progress property will contain the unary temporal operator $\Diamond$, "eventually"; safety properties do not contain $\Diamond$. The formula $\Diamond A$ is true if there is some time in the future (possibly including the present) where A is true. Progress properties are concerned with the eventuality of an action, indicating the progression of its execution.

The difference between liveness and progress has not been clearly articulated in previous work. Liveness properties may be expressed as progress properties; not all progress properties can be expressed as liveness properties [MaPn92]. Initially, liveness properties dealt only with program termination. Liveness was soon extended to include such characteristics as "request for service will eventually be answered", "message will eventually reach its destination" [OwLa82], "total correctness", "intermittent assertions", "accessibility", and "liveness" [MaPn81].

At some point, the notion of freedom from livelock became synonymous with liveness. Indeed, Manna and Pnueli initially refer to eventuality properties, with "liveness" (freedom from livelock) defined as one of these properties [MaPn81]. Later work on liveness, in particular by Manna and Pnueli [MaPn92], has replaced the idea of liveness with the notion of *progress*, defined formally be the formal structure of the property.

2.5.1 The Temporal Logic of Programs

"The Temporal Logic of Programs" [Pnu77], is recognised as the first important paper to apply temporal logic to computing. In this paper, Pnueli argues in favour of the expressiveness of temporal logics when dealing with the specification and verification of concurrent programs. Pnueli motivates this using the idea of *eventuality*, which "in its full generality [this] denotes a dependence in time in the behavior of the program" [Pnu77], and correctness of programs.

In Pnueli's work an execution is a sequence of states related by a transition relation between states. This system description is applied to sequential and concurrent programs by Pnueli. Pnueli adapts the temporal logic K_b (the modal logic S4) for temporal reasoning about concurrent programs. "This system seems much more satisfying and able to model the more intricate reasoning involved in proving temporal correctness of concurrent programs" [Pnu77].

Since Pnueli's work was published, there have been many papers published applying temporal logics to program specification and verification, including [Aba87], [Aba88], [AlSc85a], [Kro87], and [MaPn81].

2.5.2 Classes of Progress Properties

Manna and Pnueli have defined classes of progress properties, based on the structure of the corresponding formulae [MaPn92]. The classes of progress properties are defined as guarantee, obligation, response, persistence, and reactivity. This division is made according to the structure of the formulae, and is given in Table 2.1. Each class defines a different structure of formulae, with some of the classes being proper subsets of the other classes.

Guarantee Formulae: $\Diamond p$

Guarantee formulae ensure that something will eventually happen at least once. Guarantee properties are closed under union (disjunction) and intersection (conjunction); the conjunction or disjunction of two guarantee formulae is a guarantee formula. Guarantee formulae are the dual of safety formulae: all guarantee formulae can be converted to safety formulae with the identity $\circ\, p = \neg \Diamond \neg p$. A guarantee (safety) may not be the dual of any well-known safety (guarantee) formulae.

Class of Property	**Closure under**		**Class Schema**	**Remarks**
	Union	**Intersection**		
Guarantee	Yes	Yes	$\Diamond p$	Dual of safety
Obligation	Yes	No	$\circ\, p \vee \Diamond q$	Strictly includes classes of safety and guarantee
General Obligation	Yes	Yes	$\bigwedge_{i=1}^{n} (\circ\, p \vee \Diamond q)$	
Response	Yes	Yes	$\circ \Diamond p$ $\circ\, (p \rightarrow \Diamond q)$	Strictly includes class of obligation
Persistence	Yes	Yes	$\Diamond \circ\, p$ $\circ\, (p \rightarrow \Diamond \circ\, q)$	Dual of response
Reactivity	Yes	No	$\circ \Diamond p \vee \Diamond \circ\, q$	Strictly includes classes of response and persistence
General Reactivity	Yes	Yes	$\bigwedge_{i=1}^{n} (\circ \Diamond p \vee \Diamond \circ\, q)$	

Table 2.1: Classification of Progress Properties According to Formulae Structure

Obligation Formulae: $(\circ\, p \vee \Diamond q)$ or $(\Diamond r \rightarrow \Diamond q)$

Obligation formulae ensure that if something eventually happens, then some (related) event must also eventually happen. Obligation formulae do not make any statements about the relative temporal ordering of these two events. Obligation formulae are closed under disjunction. The conjunction of two obligation formulae does not give an obligation formula, but instead gives a stronger formula that includes the original obligation formulae. Manna and Pnueli therefore define general obligation formulae as follows: $\bigwedge_{i=1}^{n} (\circ\, p \vee \Diamond q)$. This definition of general obligation is closed under conjunction.

The class of obligation formulae strictly includes the classes of safety and guarantee formulae. If we have defined all possible obligation formulae for a system, we also have defined all possible safety and guarantee formulae.

Response Formulae: $(\circ \Diamond p)$ or $\circ\, (p \rightarrow \Diamond q)$

Response formulae are used to specify that some event always occurs sometime after another event. Thus if a certain response is always required for a given event, this can be specified with a response formula. Response formulae are closed under union and intersection.

The class of response formulae strictly includes the classes of safety and guarantee formulae and therefore also strictly includes the class of obligation formulae. There are response formulae that can not be derived by any Boolean connection of obligation formulae, however, for example, $\circ \Diamond$p.

Persistence Formulae: $(\Diamond \circ \mathrm{p})$ or $\circ\, (\mathrm{p} \rightarrow \Diamond \circ \mathrm{q})$

Persistence formulae usually describe the eventual stabilisation of a state. They allow for arbitrary delay before this state is reached, but require that this state be constantly maintained once reached. Persistence formulae are closed under union and intersection. The class of persistence formulae is the dual of the class of response formulae. All closure and inclusion properties of one class can be derived from the properties of the other class.

Reactivity Formulae: $(\circ \Diamond p \vee \Diamond \circ q)$ or $(\circ \Diamond r \rightarrow \circ \Diamond p)$

Reactivity formulae are used to abstract a situation in which we wish to commit a system to eventually respond to sufficiently many stimuli, but not specify a bound on how many stimuli may happen before we have an eventual response. Reactivity formulae are closed under union (disjunction) but not under intersection (conjunction) or complementation. As with obligation, Manna and Pnueli define a general reactivity formula, $\bigwedge_{i=1}^{n} (\circ \Diamond p \vee \Diamond \circ q)$.

The class of reactivity formulae includes the classes of response and persistence formulae. Each reactivity formula can be obtained by a Boolean connection of persistence and response formulae.

3. Temporal Logic

Take heart, I said to myself:
don't think of Wisdom now;
ask the help of Science.
UMBERTO ECO: Foucault's Pendulum

In this chapter we provide a brief summary of temporal logic. The use of a formal language requires that we use a rigorous and structured approach for the specification of systems and properties. This will in turn help us to avoid errors that arise from the use of natural languages (such as English) to describe systems and properties.

We begin by introducing and defining the syntax and semantics of temporal logic, including the temporal operators $\circ$, "always", and $\diamond$, "eventually." We define how to evaluate the formulae of this logic. These formulae will be used to describe the desired properties and actual behaviours of a system. We use this approach to prove formally that a system's behaviours (represented formally) satisfies the (formal) properties.

We also introduce the notion of a filtration on the language. Filtrations are used to restrict the language to subsets of the possible formulae of the original language. We will use filtrations to allow us to evaluate subsystems of the original system (or submodels of the model of the language).

3.1 Syntax of the Language

The language of temporal logic is defined using Backus-Naur form (BNF), where p is a typical member of a numerable set of (atomic) formulae Φ. $\mathcal{F}ma\{\Phi\}$ is the set of all formula generated from Φ. The syntax of the logic is given as:

Atomic formulae: $p \in \Phi$

Formulae: $A \in \mathcal{F}ma\{\Phi\}$

$A ::= p \mid \bot \mid A_1 \Rightarrow A_2 \mid \circ A \mid \mathrm{O}A \mid A_1 \,\mathcal{U}\, A_2 \mid \exists u \cdot A \mid \forall u \cdot A$

The symbols $\bot$ (false) and $\Rightarrow$ (implication) are the classical logic connectives from which the remaining standard logic connectives, $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\Leftrightarrow$ (if and only if) are derived. The symbols $\circ$ (always in the future), and, O (next time in the future) are unary temporal connectives. $\mathcal{U}$ (until) is a binary temporal connective. The unary temporal operator $\Diamond$ (eventually in the future) is defined on $\circ$ by the tautology $\Diamond A = \neg \circ \neg A$ (or $\neg\Diamond\neg A = \circ A$). The symbols $\exists$ (existence) and $\forall$ (universality) are quantifiers.

Temporal operators may deal with the past, the future, or all time. Thus the modality $\circ$ may refer to always in the past, always in the future, or always over all time. We will use the non-strict future interpretation of always, always in the future, including the present. Thus we will also use the non-strict future interpretation of eventually, eventually in the future, including the present.

3.1.1 Formulae

Every proposition $p \in \Phi$ is an *atomic formula*. A *Boolean formula*, A, is defined as follows:

$A ::= p \mid \bot \mid A_1 \Rightarrow A_2$

A *quantified formula* is a Boolean formula quantified over a free or rigid variable of the formulae:

$A ::= \exists u \cdot A \mid \forall u \cdot A$

A *state formula* includes all Boolean formulae and the quantified formulae so that:

$A ::= p \mid \bot \mid A_1 \Rightarrow A_2 \mid \exists u \cdot A \mid \forall u \cdot A$

A *state formula* is one that can be evaluated for a given position in a sequence (generated by the model). It expresses properties of the state of the sequence at a given instance.

These types of formulae, atomic, Boolean, quantified, and state are sometimes referred to as *static* formulae as they do not embody the notion of time. *Temporal formulae* are boolean formulae with temporal operators

$A ::= p \mid \bot \mid A_1 \Rightarrow A_2 \mid \circ A \mid \mathrm{O}A \mid A_1 \,\mathcal{U}\, A_2$

3.2 Semantics of the Language

The semantics of the language are defined by a *model* of the language. A model is a triple $\mathcal{M} = (\mathcal{S}, \sigma, \mathcal{V})$, where $\mathcal{S}$, σ and $\mathcal{V}$ are defined as follows:

$\mathcal{S}$ is a numerable set of boolean formulae,

$$\mathcal{S} \in \mathcal{F}ma\{\Phi\}, \mathcal{S} ::= p \mid \bot \mid \mathcal{S}_1 \Rightarrow \mathcal{S}_2$$

σ is a surjective function, $\mathbb{N} \to \mathcal{S}$, enumerating $\mathcal{S}$ as a sequence over the natural numbers $\mathbb{N}$:

$$\sigma_0, \sigma_1, \ldots, \sigma_n, \ldots$$

$\mathcal{V}$ is a valuation on all formulae, $\mathcal{V}$: $\mathcal{F}ma\{\Phi\} \to$ {true, false} for a fixed $s \in \mathcal{S}$ such that

$$\mathcal{V}_s(p) = \begin{cases} \text{true} & \text{if } s \in \mathcal{V}(p) \\ \text{false} & \text{otherwise} \end{cases}$$

Informally, $\mathcal{V}(p)$ is the valuation of the set of points s at which the proposition p is true.

Given a model, $\mathcal{M} = (\mathcal{S}, \sigma, \mathcal{V})$, and a Boolean formula, A, the relation "A is true at the j-th state σ_j in $\mathcal{M}$", written $(\sigma,j) \models A$ or $\sigma \models_j A$, is defined inductively on the formation of $A \in \mathcal{F}ma\{\Phi\}$ as follows:

$\sigma \models_j p$ iff $\sigma_j \in \mathcal{V}(p)$

$\sigma \models_j \bot$

$\sigma \models_j A_1 \Rightarrow A_2$ iff $\sigma \models_j A_1$ implies $\sigma \models_j A_2$

Temporal formulae are defined inductively based on Boolean formulae as follows:

$\sigma \models_j \bigcirc A$ iff $\sigma \models_{j+1} A$

$\sigma \models_j \circ A$ iff for all $k \geq j$, $\sigma \models_k A$

$\sigma \models_j A_1 \mathcal{U} A_2$ iff for some $k \geq j$, $\sigma \models_k A_2$ and for every i such that $j \leq i \leq k$, $\sigma \models_i A_1$

First-order (quantified) formulae are satisfied by a U-valuation on $\mathcal{V}$ as follows, where the U-valuation is a function assigning to each variable u a value $\mathcal{V}(u)$ in X:

$\sigma \models_j \exists u \cdot A$ iff $\exists \chi : X \cdot \sigma \models_j A[\mathcal{V}(u/\chi)]$

$\sigma \models_j \forall u \cdot A$ iff $\forall \chi : X \cdot \sigma \models_j A[\mathcal{V}(u/\chi)]$

A model is contained within the frame $\mathcal{F} = (\mathcal{S}, \sigma)$. A model can be thought of as a particular instantiation of a frame given a particular valuation, $\mathcal{V}$. Thus the application of different valuations to a given frame will result in different models of the frame.

3.2.1 Satisfiability and Validity

Satisfiability and validity of formulae indicate where and when they are *true* within the model. A formula A *holds*, or is true, at a given position in a sequence if and only if:

$$\sigma \models_j A$$

This formula is in turn *satisfied* by the model if and only if it holds for all points in the sequence:

$$\forall j : \mathbb{N} \cdot \sigma \models_j A$$

The same formula is *valid* in a frame, $\mathcal{F} \models A$, if and only if it is satisfied by all models within the frame $\mathcal{F} = (\mathcal{S}, \sigma)$:

$$\forall \mathcal{V} : \mathcal{F} \cdot \forall j : \mathbb{N} \cdot \sigma \models_j A$$

If the formula A is valid then it will be true no matter how we interpret the propositions of the language.

3.3 A Proof System

It is impossible to list all possible valid temporal formulae. Instead, deductive systems are used to provide a means of identifying the valid statements of a logic. A deductive system consists of the axioms (formulae that are always true) and a set of rules, defining how to derive new, valid formulae from existing valid formulae. A deductive system is therefore used to determine if formulae hold at a given instance, are satisfied by a model, or valid on a frame.

The main rules of inference and a brief discussion of the proof system used in this work is included in Appendix A.

3.4 Filtrations

Filtering is the process of collapsing a model using equivalence relations. In this work we use filtrations to produce submodels that represent subsequences within the system by filtering out unwanted events from consideration. This is based on the definitions given by Goldblatt [Gold92]. We will filter the set of atomic events Φ and thereby reduce the possible state formulae that may be derived from Φ.

A filtration set Γ of the set of formulae A is defined relative to the subformulae $\mathcal{Sf}(\mathrm{A})$. In addition to the subformulae of A, the filtration set must not filter out formulae necessary to maintain the closure properties of the filtration submodel. Given $p \in \Phi$, $\mathrm{A} \in \mathcal{Fma}\{\Phi\}$ and $\Gamma \subseteq \Phi$, if $p \in \Gamma$ then $\mathrm{A} \in \mathcal{Fma}\{\Gamma\}$, where $\mathrm{A} ::= p \mid \perp \mid \mathrm{A}_1 \Rightarrow \mathrm{A}_2$.

Definition: Filtration Set Γ [2]

The filtration set Γ is finite and has the following closure properties:

$\mathrm{A} \in \Gamma$;

Γ is closed under subformulae;

The filtration set, Γ, is therefore given as:

$\Gamma = \mathcal{Sf}(\mathrm{A})$

---

2 This definition is based on the definition given by Goldblatt (p. 92). We do not include the additional closure requirements for modalities specified by Goldblatt because the filtration set is defined on the subformulae of modality and quantifier-free formulae. The formulae of A do not contain temporal operators (modalities) and do not contain quantifiers. The closure of the filtration set therefore follows.

The filtration theorem tells us that a formula B will belong to the sequence σ over the filtered model if and only if it belongs to the filtration set Γ. Thus we will consider only events belonging to the filtration set in the filtration model and associated sequences. Closure of the filtered set Γ ensures that the subformulae of the formulae of A are also satisfied by the filtration model.

Filtration Theorem (Theorem 9.13 of Goldblatt, [Gold92])

Let $\mathcal{M}_\Gamma = (\mathcal{S}_\Gamma, \sigma, \mathcal{V}_\Gamma)$. If $B \in \Gamma$ then for any $j \in \mathbb{N}$ and $s \in \sigma_j$,

$B \in s$ iff $\mathcal{M}_\Gamma \models_j B$

3.5 Evaluation of Filtered Sequences

In this work we will often need to examine the subsequences of a system (model). The general sequence is given as σ, as evaluated by the formal model. A subsequence between a given source ($\mathbb{S}$) and destination ($\mathbb{D}$) is given as $\sigma(\mathbb{S};\mathbb{D})$.

Given the arbitrary source, $\mathbb{S}$, and arbitrary destination, $\mathbb{D}$, the relation "A is true at the *j*-th state σ_j in $\mathcal{M}_\Gamma$ where $\Gamma = \Phi(\mathbb{S}(\text{out}),\mathbb{D}(\text{in}))$", written $\sigma(\mathbb{S};\mathbb{D}) \models_j A$, is defined inductively on the formation of $A \in \mathcal{F}ma\{\Phi(\mathbb{S}(\text{out}),\mathbb{D}(\text{in}))\}$ (this notation on Φ is defined in detail in Chapter 4) as follows:

$$\sigma(\mathbb{S};\mathbb{D}) \models_j p \quad \Leftrightarrow \quad \sigma_j \in \mathcal{V}(p),\ s \in \mathcal{V}(p) \text{ and } s \in \Phi(\mathbb{S}(\text{out});\mathbb{D}(\text{in}))$$

$$\sigma(\mathbb{S};\mathbb{D}) \models_j A_1 \Rightarrow A_2 \quad \Leftrightarrow \quad \sigma \models_j A_1 \text{ implies } \sigma \models_j A_2$$
$$\text{where } A_1, A_2 \in \Phi(\mathbb{S}(\text{out});\mathbb{D}(\text{in}))$$

and so on.

The evaluation of quantified formulae deserves special attention:

$$\sigma(\mathbb{S};\mathbb{D}) \models_j \exists u : U \cdot A \quad \Leftrightarrow \quad \exists u : U \cdot \sigma(\mathbb{S};\mathbb{D}) \models_j A$$
$$\text{where } \exists u : U \cdot \sigma(\mathbb{S};\mathbb{D}) \models_j A(u)$$

$$\sigma(\mathbb{S};\mathbb{D}) \models_j \forall u : U \cdot A \quad \Leftrightarrow \quad \forall u : U \cdot \sigma(\mathbb{S};\mathbb{D}) \models_j A$$
$$\text{where } \forall u : U \cdot \sigma(\mathbb{S};\mathbb{D}) \models_j A(u)$$

Filtrations and subsequences will be discussed in more detail in Chapter 4.

3.6 Choice of Language

We have chosen to use the language of Goldblatt's future-time temporal logic of concurrency over other logics, such as Manna and Pnueli's past and future-time temporal logic [MaPn92], Abadi and Lamport's temporal logic of refinement [AbLa90], or such well-established logics as K_b, as originally used by Pnueli [Pnu77]. The temporal logic is used as the basis for the definitions of security presented in this dissertation. This formal approach is used to ensure that we avoid the ambiguities and nuances of natural language, encountered with informal (English) descriptions of security and security properties.

The temporal language used by Manna and Pnueli includes the language of the temporal logic of concurrency, as well as additional past-time temporal operators. We have chosen to use the simpler future-only temporal language presented by Goldblatt [Gold92]. The future axioms given by Manna and Pnueli are also axioms of Goldblatt's language. The interested reader is referred to Manna and Pnueli for a complete listing of axioms of the temporal language.

Lamport's temporal logic of actions is "adequate for describing and proving simple safety and liveness properties of a single [concurrent] program." [Lam90]. We chose not to use this logic for several reasons, the strongest being that it is designed to consider safety and liveness according to the Alpern and Schneider classification, whereas we consider safety and progress, where not all progress properties can be expressed as liveness properties. Also, as Lamport points out in this paper, "there are useful properties that [the logic] cannot express, it does not support proofs that one program implements another, and it does not permit a simple compositional semantics of programs." [Lam90].

The logic S4 used by Pnueli is defined by the schemata K, T (reflexivity) and 4 (transitivity). It does not include the temporal operator $\bigcirc$, or $\mathcal{U}$, nor the schemata associated with these operators. We believe that the inclusion of the next-time operator and the until operator will be useful during the specification and verification of temporal security properties of a system.

4. System Specification

'Gentlemen, please, behave yourselves.
You can't fight in here ...This is the war room!'
TERRY SOUTHERN, STANLEY KUBRICK: Dr. Strangelove

In this chapter we introduce an event and trace-based notation used to describe systems and system behaviour. Using this notation we describe components, systems of components, the environment of components and systems, and their behaviours. The notation used is based on that introduced by McLean [McLe94] and incorporates the temporal language introduced in the previous chapter.

A system is represented by a model in the temporal language. The possible sequences of the model correspond to sequences of events received at and produced by a system. These sequences may consist of multiple sequences within the system. We use the filtrations introduced in the previous chapter to filter out unwanted sequences to examine a particular subsequence.

This notation is used to represent formally the behaviour of components and interconnections of components. We define the relationship of inputs and outputs for a given component, and the behaviour axiom that incorporates the state of the component and the eventuality of output events given input events. These behaviour axioms define the predictable behaviour of the component and systems.

We discuss the interconnection of systems into larger systems and introduce three composite structures from which most composite systems can be derived. This approach, based on temporal models of the system and sequences within the system, allows us to easily represent the temporal, causal and simultaneity relationships between sequences of events within a system.

4.1 Specification of Events

An *event* is a model-relevant action. We are dealing with safety and progress issues, and therefore the model relevant actions will be security relevant actions. The granularity of an event is defined by the modeller. The set of events is the set of all model relevant actions that act on or are produced by a system. The set of all events describes the domain in which a system exists; it defines all possible security relevant actions for the system under consideration.

Formally, events are treated as propositions, meaning that they are true or false. If an event occurs, then the corresponding proposition is true. If an event does not occur then the corresponding proposition will be false. These types of propositions are sometimes referred to as (propositional) fluents [McHa69] because their truth value is fluent, changing over time.

Events have many different characteristics based on their effect on a system. An event will be either *relevant* (i.e., it will act as a stimulus) or *irrelevant* (i.e., it has no direct or indirect effect). Relevant events may be broken down into the classes of input or output, acceptable or unacceptable, and, either observation exclusive modification. These classes are defined as:

Irrelevant: Will not have any direct or indirect effect

Relevant: May act as a stimulus

Unknown: An event that is not previously known or defined. We must assume that all such events are *unacceptable*.

Known:

Input: May be received by a system

Output: May be produced by a system

Acceptable: Will never cause a bad thing to happen

Unacceptable: May cause a bad thing to happen

Observation: Has the effect of allowing observation of information

Modification: Has the effect of allowing modification of information

The classes of these events are defined as subsets on the set of events for the system. They can be thought of as filtrations on the set of events. Thus, Φ(obs) filters out all events that are not observation events. The classes are represented as follows:

- Input or Output: Φ(in), Φ(out)
- Acceptable or Unacceptable: Φ(A), Φ(U)
- Observe or Modify: Φ(obs), Φ(mod)

Because of the predictable nature of systems, we can define the relationship between input and output events at a system. A given event may act as both an input and an output at the same system. The same event may act as an output at one system and an input to another.

A given event can be classed as one and only one of acceptable or unacceptable at a system. The same event may be acceptable at one system and unacceptable at another system.

We define all output events produced by a component in response to an acceptable event to be acceptable events. A component cannot produce an event that is unacceptable to that component as an output in response to an acceptable input.

An unacceptable event may be rendered *harmless* at a component. Once rendered harmless, an event will remain harmless. Thus the set of harmless events may be considered to be a subset of the set of acceptable events.

For the purposes of this work, we will assume that all events can be uniquely classified as either observation or modification events[3]. All events in the system can be classified as one and only one of a request to observe information or a request to modify information. The classification of *observation* events or *modification* events is independent of the previously discussed classifications of events. This classification of events is constant across systems and is not mutable. The sets of events generated by the filtrations Φ(obs) and Φ(mod) are distinct, that is, $\Phi(\text{obs}) \cap \Phi(\text{mod}) = \varnothing$.

There may be events that are both observation and modification events, such as spin-locks. We do not explicitly address this category of events. We do postulate that such events will have unique characteristics and that these can be handled by the use of a third, mutually exclusive category, *both*, defined as $\Phi(\text{obs}) \cap \Phi(\text{mod}) \cap \Phi(\text{both}) = \varnothing$.

The set of events with multiple, common characteristics is defined by the intersection of the individual subsets. Instead of using the long-hand notation of

$\Phi(\text{characteristic 1}) \cap \Phi(\text{characteristic 2}) \cap \ldots$

we will write

$\Phi(\text{characteristic 1, characteristic 2, } \ldots)$.

The subset of events relevant to component **i** are given as the subset Φ(i). For characteristics that are specific to a given component **i**, we write

$\Phi(\text{i}) \cap \Phi(\text{characteristic 1}) \cap \Phi(\text{characteristic 2})$

which is in turn represented as

$\Phi(\text{ i(characteristic 1, characteristic 2) })$.

This notation is used to indicate that the characteristics 1 and 2 are characteristics of component **i**, even though they may not be characteristics of the other components within a system.

The representation of a component-level characteristic for all components in a system is given as $\Phi(\mathbf{N}(\text{characteristic}))$ where there are $\mathbf{N}+1$ components in the system. For example, the set of acceptable events common to all components in a $\mathbf{N}+1$-component system is represented as $\Phi(\mathbf{N}(\text{A}))$, where $\mathbf{n}=\mathbf{N}$, and $\Phi(\mathbf{N}(\text{A})) = \Phi(0(\text{A})) \cap \Phi(1(\text{A})) \cap \ldots \cap \Phi(\mathbf{n}(\text{A}))$.

There is one event that belongs to all categories of relevant events described above. This is the *null* event. This event has no effect on the system. This event is used as a modelling abstraction to represent the lack of an event. The *null* event has the following characteristics:

---

3 With the exception of the null event. This will be discussed in detail later in this section.

$null \in \Phi$

$\forall i \in \mathbb{N} \cdot null \in \Phi(i)$

$null \in \Phi(\text{input})$ $\qquad null \in \Phi(\text{output})$

$null \in \Phi(A)$ $\qquad null \in \Phi(U)$

$null \in \Phi(\text{obs})$ $\qquad null \in \Phi(\text{mod})$

This has the effect of changing the definition of $\Phi(\text{obs}) \cap \Phi(\text{mod})$ given earlier. Because the *null* event has no effect, it does not violate the spirit of the previous definition that requires the sets of observation and modification events to be distinct. Nevertheless, we must re-define $\Phi(\text{obs}) \cap \Phi(\text{mod})$ and $\Phi(\text{obs}) \cap \Phi(\text{mod}) \cap \Phi(\text{both})$ as follows:

$\Phi(\text{obs}) \cap \Phi(\text{mod}) = null$

$\Phi(\text{obs}) \cap \Phi(\text{mod}) \cap \Phi(\text{both}) = null$

4.2 Specification of Sequences

A sequence (trace) is an ordered collection of state formulae representing the input and output events to a component (system) over time. In security applications a trace is often meant to be a valid sequence (i.e., safe or live). We will use the term sequence instead of trace as it reinforces the use of the formal model introduced in the previous chapter.

For a simple, stand-alone component (one that sends and receives events from the environment only), the sequence σ has the form:

$\sigma = \langle \langle in_0 , out_0 \rangle, \langle in_1 , out_1 \rangle, ... \rangle$

where in_x and out_x represent the inputs and outputs to the component at time x. For a system with $\mathbb{N}$ components, where $\mathbf{n{=}\mathbb{N}}$ the sequences have the form:

$\sigma = \langle \langle in^0_0, out^0_0 , in^1_0 , out^1_0 , ... , in^{n-1}_0 , out^{n-1}_0 \rangle,$

$\langle in^0_1, out^0_1 , in^1_1 , out^1_1 , ... , in^{n-1}_1 , out^{n-1}_1 \rangle, ... \rangle$

where in^i_x and out^i_x represent the inputs and outputs respectively, to component **i** at time x. If we wish to consider a given instance s_x within a sequence, we use the notation "$\models_x$", "entails at x", where

$\sigma \models_x A \Leftrightarrow s_x = A$

and $A = (in^0_x \wedge out^0_x \wedge in^1_x \wedge out^1_x \wedge ... \wedge in^{n-1}_x \wedge out^{n-1}_x)$ and is defined inductively on Φ.

There are several instances where we may require more detailed information, that is, more "specialised" subsequences of σ. The trace of inputs or outputs of a given sequence is represented as:

$\sigma(\text{in}) = \langle \langle in^0_0, in^1_0 , ... , in^{n-1}_0 \rangle, \langle in^0_1, in^1_1 , ... , in^{n-1}_1 \rangle, ... \rangle$

$\sigma(\text{out}) = \langle \langle out^0_0, out^1_0 , ... , out^{n-1}_0 \rangle, \langle out^0_1, out^1_1 , ... , out^{n-1}_1 \rangle, ... \rangle .$

We may wish to examine the output subsequence sent by component **i** to a particular destination. The source and destination of sequences are represented using the notation ($\mathbb{S}$;$\mathbb{D}$), where the source or destination may be the environment or a component, or some combination of the two. A sequence between components **i** and **j**, where the outputs of **i** are sent as input to **j**, is described as follows:

$$\sigma(i; j) = \langle \langle in^j_0 = out^i_0 \rangle, \langle in^j_1 = out^i_1 \rangle, ... \rangle$$

If there are multiple input or output channels at a component, then in^i_x and out^i_x represent all inputs or outputs occurring on all input or output channels (respectively) at component **i** at time x. For a component with m distinct channels, where there are ℓ input channels, 0, 1, ..., ℓ-1 and m-ℓ-1 output channels, ℓ, ℓ+1, ..., m-1, then σ[channel]:

$$\sigma[k] = \langle \langle in_0[k] \rangle, \langle in_1[k] , ... \rangle \quad \text{iff } k < \ell$$

$$\sigma[k] = \langle \langle out_0[k] \rangle, \langle out_1[k] , ... \rangle \quad \text{iff } k \geq \ell$$

where

$$\sigma(in) = \sigma[0] \wedge \sigma[1] \wedge ... \wedge \sigma[\ell\text{-}1]$$

$$\sigma(out) = \sigma[\ell] \wedge \sigma[\ell+1] \wedge ... \wedge \sigma[m\text{-}1] .$$

We need to separate the channels from the ($\mathbb{S}$;$\mathbb{D}$) subsequences of the system for several reasons. It is possible that a component will have more than one channel to or from another component in the system, such as in multi-level security systems. It is also possible that a channel will be sent to several destinations, such as in broadcast systems.

The sequence of inputs from all sources or outputs to all destinations at a component **i** is given as $\sigma\{\mathbf{i}\}(in)$ and $\sigma\{\mathbf{i}\}(out)$, respectively.

$$\sigma\{\mathbf{i}\}(in) = \quad \langle \langle in^i_0 \rangle, \langle in^i_1 \rangle , ... \rangle$$

$$\sigma\{\mathbf{i}\}(out) = \quad \langle \langle out^i_0 \rangle, \langle out^i_1 \rangle , ... \rangle$$

If a component has m input and output channels, ℓ-1 input channels and m-ℓ-1 output channels then

$$\sigma\{\mathbf{i}\}(in) = \sigma\{\mathbf{i}\}(in)[0][1]...[\ell\text{-}1]$$

$$= \langle \langle in^i_0[0] \wedge in^i_0[1] \wedge ... \wedge in^i_0[\ell\text{-}1] \rangle, \langle in^i_1[0] \wedge in^i_1[1] \wedge ... \wedge in^i_1[\ell\text{-}1] \rangle , ... \rangle$$

$$\sigma\{\mathbf{i}\}(out) = \sigma\{\mathbf{i}\}(out)[\ell][\ell+1]... [m\text{-}1] .$$

A sequence of events from multiple sources (or to multiple destinations) will use the superscript '+' to "join" the sources or destinations. The script ($\mathbb{S}^+\mathbb{S}$; $\mathbb{D}$) or ($\mathbb{S}$; $\mathbb{D}^+\mathbb{D}$) is used. If we wish to represent all components in the system, we will use the notation $\mathbb{N}$ instead of listing the conjunction of all components, ($0^+1^+...^+n$-1). For example, the sequence of events from the environment and from component **i** to component **j** is given by $\sigma(\mathbb{E}^+i;j)$. The sequence from all components to the environment is $\sigma(\mathbb{N};\mathbb{E})$.

If we are given a "conjoined" sequence where we do not know which events occur as part of which subsequence, we must consider all possible events when examining a particular subsequence of the conjoined sequence. For example, if we are given the sequence $\sigma(i^+j;\mathbb{E})$

such that at time x the event α occurs, then either or both of the individual subsequences may entail event α at time x.

$$\sigma(i^{+}j;\text{Ɇ}) \models_{x} \alpha \Rightarrow \sigma(i;\text{Ɇ}) \models_{x} \alpha \vee \sigma(j;\text{Ɇ}) \models_{x} \alpha$$

If at time x the events $(\alpha \vee \beta)$ occur in the sequence $\sigma(i^{+}j;\text{Ɇ})$ then

$$\sigma(i^{+}j;\text{Ɇ}) \models_{x} (\alpha \vee \beta) \Rightarrow \sigma(i;\text{Ɇ}) \models_{x} (\alpha \vee \beta) \ \vee \sigma(j;\text{Ɇ}) \models_{x} (\alpha \vee \beta)$$
$$\vee \, (\sigma(i;\text{Ɇ}) \models_{x} \alpha \wedge \sigma(j;\text{Ɇ}) \models_{x} \beta \,) \vee (\, \sigma(i;\text{Ɇ}) \models_{x} \beta \wedge \sigma(j;\text{Ɇ}) \models_{x} \alpha \,) \,.$$

Figure 4.1, below, is a snapshot of a system with two components, 0 and 1, at time t=x.

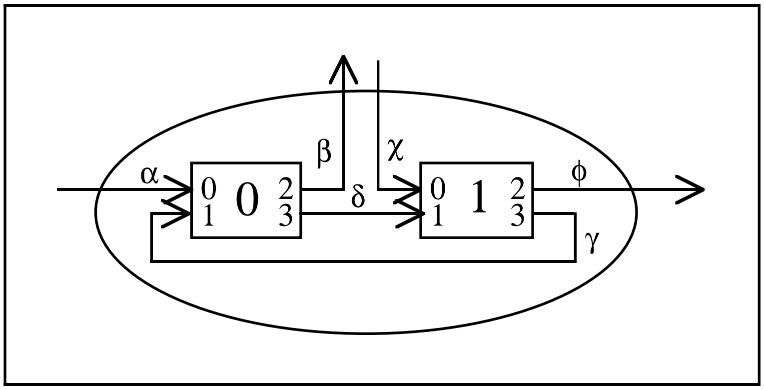

Figure 4.1: Sequences and Subsequences of Composite System

The general sequences define the system:

$\sigma \models_{x} (\alpha \wedge \beta \wedge \chi \wedge \delta \wedge \phi \wedge \gamma)$

$\sigma(\text{Ɇ};0) \models_{x} \alpha$

$\sigma(1;\text{Ɇ}) \models_{x} \phi$ $\qquad$ $\sigma(1;0) \models_{x} \gamma$

$\sigma(0^{+}1;\text{Ɇ}) \models_{x} (\beta \wedge \phi)$ $\qquad$ $\sigma(\mathbf{N};\text{Ɇ}) \models_{x} (\beta \wedge \phi)$

$\sigma(\text{Ɇ};\mathbf{N}) \models_{x} (\alpha \wedge \chi)$ $\qquad$ $\sigma(\text{Ɇ};0^{+}1) \models_{x} (\alpha \wedge \chi)$

$\sigma\{0\}(in) \models_{x} (\alpha \wedge \gamma)$ $\qquad$ $\sigma\{0\}(in)[0][1] \models_{x} (\alpha \wedge \gamma)$ $\qquad$ $\sigma(\text{Ɇ}^{+}1;0) \models_{x} (\alpha \wedge \gamma)$

$\sigma\{1\}(in) \models_{x} (\chi \wedge \delta)$ $\qquad$ $\sigma\{1\}(in)[0][1] \models_{x} (\chi \wedge \delta)$ $\qquad$ $\sigma(\text{Ɇ}^{+}0;1) \models_{x} (\chi \wedge \delta)$

$\sigma(in) \models_{x} (\alpha \wedge \chi \wedge \delta \wedge \gamma)$

$\sigma\{0\}(out) \models_{x} (\beta \wedge \delta)$ $\qquad$ $\sigma\{0\}(out)[0][1] \models_{x} (\beta \wedge \delta)$ $\qquad$ $\sigma(0;\text{Ɇ}^{+}1) \models_{x} (\beta \wedge \delta)$

$\sigma\{1\}(out) \models_{x} (\phi \wedge \gamma)$ $\qquad$ $\sigma\{1\}(out)[0][1] \models_{x} (\beta \wedge \delta)$ $\qquad$ $\sigma(1;\text{Ɇ}^{+}0) \models_{x} (\phi \wedge \gamma)$

$\sigma(out) \models_{x} (\beta \wedge \delta \wedge \phi \wedge \gamma)$

4.2.1 Sequence of States

The internal variables of a system represent the current state of that system. Each input to a system produces an output and may cause a change of the state of the system. The state of a system may change when a process completes, when a process produces an output, or when an input is received. State is often derived from the history of the input events received at a system given an initial internal state. We take the approach of Millen: "If you prefer not to worry about the entire history of inputs to the machine [system] since its initial state, there is an alternative solution: consider the current state. The current state has all other information needed to determine the next [Y-] output." [Mil89]

Each input event will initiate processing at a system. This will be represented as a process, distinct from other processes, with the same name as the input event. The state of a component within a system is represented by:

$\sigma\{\text{component}\}$

$= \sigma\{\mathbf{i}\} = \langle \langle \text{state}_0 \rangle , \langle \text{state}_1 \rangle , \ldots \rangle .$

4.2.2 Subsequences of Sequences

The use of subsequences of the general system sequence corresponds to *filtrations* on the system model. In our approach, filtrations are used to remove from consideration all events that are not relevant to a given component or characteristic. We will represent the filtration set as the set G, where G is a subset of the set of state formulae derived from the set of atomic events Φ. We require that each occurrence of an event as a possible event between two distinct components have a unique name, in order to ensure that we filter all events that are not relevant to a given subsequence.

For example, let Φ be the set of all events for a given system. We wish to consider the sequence of input events at component **i**. The set of such events is defined as Φ(i(inputs)). The filtration on the system model is therefore defined as

$G = \mathcal{F}ma\{\ \Phi(i(inputs))\ \}$

$\Gamma = \mathcal{S}f\,(\mathcal{F}ma\{\ \Phi(i(inputs))\ \}\ ) .$

Thus given

$\sigma(\mathbb{E}^{+}\mathbb{N};\mathbb{N}^{+}\mathbb{E}) \models_{X} (\alpha \wedge \beta \wedge \chi)$ and $A = \mathcal{F}ma\{\Phi(i(inputs))\} = \{\ \beta, \delta, \phi, \eta\ \}$

we can write

$\sigma(\mathbb{E}^{+}\mathbb{N};i) \models_{X} \beta .$

The restriction of formulae to Boolean formulae is realistic, since in practice a system could not test the trust value of a formula involving quantification of variables ranging over infinite sets [Gold92].

4.3 Specification of Environment

The source of input events and destination of output events of a stand-alone component is its *environment*. The environment contains agents, or users, that cause input events to be sent to a component. These agents are autonomous; their behaviour cannot be defined or predicted. We can often make reasonable assumptions, but not guarantees, about the behaviour of these agents. For example, it may be considered to be a reasonable assumption that the users of a UNIX system will not delete their personal .login and .cshrc files.

Those events that may exist in the environment define the set Φ of possible events for the component. By exclusion, any event not defined or allowed must be treated as harmful. The environment may contain events that are acceptable or unacceptable to the component.

A component that is interconnected with another component will act as a source of inputs to that component. Instead of allowing the unpredictable environment agents to subsume the behaviour of the predictable, interconnected component, we introduce the *meta-environment* of the component. The meta-environment consists of all possible source of input to a component. The sources of inputs from the meta-environment are the predictable, interconnected components and the unpredictable agents of the component's stand-alone environment. The meta-environment of an individual, stand-alone component is equivalent to the environment of that component.

4.4 Specification of Components

A component is a predictable functional element that may exist as a stand-alone element or may be connected with other components in a hierarchical fashion. A stand-alone component is modelled with a single input source, the environment, and a single output destination, the environment. The sequence of inputs received and outputs produced over time characterises the behaviour of a component over time. The set of all possible input and output is defined by the set Φ.

A "normal" component may receive inputs and produce outputs, including null events, where $\{\alpha, \beta, \chi, \delta, \phi, \eta, \mathit{null}\} \subset \Phi$.

$$\alpha, \beta, \chi, \mathit{null}, \ldots \rightarrow \boxed{\text{Normal}} \rightarrow \delta, \mathit{null}, \phi, \eta, \ldots$$

A "sink" type component may receive inputs but will never produce an output. As a modelling abstraction of this type of component, a sink-type component will produce a constant stream of null outputs, where $\{\alpha, \beta, \chi, \mathit{null}\} \subset \Phi$.

$$\alpha, \beta, \chi \ldots \rightarrow \boxed{\text{Sink}} \rightarrow \mathit{null}, \mathit{null}, \mathit{null}, \ldots$$

A "source" type component will never receive any inputs but may produce outputs. As a modelling abstraction of this type of component, a source-type component will receive a constant stream of null outputs, where $\{\delta, \phi, \eta, \mathit{null}\} \subset \Phi$.

$$\mathit{null}, \mathit{null}, \mathit{null}, \ldots \rightarrow \boxed{\text{Source}} \rightarrow \delta, \phi, \eta, \ldots$$

4.4.1 Component Model

Formally, a component is specified as a model $\mathcal{M} = (\mathcal{S}, \sigma, \mathcal{V})$ where

$\mathcal{S}$ represents the possible combination of events (predicates), occurring in sequences of the system, $\mathcal{S} \subseteq \mathcal{F}ma(\Phi)$.

σ defines the possible sequences of input and output events in the system

$\mathcal{V}$ assigns a truth value to the predicates representing events in the system, thereby evaluating the sequence σ.

The set Φ represents the set of all possible events within the system. The formulae $A \in \mathcal{F}ma(\Phi)$ of events occurring at component are defined as

$A ::= p \mid \perp \mid A_1 \Rightarrow A_2$

This definition ensures that an event will not contain any temporal information about its existence in the system; the states of a sequence are static formulae.

Let us consider a component where the event α is received at time t=x. The event α is defined such that $\alpha \in \mathcal{S}$, according to the model, $\alpha \in \Phi$ according to the definition of possible events, so that $\mathcal{S} \subseteq \Phi$. The event α is true at time x, so that $s_x = \alpha$ where $s_x \in \sigma$ and $\sigma : j \in \mathbb{N} \rightarrow \alpha \in \mathcal{S}$.

4.4.2 Non-Determinism of Components

'Researchers in formal security models have tended to use the term "non-deterministic" to refer to machines whose operation contains some elements of randomness...it might be less misleading to use one of "random" or "stochastic".' (p. 164 [LBTS92]) As defined by Lee et. al., there are three "types" of non-determinism. Type 1 effects are the

> manifestations of true stochastic processes, as addressed in [Gray90]. These exist in everything we build, but in most applications a substantial part of the effort of engineering design is directed towards minimising their effects. Rarely, devices are designed so that their operation intentionally contains a significant random component. A random number generator built around a noise diode would be an example. Such devices are the exception rather than the rule. (p. 170, [LBTS92])

Type 2 non-determinism is a result of under-specification, due to unknown or unresolved details of the model. Type 3 non-determinism is a result of modelling abstractions, causing behaviour that is deterministic to appear as non-deterministic in the model.

In our work, components are modelled as predictable processes, where the outputs are governed by the laws of probability. The probability of any output other than those defined by the component behaviour is zero. Given the inputs to a component, we define its predictable behaviour. True non-deterministic behaviour, where the output is not predictable, is not handled within our model or our approach. Any non-determinism in our work is Type 1 non-determinism as defined by Lee et. al.

4.4.3 Component Behaviour Rules

The predictable behaviour of a component is defined by a set of *input-output behaviour rules* correlating the input and output events at the component. These define the events received and the corresponding possible outputs produced by the components. The actual relationship between input and output, including any dependencies on the state of a component, are defined by the *component behaviour axioms*, and discussed in a later section.

If a component is known to always produce the same output given a certain input, the input-output behaviour rules will define

$$(\alpha ; \beta)$$

If a component may produce one of several defined outputs in response to a given input, the association will define

$$(\alpha ; \beta \vee \chi)$$

If a component may produce several outputs in response to an input, the association will define:

$$(\alpha ; \beta \wedge \delta)$$

The input-output behaviour rule may define specific input and output events, or it may refer to *classes* of input and output events. Using classes of events will reduce the number of combinations for which we must define association rules.

4.4.4 Component Behaviour Axioms

The behaviour of a component reflects the inputs and outputs to the component. This is represented by the *component behaviour axioms*. The structure of a component behaviour axiom is given as:

$$\sigma\{\mathbf{i}\} \models_x \text{(state of component)} \wedge \sigma\{\mathbf{i}\}\text{(in)} \models_x \text{(input)}$$

$$\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond\text{(new state of component)} \wedge \sigma\{\mathbf{i}\}\text{(out)} \models_x \Diamond\text{(output)}$$

$$= \sigma\{\mathbf{i}\} \models_x \text{(state of component)} \wedge \sigma(\text{Ɇ};i) \models_x \text{(input)}$$

$$\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond\text{(new state of component)} \wedge \sigma(i;\text{Ɇ}) \models_x \Diamond\text{(output)}$$

Behaviour axioms may include the state of a component for situations where different states allow different inputs or require different outputs or destinations. The component behaviour axioms define the state of the component required for a given input to be processed by the component. If the behaviour axiom is defined such that the component is not required to be in any particular state, then the component will always be able to process the given input. Such an input-state dependent behaviour axiom is given as:

$$\sigma\{\mathbf{i}\}\text{(in)} \models_x \text{(input)}$$

$$\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond\text{(new state of component)} \wedge \sigma\{\mathbf{i}\}\text{(out)} \models_x \Diamond\text{(output)}$$

If the processing of an event does not affect the state of a component, then the output-state independent behaviour axiom is given as:

$$\sigma\{\mathbf{i}\} \models_x \text{(state of component)} \wedge \sigma\{\mathbf{i}\}\text{(in)} \models_x \text{(input)}$$

$$\Rightarrow \sigma\{\mathbf{i}\}\text{(out)} \models_x \Diamond\text{(output)}$$

4.5 Specification of System

A system is an interconnection of one or more components, referred to as a *composite* or *interconnected* system. A composite system is represented by inputs and outputs to the system components and the interconnections between the system components. A *simple system* is the representation of a composite system as a stand-alone component using only the input-output behaviour rules of the overall system. Such a representation does not include the internal details of the composite system.

4.5.1 Interconnection of Components

The interconnection of components is defined by a set of *interconnection rules*. These rules fully describe all possible input and output channels for each component in the system. There is (at least) one interconnection rule for each component and one for the environment. The rule defines all sources of input events (the outputs from all components and the environment) received at a component (or the environment).

The general interconnection rule for a system with **N** components, 0,1,..., n-1, is defined as follows for component **i**:

$$(\text{environs}, \text{out}^{0}, \text{out}^{1}, \ldots, \text{out}^{n-1} \rightarrow \text{in}^{i})$$

where the term *environs* represents the agents in the environment and out^i and in^i represent the outputs and inputs to component **i**.

The interconnection rule for the environment defines all components that have an output channel to the environment. The general interconnection rule for a system with **n** components, where each component sends an output to the environment is defined as:

$$(\text{out}^{0}, \text{out}^{1}, \ldots, \text{out}^{n-1} \rightarrow \text{environs})$$

A component may have multiple interconnections with the same component. Consider a component, **i**, that has two interconnections with component **j**, on channels 0 and ℓ. This is represented in the interconnection rules as follows:

$$(\text{out}^{i}[0], \text{out}^{i}[\ell] \rightarrow \text{in}^{j})$$

Interconnection rules can also be defined by all possible destinations of the output of a component:

$$(\text{out}^{i} \rightarrow \text{in}^{0}, \ldots, \text{in}^{n}, \text{environs})$$

We will use the first approach to define all the connections within a system. We may choose to use the latter form of interconnection rule, as derived from the set of interconnection rules of a system, when we are examining the outputs of a given component.

4.6 Composite System Structures

We consider three types of composite systems: *product*, *cascade*, and *feedback*. From these "basic" composite systems we can create more general, complex composite systems. 'Product and feedback are sufficient for performing general composition.' [McLe94, LiVi90] Because not all systems will contain feedback, we also examine the cascade composition construct. The three types are differentiated by the connections that are possible for each type. In this work we assume that all physical connections defined by interconnection rules are static. We will not deal with dynamic connections such as library links in this work[4].

The product composition of components, shown in Figure 4.2, is simply two or more components that continue to interact solely with the environment but are considered as a single system. The interconnection rules of a product composition of components have the following form for a system with $\mathbf{N}$ components:

(environs $\rightarrow$ in^0, in^1, ... , in^{n-1})

(out^0, out^1, ... , out^{n-1} $\rightarrow$ environs)

Components in such a system do not interact with each other.

The cascade composition of components, shown in Figure 4.3, is the one-way connection of two or more components. A component may not send an input to a component from which it receives an input. These components may continue to interact with the environment. The interconnection rules of a general cascade composition of components have the following form for a system with $\mathbf{N}$ components:

(environs $\rightarrow$ in^0, in^1, ... , in^{n-1})

(out^0 $\rightarrow$ in^1, ... , in^{n-1}, environs)

(out^i $\rightarrow$ in^{i+1}, ... , in^{n-1}, environs)

(out^{n-1} $\rightarrow$ environs)

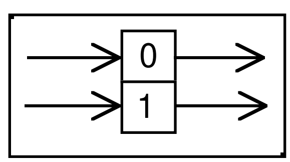

Figure 4.2: General Product Composition of Two Components

A general cascade structure contains all allowable cascade connections. A cascade structure need not contain all of these interconnections. If the interconnection between

---

[4] Dynamic links can be modelled by considering all possible dynamic links as existing static links. Thus all components will always be connected to a system. This will have the drawback of assessing as insecure behaviours that may be otherwise secure if the dynamic link were not in place (so that the component was not part of the system). This will result in a more restrictive assessment of security than if dynamic links were allowed.

components 0 and 1 is removed, the composite structure can be shown to be equivalent to a product structure.

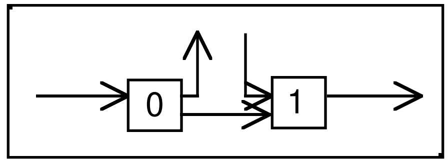

Figure 4.3: General Cascade Composition of Two Components

Components connected in a feedback composite system may send and receive events from any components in the system. The interconnection rules of a general feedback composition of components, shown in Figure 4.4, have the following form for a system with **n** components:

(environs $\rightarrow$ in^0, in^1, ... , in^{n-1})

(out^0 $\rightarrow$ in^0, in^1, ... , in^{n-1}, environs)

(out^i $\rightarrow$ in^0, in^1, ... , in^{i+1}, ... , in^{n-1}, environs)

(out^{n-1} $\rightarrow$ in^0, in^1, ... , in^{i+1}, ... , in^{n-1}, environs)

Two-way communication between components is possible in a system with feedback.

A general feedback structure contains all allowable feedback connections. A feedback structure need not contain all of these interconnections. If one of the interconnections between components 0 and 1 is removed, the structure can be shown to be equivalent to a cascade structure.

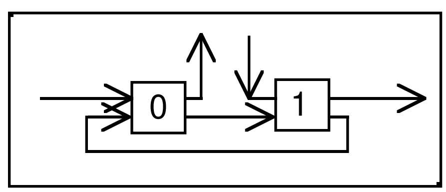

Figure 4.4: General Feedback Composition of Two Components

Feedback, the connection of a given component's output to the input of a component, from which the given component (directly or indirectly) receives an input, is defined by interconnection rules such as:

(out^i $\rightarrow$ ... , in^j, ... ,)

(out^j $\rightarrow$ ... , in^i, ... ,)

A special case of a feedback composition is the feedback composition of one component. Self-feedback, the connection of a component's output to its own input, is defined by interconnection rules such as:

(out^i $\rightarrow$, ... , in^i, ...)

A general composite system with a feedback connection may in some situations be simplified into a single (complex) system with a reflexive connection. If a system has multiple feedback connections between different components, this simplification may not be possible. Therefore we must consider both self-feedback and feedback connections.

We represent the composition of two components C_i and C_j as $\&(C_i, C_j)$. Corresponding to the three types of composition, we define

$\&^P(C_i, C_j)$: Product Composition of Components C_i and C_j

$\&^C(C_i, C_j)$: Cascade Composition of Components C_i and C_j

$\&^F(C_i, C_j)$: Feedback Composition of Components C_i and C_j

$\&^R(C_i)$: Feedback Composition of Component C_i

The composite structure $\&^P$ is order-independent: $\&^P(C_i, C_j) = \&^P(C_j, C_i)$ because the components do not interact with each other. The composite structure $\&^C$ is order-dependent, as shown in Figure 4.4. This notation represents the asymmetric composition of C_i and C_j given the interconnection rule ($out^i \rightarrow in^j$). The composite structure $\&^F$ represents the general feedback composition where every component is connected to every other component. Such a composition is order-independent, so that $\&^F(C_i, C_j) = \&^F(C_j, C_i)$.

If a cascade or feedback system does not exhibit all possible, defined interconnections to and from the environment, then it is not a general cascade or general feedback structure. A non-general cascade structure is represented as $\&^{\sim C}$ and a non-general feedback structure is represented as $\&^{\sim F}$. The connections between components for general and non-general structures must be implemented; these are the connections that define the structure of the system as product, cascade or feedback.

The composite system $\&(C_i,C_j)$ may be hierarchically composed with other components or systems. The composition of the system $\&(C_i,C_j)$ with the component C_k is represented as $\&(\&(C_i,C_j),C_k)$. The composition of the system $\&(C_i,C_j)$ with the system $\&(C_j,C_k)$ is represented as $\&(\&(C_i,C_j),\&(C_j,C_k))$. Thus hierarchical composition will be used to create large, complex systems from less complex individual systems and components.

4.7 Composition of Environments

The process of interconnecting components may cause a component's meta-environment to change, to include any newly connected input sources and events. The meta-environment must be re-defined to include these new input sources. A meta-environment that is safe or live for a component may become a meta-environment that is not safe or live for that component when interconnected with additional components.

This process may also change the possible input events received from each component's environment. This occurs because the unpredictable agents of one component's environment may now behave as unpredictable agents in the other component's environment. That is, there may be "new" events that can be received from the unpredictable agents as a result of the interconnection of components.

The possible input events received at a component may change because of the possible input events that may be received from interconnected components. There may be output events of interconnected components that may be received as input events, where these events were not previously possible input events.

The set of possible input events in a composite meta-environment is defined as the union of the sets of possible input events of the individual components. For example, the interconnection of two components **i** and **j**, with the input events of the individual components defined by the sets of events Φ^i and Φ^j, is given as $\Phi^i \cup \Phi^j$.

The same event may have a different name within different environments. We must ensure the consistency of names for events exchanged between systems. We will therefore *rename* events so that any such communication event has the same name in the composite environment.

In the previous definition of composition of environments [Hin93], systems could be said to be safe only if the composite environment was a subset of each of the individual environments. This implies that the composite environment continues to contain only events acceptable to all components. We define a less restricted approach, allowing for the possibility of unacceptable events in the composite environment and in the component's meta-environment.

4.8 Composition of System Models

The formal model of a composite system is a combination of the models of the individual components and represents the behaviour of these components within the composite system. Given the individual models of the components we can define the model of the composite system as follows.

Definition: Composite System Model

Given two individual systems, **i** and **j**, represented as $\mathcal{M}^i = (\mathcal{S}^i, \sigma, \mathcal{V}^i)$ and $\mathcal{M}^j = (\mathcal{S}^j, \sigma, \mathcal{V}^j)$ where

$\mathcal{S}^i = \mathcal{F}ma(\Phi^i)$ and

$\sigma^i : \omega \rightarrow \mathcal{S}^i$ enumerating $\mathcal{S}^i$ as a sequence $s^i_0, s^i_1, s^i_2, \dots s^i_n,$

$\mathcal{V}^i : \mathcal{F}ma(\Phi^i) \rightarrow \{\text{true, false}\}$

$\mathcal{S}^j = \mathcal{F}ma(\Phi^j)$ and

$\sigma^j : \omega \rightarrow \mathcal{S}^j$ enumerating $\mathcal{S}^j$ as a sequence $s^j_0, s^j_1, s^j_2, \dots s^j_n$

$\mathcal{V}^j : \mathcal{F}ma(\Phi^j) \rightarrow \{\text{true, false}\}$

The composite system model of the composition of components **i** and **j** is:

$\mathcal{M}^{i+j} = (\mathcal{S}^{i+j}, \sigma, \mathcal{V}^{i+j})$ where
$\mathcal{S}^{i+j} = \mathcal{F}ma(\Phi^{i}) \cup \mathcal{F}ma(\Phi^{j})$ and
$\mathcal{V}^{i+j}: \mathcal{F}ma(\Phi^{i}) \cup \mathcal{F}ma(\Phi^{j}) \rightarrow \{\text{true, false}\}$
$\sigma^{i+j} : \omega \rightarrow \mathcal{S}^{i+j}$ enumerating $\mathcal{S}^{i+j}$ as a sequence

$$\sigma^{i} \wedge \sigma^{j} = s^{i}_{0} \wedge s^{j}_{0}, s^{i}_{1} \wedge s^{j}_{1}, \ldots, s^{i}_{n} \wedge s^{j}_{n} \ldots$$

The composite set $\mathcal{S}^{i+j}$ represents the set of numerable events for the composite system. Thus composition of systems may result in new events, including the possibility that the possible events of system **i** in the composite system may be a superset of the possible events of the individual system **i**.

The composite sequence σ^{i+j} defines all events that occur at both components at any given time in the sequence. Thus the sequence contains all possible subsequences in the composite system, including sequences to and from the environment.

For modelling purposes we assume that a channel exists for each possible interconnection that a component may exhibit. Channels that are not connected are modelled as containing only null events, evaluated by the function $\mathcal{V}^{i+j}$.

In practice, we require that any events occurring on any unconnected channels, will be sent to the environment and that the effects of such events be evaluated. When this component is interconnected with other components, events on the channels will be sent to the appropriate component destinations instead of bypassing to the environment. We cannot ignore these channels when unconnected because they still have the potential, if used (incorrectly) by hostile or malicious sources, to violate the properties of the system. This is part of a risk analysis of a system and will not be addressed in this work.

4.9 Composite System Behaviour

The system behaviour axioms reflect the behaviour of the system components given the interconnections specified by the interconnection rules and the component behaviour axioms. The most general form of the behaviour axioms defining the behaviour of a component **i** within a composite system is given as:

$\sigma\{\mathbf{i}\} \models_{x}$ (state of component) $\wedge\ \sigma\{\mathbf{i}\}(\text{in}) \models_{x}$ (input)
$\Rightarrow \sigma\{\mathbf{i}\} \models_{x} \Diamond$(state of component) $\wedge\ \sigma\{\mathbf{i}\}(\text{out}) \models_{x} \Diamond$(output)
$\equiv \sigma\{\mathbf{i}\} \models_{x}$ (state of component) $\wedge\ \sigma(\mathcal{E}^{+}0^{+} \ldots ; \text{i}) \models_{x}$ (input)
$\Rightarrow \sigma\{\mathbf{i}\} \models_{x} \Diamond$(state of component) $\wedge\ \sigma(\text{i};\ \mathcal{E}^{+}0^{+} \ldots) \models_{x} \Diamond$(output)

Component and system behaviour axioms can be classified as input-state independent, output-state independent, and state-dependent, as follows, where A, B, X, Δ are formulae defining the events occurring at time x in each subsequence:

Input-State Independent Behaviour Axiom

$$\sigma\{\mathbf{i}\}(\text{in}) \models_x A \ \Rightarrow \sigma\{\mathbf{i}\}(\text{out}) \models_x \Diamond X \wedge \sigma\{\mathbf{i}\} \models_x \bigcirc\Delta$$

Output-State Independent Behaviour Axiom

$$\sigma\{\mathbf{i}\}(\text{in}) \models_x A \wedge \sigma\{\mathbf{i}\} \models_x B \Rightarrow \sigma\{\mathbf{i}\}(\text{out}) \models_x \Diamond X \wedge \sigma\{\mathbf{i}\} \models_x \bigcirc B$$

State-Dependent Behaviour Axiom

$$\sigma\{\mathbf{i}\}(\text{in}) \models_x A \ \wedge \sigma\{\mathbf{i}\} \models_x B \Rightarrow \sigma\{\mathbf{i}\}(\text{out}) \models_x \Diamond X \wedge \sigma\{\mathbf{i}\} \models_x \bigcirc\Delta$$

It may be the case that a given input may be received only from one possible source (or a subset of sources). In this case, we may wish to define the system behaviour axioms over the subset of possible input sources, such as

$$\sigma\{\mathbf{i}\} \models_x (\text{state of component}) \wedge \sigma(0^+1;\, i) \models_x (\text{input})$$
$$\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond(\text{new state of component}) \wedge \sigma(i;\, \text{Ɇ}) \models_x \Diamond(\text{output})$$

Given the interconnection rules of the system, we can determine the source and destination of the composite system behaviour axioms. If the interconnections of two components, **i** and **j**, are completely defined by the interconnection rules (environs $\rightarrow$ in^i), ($out^i \rightarrow in^j$), ($out^i \rightarrow$ environs), (environs $\rightarrow in^j$), ($out^j \rightarrow$ environs) then the composite behaviour axioms are given, in their general form, as:

$$\sigma\{\mathbf{i}\} \models_x (\text{state of component}) \wedge \sigma(\text{Ɇ};i) \models_x (\text{input})$$
$$\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond(\text{new state of component}) \wedge \sigma(i;j^+\text{Ɇ}) \models_x \Diamond(\text{output})$$

and

$$\sigma\{\mathbf{j}\} \models_x (\text{state of component}) \wedge \sigma(i^+\text{Ɇ};j) \models_x (\text{input})$$
$$\Rightarrow \sigma\{\mathbf{j}\} \models_x \Diamond(\text{new state of component}) \wedge \sigma(j;\text{Ɇ}) \models_x \Diamond(\text{output})$$

For example, the following behaviour axiom

$$\sigma(\text{Ɇ};i) \models_x A \Rightarrow \sigma(i;j^+\text{Ɇ}) \models_x \Diamond B$$
$$= \sigma(\text{Ɇ};i) \models_x A \text{ and for some } y \geq x,\ \sigma(i;j^+\text{Ɇ}) \models_y B$$

is defined over the subsequences $\sigma(\text{Ɇ};i)$ and $\sigma(i;j^+\text{Ɇ})$ where $A \in \Phi(\text{Ɇ}(\text{out}), i(\text{in}))$, and $B \in \Phi(i(\text{out}), j(\text{in}), \text{Ɇ}(\text{in}))$.

The interconnection rules define the channels between components in the system and therefore the *possible* destinations of events. The system behaviour axioms include the actual destinations of events produced by the components in the system. These formally defined behaviour axioms are used to describe the behaviour of a system. They may define the behaviour of the entire system or of parts of the system. These unambiguous descriptions of the behaviour are then analysed given the desired (formally defined) goals of the behaviour.

5. Property Specification

Simply because they change and hide their names,
do not give their right age, and by their own admission
go about without allowing themselves to be recognised,
there is no logic that can deny that they necessarily must exist.
HIENRICH NEUHAUS: Pia et ultimissima admonestatio de Fratribus Rosae-Crucis,
nimirum: an sint? quales sint? unde nomen illud sibi asciverunt
(French ed. 1623)

In this chapter we examine the properties used to define the desired behaviour of systems, as compared with the actual (defined) behaviour introduced in the previous chapter. A property is a formal instantiation of the desired behaviour of a system, representing a goal that must be satisfied by all possible behaviours. The actual behaviour of the system may or may not satisfy a given property. In this chapter we examine the definition and specification of the properties (desired behaviours) of a system.

We examine how the goals of a property affect its formal definition and how to classify properties according to this definition. The classification and meta-properties of a property (the properties of a property) help determine if the desired behaviour of a system is retained when the system is composed with other systems.

5.1 Specification of Policies

Security is defined by a set of security policies, designed to completely specify the goals of the system and to specify those behaviours (sequences) of a system that violate these goals. Policies are goals or objectives of a system's behaviour, written informally, as opposed to the formal notation used to describe properties. Policies are often written with respect to an *open system model* where users are outside the system model. They make little or no reference to the environment in which a system exists. We believe that a policy must be expressed formally as a property on a *closed system model*, where a closed model includes both the system and the environment.

Some policies that make specifications about the environment are too restrictive for practical use. For example, consider the restrictive policy "The system will always have a constant supply of power." We must ensure that the environment realistically reflects the environment in which a system will operate. Thus a less restrictive policy may be given as "The system will have a constant power supply and will fail gracefully if the power supply fails." This less restrictive property includes the "perfect world" desired behaviour and the known exceptions to that perfect world. We must also examine the effects of any changes to the environment on both the satisfiability of the security properties and on the assumptions that have been made about the behaviour of the environment.

5.2 Specification of Properties

Properties are the formal specification of policies defining the desired behaviour of a system. Properties differ from behaviour axioms; behaviour axioms define the actual behaviour of a system, whereas properties define the desired behaviour of a system.

We define a property to be satisfied by the behaviour of a system as follows: the behaviour of a system, as defined by the system behaviour axioms, implies those behaviours specified by a property. If there is at least one possible sequence of events that does not imply a given property, then the property is not satisfied for the given system.

A property, P_k, consists of a domain and a body, $P_k = Q_k \cdot S_k$. The domain, Q_k, is a quantification of the variables of interest. The property domain includes the definition of the environment over which the property will be satisfied. The property domain in its general form is

$$Q_k \equiv \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A : \mathcal{F}ma\{\Phi\}$$

where λ is a quantifier, $\lambda = \{\forall, \exists\}$, $\mathbb{N}$ is the natural numbers, $\mathcal{N}$ is the set of components in the system, and $\mathcal{F}ma\{\Phi\}$ are the formulae defining the combinations of atomic events that may occur in a sequence. For example,

$$Q_k = \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A,B : \mathcal{F}ma\{\Phi\}$$

states that for all positions (in the sequence σ) and for all components in the system, there are two combinations of atomic events, A and B.

Within the property domain we define the *event domain*, the set of possible events, $\lambda A : \mathcal{F}ma\{\Phi\}$. The event domain is sometimes referred to as the environment, when we wish to indicate the possible actions of the environment. A property may be defined over all or some events in the set $\mathcal{F}ma\{\Phi\}$ or over all or some events in a subset of $\mathcal{F}ma\{\Phi\}$, such as $\mathcal{F}ma\{\Phi(A,obs)\}$.

The body, S_k, is the specification a Boolean expression whose variables represent quantities of interest. This *property specification* defines the formal requirements of the goals of the behaviour of the system.

In its general form, the property specification will include a boolean expression on the events (the world, W_k over which the specification is defined) and the relationship between events (the required behaviour, R_k).

$$S_k \equiv \lambda A \in \mathcal{F}ma\{W_k\} \wedge R_k$$

where λ is a quantifier, $\lambda = \{\exists, \forall\}$. The world W_k is a subset of the set of possible events, $W_k \subseteq \Phi$ and defines the set of *allowable* events. It is possible that a property be defined with a domain that includes unwanted events. The world W_k defines those events in the environment that will not violate the property. By exclusion, therefore, those events that are defined in Q_k but not in W_k, if received, will violate the requirements of the property.

The is no general form of R_k for all properties. The basic properties defined in this work have two general forms of required behaviour. The safety properties have static formulae defining the required behaviour that must be satisfied over all time. These formulae are of the form $(\sigma \models_x \alpha \Rightarrow \sigma \models_x \beta)$. The progress properties define the temporal relationship between behaviours over (potentially all) time. These formulae are of the form $(\sigma \models_x \alpha \Rightarrow \sigma \models_x \Diamond \chi)$.

Thus a property applied to component **j** might be given as:

$$\begin{aligned} P \equiv\ & \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall A : \mathcal{F}ma\{\Phi(i,\text{constant\_power\_supply})\} \cdot \\ & \quad \sigma\{\mathbf{i}\}(in) \models_x A \vee (\sigma\{\mathbf{i}\}(in) \models_x \neg A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \mathit{null}) \\ =\ & \circ\ (\forall i : \mathbb{N} \cdot \forall A : \mathcal{F}ma\{\Phi(j,\text{constant\_power\_supply})\} \cdot \\ & \quad \sigma\{\mathbf{j}\}(in) \models_x A \vee (\sigma\{\mathbf{j}\}(in) \models_x \neg A \Rightarrow \sigma\{\mathbf{j}\}(out) \models_x \mathit{null})) \end{aligned}$$

This states that all components must receive as an input a constant power supply or if they do not receive a constant power supply they will produce only a null output.

The set of properties of a component C_i is represented as $\mathbf{P}(C_i)$. If P_k is a property of C_i then $P_k \in \mathbf{P}(C_i)$.

5.3 Domain-Dependent and Independent Properties

Properties, as instantiated for an individual component or system, are defined with respect to the environment of that system. If either the environment or the system changes then a given property of the system may not longer be satisfied. In addition, some properties may or may not be satisfied according to the behaviour of an environment even if the possible events of the environment and the system have not changed.

A *domain-dependent* property makes assumptions about the behaviour of the meta-environment, that is, the (unpredictable) agents in the environment and the (predictable) components of the system. A domain-dependent property is one that is defined over a subset of input events within Q_k or W_k. If $Q_k = \forall x{:}\mathbb{N} \cdot \lambda_1 i{:}\mathbb{N} \cdot \lambda_2 A{:}\mathcal{F}ma\{\Phi\}$ where $\lambda_2 = \exists$ then P_k is a domain-dependent property. If $S_k = \lambda A \in \mathcal{F}ma\{W_k\} \wedge R_k$, where $\lambda = \exists$ or $W_k \neq \Phi$ then P_k is a domain-dependent property.

Such a property states that not all possible inputs are created equal. Domain-dependent properties place additional restrictions on the inputs received. If not all inputs are equally possible (e.g., $\exists A{:}\mathcal{F}ma\{\Phi(in)\}$) then the property will be violated if an event other than A is received as an input. If the set of all events that are equally possible is a subset of the set of all defined events (e.g., $\forall A \in \mathcal{F}ma\{\Phi(i(in))\}$) then the property will be violated if an event other than one of these allowable events is received as an input. Thus these properties rely on the input events sent to a component from other components and from the environment.

Properties that are defined over all possible input events ($\forall A : \mathcal{F}ma\{\Phi\}$) are defined such that all possible input events are allowable events ($W_k = \Phi$). Such a property is *domain-independent*: it does not place additional requirements on the possible input events of the system. A simple example of a domain-independent policy is "*all input events received at a component will eventually cause a corresponding output*."

A simple example of a domain-dependent policy is "*the event rm *.* is a possible event but it is not allowable from the directory /root*." A domain-dependent property depends on the behaviour of the meta-environment and is therefore ultimately as dependent as the meta-environment. Many safety properties examined in previous work are domain-dependent properties that have been treated as domain-independent properties. This often results in seemingly inexplicable and unexpected violations of properties when the actual environment differs from the environment that has been assumed for a system.

5.4 Evaluation of Properties

Properties are evaluated (or assessed) to determine if they are satisfied by a system's behaviours. To evaluate a property we examine the property domain with respect to the environment of the system and the property specification with respect to the behaviours of the system, for all relevant components of the system.

5.4.1 Evaluation of Property Domain

The property domain must correspond to the environment of the system. The property must be specified over the relevant components in the system ($\forall i{:}\mathbb{N}$) and over the possible events (Φ). This should follow given the correct definition and application of the property for the system in question.

If the events defined by the event domain (Φ) correspond to the set of allowable events (W_k), then all possible events are allowable events. This means that no possible event will violate the property. Thus we do not need to consider the effects of individual events at the system.

Events in the event domain that do not belong to the set of allowable events are events such as α where $\alpha \in \Phi$ and $\alpha \notin W_k$. No such event α may occur in any of the sequences of the system without violating the property. We must assess the behaviour of the system to ensure that such events will not occur as the result of the actions of the meta-environment.

We choose to allow for the possibility of $W_k \subset Q_k$ because we believe that in many instances it is impractical to define or implement a system with $W_k = Q_k$. If we identify an event α that is harmful to a component and yet is a possible event, we may use a mechanism, such as a filter, to remove this event whenever it occurs as an input. The presence of events such as α may be the result of actions that are too difficult to protect against, have a very low threat to the system, have a very low risk of occurring, or for which the system has a low vulnerability. We allow for events such as α because we believe that it may be impractical to define a mechanism(s) to remove all potentially harmful events.

We must evaluate the risk associated with every potentially harmful (atomic) event α, where $\alpha \in Q_k \backslash W_k$. The set restriction operator, \, is used to represent the set A\B, the set of all elements that are in A but are not in B. The evaluation of $Q_k \backslash W_k$ is given in the Figure 5.1.

$\forall \alpha \in Q_k \backslash W_k$
  if {there is absolute trust that agent will not send α
    or there is little threat to the system if α is received
    or the risk associated with α is minimal
    or the vulnerability of the system to α is low}
    then
    judge as acceptable risk of allowing α
  else
    determine effect to system if event received
      choose to leave system as is,
      given knowledge of threat, risk, vulnerability
    or
      choose to modify system to ensure event is no
        longer possible
      or
      choose to implement mechanism to remove α

Figure 5.1: Evaluation of Allowable Events in an Environment

5.4.2 Evaluation of Property Specification

We must also ensure that the possible behaviours of the system satisfy the required property. This is represented as

$$\mathbf{sat}(C_i, P_k) : C_i \times P_k \rightarrow \{\top, \bot\}$$

where $\mathbf{sat}(C_i, P_k) = \top$ if and only if the behaviours of component C_i imply the property P_k where P_k is defined to account for the connections of the component.

$$\mathbf{sat}(C_i, P_k) = \top \Leftrightarrow C_i \Rightarrow P_k$$

where for all behaviours of the component

$$\mathbf{sat}(C_i, P_k) = \top \Leftrightarrow \forall B_i : C_i \cdot B_i \Rightarrow P_k$$

If a behaviour axiom defines a behaviour that is irrelevant to the goal of P_k we need not consider that axiom with respect to P_k. Given the behaviour axiom that is relevant to the goal of P_k, we use the axioms and schemata of the temporal logic and attempt to prove that $B_i \Rightarrow P_k$ for $\forall B_i \in C_i$.

For example, if a component C_j has the connections $(out_i \rightarrow in_j)$ and $(out_j \rightarrow environs)$ and P_k is given as

$$P_k = \forall x : \mathbb{N} \cdot \lambda i : \mathbb{N} \cdot \forall A,B : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(in) \models_x B)$$

then P_k as applied to component C_j is defined as

$$P_k = \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{j}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{j}\}(in) \models_x B)$$

Given the behaviour B_i

$$B_i = \forall x : \mathbb{N} \cdot \forall A : \mathcal{Fma}\{\Phi(in)\} \cdot \forall B,X : \mathcal{Fma}\{\Phi(out)\} \cdot$$
$$(\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B \vee X)$$

then

$$\mathbf{sat}(B_i, P_k) \Leftrightarrow B_i \Rightarrow P_k$$
$$= \forall x : \mathbb{N} \cdot \forall A : \mathcal{Fma}\{\Phi(in)\} \cdot \forall B,X : \mathcal{Fma}\{\Phi(out)\} \cdot$$
$$(\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B \vee X)$$
$$\Rightarrow \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B)$$

Let $\Delta = B \vee X$. By the definition of $\mathcal{Fma}\{\Phi\}$, if $\forall B,X : \mathcal{Fma}\{\Phi\}$ then $\forall \Delta : \mathcal{Fma}\{\Phi\}$.

$$= \forall x : \mathbb{N} \cdot \forall A : \mathcal{Fma}\{\Phi(in)\} \cdot \forall \Delta : \mathcal{Fma}\{\Phi(out)\} \cdot$$
$$(\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Delta)$$
$$\Rightarrow \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B)$$

Also by definition, if $A : \mathcal{Fma}\{\Phi(in)\}$ then $A : \mathcal{Fma}\{\Phi\}$

$$= \forall x : \mathbb{N} \cdot \forall A,\Delta : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Delta)$$
$$\Rightarrow \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{Fma}\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B)$$
$$= \top$$

Therefore $\mathbf{sat}(B_i, P_k) = \top$; the behaviour of B_i satisfies the goal of property P_k.

5.4.3 Enforcement of Properties

Given the evaluation of a system, a property will either be inherently satisfied by the behaviour of the system or it will not be satisfied. A property is inherently satisfied if every possible behaviour of a system, defined by the behaviour axioms, satisfies the property in question.

If a given property is not satisfied by the behaviour of a component, we may choose to *apply* the property to the system by configuring the system so that all behaviours satisfy the property. Given the behaviours for which a property may fail, we may, if possible, choose to modify the system so that those behaviours are no longer possible. For example, if the desired property does not hold for a system because of the behaviour of one individual component of the system, we may choose to remove that component from the system.

It is possible that no alteration to a system will result in a system that will exhibit a certain property. Such a property is not satisfiable; such a system must be redesigned if this property is to be satisfied.

If a system's behaviours do not inherently satisfy a property, and we are not able to modify a system so that all behaviours will satisfy the property, we may choose to use *mechanisms* to enforce the property. A mechanism is a filter that disallows any behaviours that would violate a property. Thus harmful events and sequences will be possible, but will be removed from the system by the mechanism enforcing a given property. Mechanisms may be implemented in software or hardware. An example of a mechanism is the access-control lists used to verify discretionary access control as described as part of the *Trusted Computer Systems Evaluation Criteria* [DACTS].

5.5 Properties of Composite Systems

In previous work the problem of evaluating properties for composite systems has been referred to as the composition of properties. In this work the composition of systems is the process of creating larger, complex systems from individual components. The problem of the composability of properties is really the problem of evaluating the satisfiability of a property for a composite system.

The satisfiability of a property P_k for the system composed of components C_i and C_j is represented as **sat**(**&**(C_i, C_i), P_k). For large composite systems (those with multiple components), we may choose to represent the composite system **&**(C_i, C_i) as $\mathfrak{C}$, so that **sat**($\mathfrak{C}$, P_k).

For a property P_k applied to a composite system we must examine the property for the system as a whole. The property will be satisfied by the composite system if it is satisfied by the system considered as a stand-alone component (as opposed to a collection of components).

The property P_k applied to all components in a system is given as

$$P_k = \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \lambda A : \mathcal{F}ma\{\Phi\} \cdot S_k$$
$$= \forall x : \mathbb{N} \cdot \lambda A : \mathcal{F}ma\{\Phi\} \cdot S_{k0} \wedge S_{k1} \wedge \ldots \wedge S_{k\mathbb{N}}$$

The property specification S_{ki} defines the instantiation of the property at a the component **i**, given the connections (input and output) of the component.

For example, consider the property P_k defined as

$$P_k = \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall A : \mathcal{F}ma\{\Phi\} \cdot (\sigma\{\mathbf{i}\}(in) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x B)$$

If this property is applied to a system with components C_1 and C_2 where the connections of the composite system are defined by (environs $\rightarrow$ in_1, in_2), ($out_1 \rightarrow in_2$), and ($out_2 \rightarrow$ environs) then P_k is defined for the composite system as:

$$P_k = \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{F}ma\{\Phi\} \cdot \sigma\{\mathbb{C}\}(in) \models_x A \Rightarrow \sigma\{\mathbb{C}\}(out) \models_x B$$
$$= \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{F}ma\{\Phi\} \cdot \sigma(\mathbb{E};\mathbb{C})(in) \models_x A \Rightarrow \sigma(\mathbb{C};\mathbb{E})(out) \models_x B$$
$$= \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{F}ma\{\Phi\} \cdot \sigma(\mathbb{E};1{+}2)(in) \models_x A \Rightarrow \sigma(2;\mathbb{E})(out) \models_x B$$
$$= \forall x : \mathbb{N} \cdot \forall A,B : \mathcal{F}ma\{\Phi\} \cdot (\ \sigma(\mathbb{E};1)(in) \models_x A \ \Rightarrow \sigma(2;\mathbb{E})(out) \models_x B\ )$$
$$\vee (\ \sigma(\mathbb{E};2)(in) \models_x A \Rightarrow \sigma(2;\mathbb{E})(out) \models_x B\ )$$

5.6 Meta-Properties of Properties

The reflexivity, transitivity, symmetry and closure of properties are *meta-properties* of properties. Transitivity and closure have been shown to be important in previous work [DBY94]. We introduce for consideration reflexivity and symmetry. These two meta-properties are used in the evaluation of the satisfiability of a property for a system with feedback. We examine all meta-properties for different composite structures.

Meta-properties of properties are important because in many cases it will be simpler to determine the meta-properties of a property than to re-evaluate the property with respect to a large, complex, composite system. Meta-properties are dependent on the property and the type of system composition. For example, a property that exhibits transitivity for a product composition of two components may not exhibit transitivity for a cascade composition of the same two components.

A set Σ is *closed* under a binary operation $\bullet$ if

$$\forall A,B \in \Sigma \cdot A \bullet B \in \Sigma$$

This is closure in the set theoretic sense. Closure is a function of composition used to compose models (systems). If the set Σ represents the set of components satisfying property P_k, then the set is closed under composition if the composition of any two components is equivalent to a component (in Σ) satisfying P_k. Closure of a property under the composition of C_i and C_j is defined as

$$\mathbf{sat}(C_i, P_k) \ \wedge\ \mathbf{sat}(C_j, P_k) \Rightarrow \mathbf{sat}(\mathbf{\&}(C_i, C_j), P_k)$$

This indicates that $(C_i \Rightarrow P_k) \wedge (C_j \Rightarrow P_k) \Rightarrow (\&(C_i, C_j) \Rightarrow P_k)$.

A property P_k that is reflexive on composition is satisfied by the behaviour of the component C_i and it is satisfied by the behaviour of the reflexive composition of C_i, $\&^R(C_i)$. A property P_k is reflexive on composition if and only if:

$$\mathbf{sat}(C_i, P_k) \Rightarrow \mathbf{sat}(\&^R(C_i), P_k)$$

The reflexive composition of a component is shown in Figure 5.2, where the reflexive connection is made from the output of the component to its input.

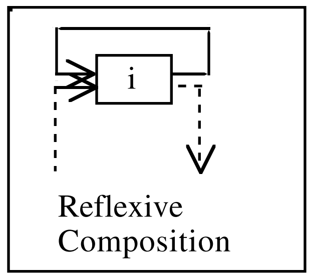

Figure 5.2: Reflexive Composite Structure

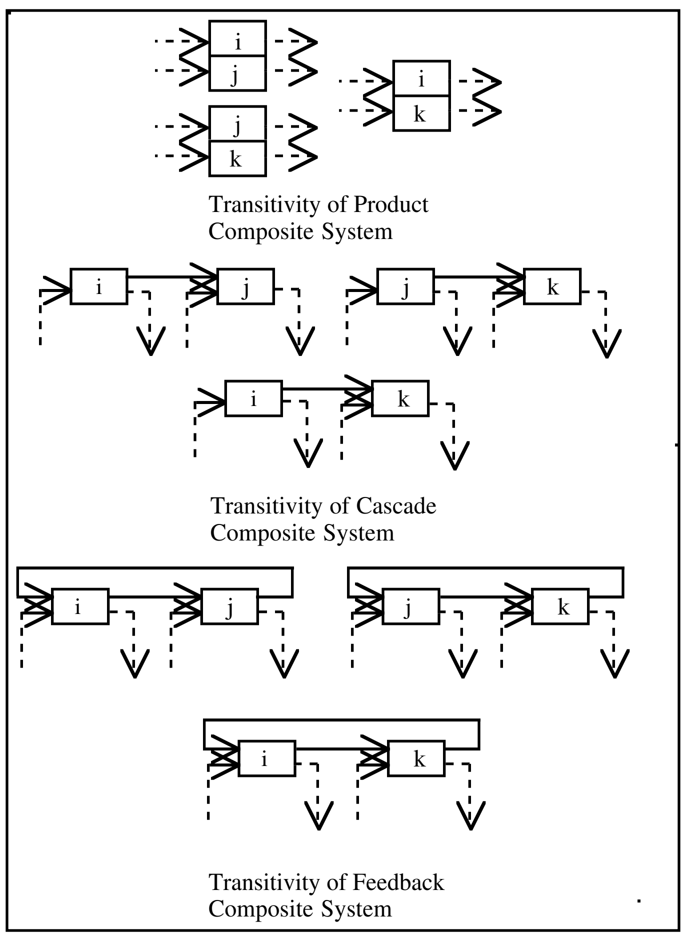

Figure 5.3: Transitivity of Composite Structures

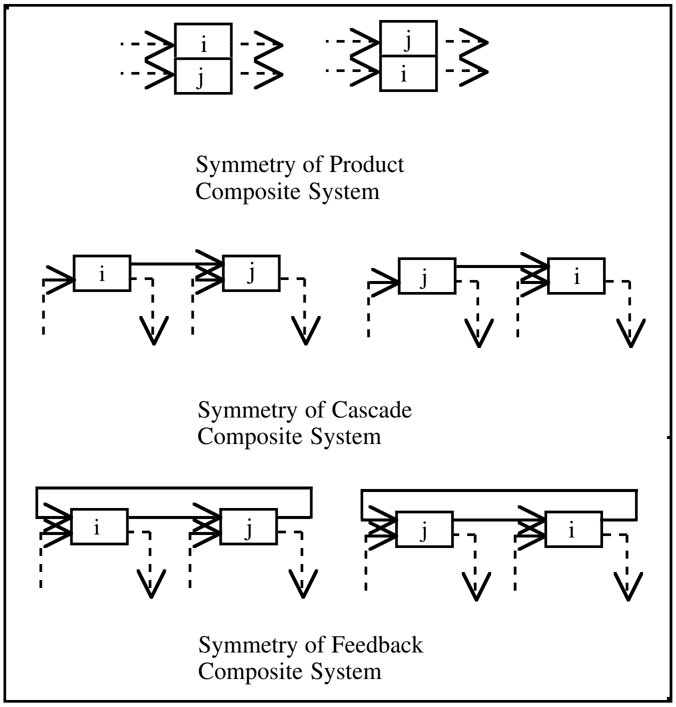

Figure 5.4: Symmetry of Composite Structures

A transitive property P_k that holds for the composition of components C_i and C_j, and the composition of components C_j and C_k, will also hold for the composition of components C_i and C_k. A property P_k is transitive on composition if and only if:

$$\mathbf{sat}(\boldsymbol{\&}(C_i, C_j), P_k) \wedge \mathbf{sat}(\boldsymbol{\&}(C_j, C_k), P_k) \Rightarrow \mathbf{sat}(\boldsymbol{\&}(C_i, C_k), P_k)$$

The transitive composition of components is shown in Figure 5.3.

A symmetric property P_k that holds for the composition of C_i and C_j will hold for the composition of components C_j and C_i. A property P_k is symmetric on composition if and only if:

$$\mathbf{sat}(\boldsymbol{\&}(C_i, C_j), P_k) \Rightarrow \mathbf{sat}(\boldsymbol{\&}(C_j, C_i), P_k)$$

The symmetric composition of components is shown in Figure 5.4.

If a property is antisymmetric on composition, then

$$\mathbf{sat}(\boldsymbol{\&}(C_i, C_j), P_k) \wedge \neg\, \mathbf{sat}(\boldsymbol{\&}(C_j, C_i), P_k)$$

Antisymmetry does not allow for two-way communication, or feedback, between components.

A property that is reflexive, transitive, and symmetric with respect to composition will be satisfied by the general composition of components. This corresponds to an equivalence relation for the components of the system. An equivalence relation defines an equivalence class over a set A for a relation R:

$$[a]_R = \{ b : b \text{ is in } A \text{ and } aRb \}$$

The analogous equivalence relation for properties of systems is

$$[C_i]_R = \{ C_j : C_j \text{ is in } \mathbb{C} \text{ and } \mathbf{sat}(\boldsymbol{\&}(C_i, C_j), P_k) \}$$

where $\mathfrak{C}$ is a set of components. Thus the components in the "equivalence classes" will satisfy the property P_k for the general composition of the components.

Closure of a property for a component, product, cascade, or feedback system indicates that the component, product, cascade or feedback system may be composed with like systems. Transitivity and closure indicate that a property will continue to be satisfied on hierarchical composition.

Reflexivity allows for self-feedback and symmetry allows for feedback. If the components of a system with a feedback connection can be represented as a single composite system with a reflexive connection, then reflexivity and feedback can be shown to be analogous. In the context of hierarchical composition, feedback across a composite system implies reflexive composition of the system. When we hierarchically compose systems, each system is treated as an individual component. Thus the feedback composition across the system corresponds to a reflexive connection treated as an individual component.

It is not always the case, however, that feedback over multiple components can be represented as reflexivity over a single complex component. A system with multiple feedback connections between different components can not always be simplified into a single composite system with a single reflexive feedback connection. For such a case, feedback over the system does not imply reflexivity over the system. Thus the set of reflexive connections is a subset of the set of feedback connections.

Given these meta-properties and the corresponding composite constructs we are able to compose general systems and predict the satisfiability of properties for these systems.

5.7 Completeness of Set of Properties

The properties of a system must accurately represent the complete set of requirements for a component's behaviour. These properties must define *all* goals of the system behaviour. In particular, they must disallow all behaviours that violate these goals. We postulate that we cannot prove or even know if we have defined all properties relevant to a goal such as "safe behaviour" or "live behaviour." This is because these goals change with the threats, risks, and vulnerabilities of systems and their information.

If we can identify ALL possible threats, risks, and vulnerabilities, then we can define properties of behaviours required to nullify these threats. Threats, risks, and vulnerabilities may change, however, dependent upon the information in the system; the remaining life of the system or its contained information, the allowable users of the system, et cetera. It has been observed that

> Means of causing loss can be categorized only incompletely in ways that always leave the next possible uncategorized means of causing the next loss a mystery ... There will always be a new problem to face. [Par91]

We believe that we must define the properties of a system based on a detailed knowledge of the system, its intended use, previously defined properties, and the known and postulated threats to the system.

6. Safety Properties

> Since therefore the knowledge and survey of
> vice is in this world so necessary to the
> constituting of human virtue, and the
> scanning of error to the confirmation of truth,
> how can we more safely, and with less
> danger, scout into the regions of sin and
> falsity than by reading all manner of
> tractates and hearing all manner of reason?
> JOHN MILTON: Areopagitica

We know from previous work that there are a number of important safety properties. Safety properties, expressed using temporal logic, are those of the form

$$\circ\ S$$

where S is a static formula. Each safety property must always be true, for all time and for all executions of an event system.

In this chapter we introduce a set of basic safety properties, given the general definition of properties discussed in Chapter 5. The basic safety properties define the minimum requirements for the safe behaviour of a system. We define and discuss the role of these properties in the safety of a system. In particular, we focus on the assumptions that are made about the behaviours of the system and its environment, and how these assumptions affect the satisfiability of a property. These assumptions are often associated with different levels of optimism about the behaviour of the system or environment.

We discuss the necessity of the basic safety properties in the overall safety of a system and examine the meta-properties of these properties. We also address the identification and specification of safety properties that are unique to a given system.

6.1 Basic Safety Properties

The basic safety properties introduced in this chapter are based on the safety properties introduced in [Hin93, HiLe93]. We refer to these properties as

- Authenticity
- Integrity
- Confidentiality

The definitions of confidentiality and integrity are very restrictive, pessimistic, approaches. They correspond to the idea that *no* unacceptable events may be possible input events. This has the result that *any* event that is unacceptable to at least one component must be treated as unacceptable to all components; these properties will be satisfied only by very restricted systems.

6.1.1 Authenticity

Authenticity requires that all events produced by a component or the environment be defined and genuine.

$$\mathrm{AU} \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \exists A : \mathcal{F}ma\{\Phi(i)\} \cdot \sigma\{\mathbf{i}\}(\mathrm{out}) \models_{x} A$$

$$\mathrm{AU\text{-}E} \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \exists A : \mathcal{F}ma\{\Phi(\mathbb{E})\} \cdot \sigma\{\mathbb{E}\}(\mathrm{out}) \models_{x} A$$

Because a component is defined to produce only acceptable output events in response to acceptable inputs the known and genuine events authentically produced by a component will be acceptable events.

6.1.2 Integrity

No unacceptable requests to modify information will be received by the system.

$$\mathrm{I} \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \exists A : \mathcal{F}ma\ \{\Phi(\mathbf{N},\mathrm{obs}) \cup \Phi(\mathbf{N}(A),\mathrm{mod})\} \cdot \sigma\{\mathbf{i}\}(\mathrm{in}) \models_{x} A$$

This property constrains the modification-type events and makes no requirements on the observation-type events. We therefore include in the specification of this property domain both the constrained modification events and the unconstrained observation events. Integrity of the information and the system, is maintained by only allowing known, acceptable changes to and within the system.

6.1.3 Confidentiality

No unacceptable requests to observe information will be received by the system.

$$\mathrm{C} \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \exists A : \mathcal{F}ma\ \{\Phi(\mathbf{N}(A),\mathrm{obs}) \cup \Phi(\mathbf{N},\mathrm{mod})\} \cdot \sigma\{\mathbf{i}\}(\mathrm{in}) \models_{x} A$$

This property places a restriction on the observation events only. Whereas the integrity property must include the unconstrained observation events, the confidentiality property must include the unconstrained modification events.

Confidentiality, or secrecy, of information is maintained by monitoring the disclosure of information in the system, and the requests to disclose this information. We separate the notion of integrity and confidentiality into observation and modification of information for several reasons. This approach will allow different policies to be enforced for observation and modification of information within a system.

6.2 Safety Characteristics

6.2.1 Safe Component

A safe component is one that behaves in a known, predictable manner, satisfies the basic safety properties, and behaves in a manner consistent with any additional goals of safe behaviour.

6.2.2 Safe Environment

A safe environment will not send unacceptable (potentially harmful) events to a component. If the environment does contain unacceptable events then the unpredictable behaviour of the users in the environment may not cause any unacceptable events to be sent to the system. Such an environment is safe only if we can trust the users not to send these unacceptable events to the component.

Within a composite system, a component's meta-environment is safe if it will not receive any unacceptable events from the environment or from any interconnected components. Thus a safe meta-environment depends on the unpredictable behaviour of the users and the predictable behaviour of the components.

6.2.3 Safe System

A safe system is made up of safe components, each within a safe meta-environment, that itself has a safe environment. The behaviour of a safe system must satisfy the (necessary) basic safety properties. Additional, system-specific, safety properties may be required by a system for its overall behaviour. This will be addressed in detail in section 6.7.

6.3 Domain-Dependent Safety Properties

The basic safety properties of confidentiality and integrity are, by definition, domain-dependent properties. These properties are requirements on the predictable behaviours of the components in a system and prescriptions on the unpredictable behaviours of the users of a system. Any change to a system or the meta-environment, including the users of the system, may affect the satisfiability of these properties.

For this reason we introduce different levels of strength, or optimism, of confidentiality and integrity. These different levels of strength correspond to different assumptions about the behaviour of the meta-environments of a system. We define three optimistic properties for each of confidentiality and integrity: environment optimistic, component optimistic, and strongly optimistic. The definition of integrity and confidentiality given above corresponds to a pessimistic integrity and confidentiality. These properties are based on similar properties introduced in [Hin93].

6.3.1 Pessimistic

Only those acceptable (observation, modification) events that are common to *all* components in the system will be allowable and possible.

$$PC \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{ \Phi(\mathcal{N}(A),obs) \cup \Phi(\mathcal{N},mod) \} \cdot \sigma\{\mathbf{i}\}(in) \models_x A$$

$$PI \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{ \Phi(\mathcal{N},obs) \cup \Phi(\mathcal{N}(A),mod) \} \cdot \sigma\{\mathbf{i}\}(in) \models_x A$$

For a simple system, pessimistic confidentiality and integrity are defined as:

$$PC \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{ \Phi(\mathcal{N}(A),obs) \cup \Phi(\mathcal{N},mod) \} \cdot \sigma(\mathcal{E};i)(in) \models_x A$$

$$PI \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{ \Phi(\mathcal{N},obs) \cup \Phi(\mathcal{N}(A),mod) \} \cdot \sigma(\mathcal{E};i)(in) \models_x A$$

These properties are therefore assumptions about the behaviour of the environment.

6.3.2 Environment Optimistic

The environment may contain unacceptable (observation, modification) events; the agents in the environment will not send harmful events to vulnerable components. No component in the system will produce an event that is unacceptable to any component in the system.

$$\begin{aligned} EOC \equiv\ & \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{\Phi(i(A),obs) \cup \Phi(i,mod)\} \cdot \\ & \quad \exists B : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot \\ & \quad A,B \in \mathcal{F}ma\{\Phi(i(A),obs) \cup \Phi(i,mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B \end{aligned}$$

$$\begin{aligned} EOI \equiv\ & \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathit{Fma}\{\Phi(i,obs) \cup \Phi(i(A),mod)\} \cdot \\ & \quad \exists B : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot \\ & \quad A,B \in \mathcal{F}ma\{\Phi(i,obs) \cup \Phi(i(A),mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B \end{aligned}$$

6.3.3 Component Optimistic

Components in a system may produce events that are unacceptable (obs, mod) events if received at other components; these events will not be sent to any vulnerable components. It is not possible for the environment to send an unacceptable event to a component in the system

$$\begin{aligned} COC \equiv\ & \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot \\ & \quad \exists B : \mathcal{F}ma\{\Phi(i(A),obs) \cup \Phi(i,mod)\} \cdot \\ & \quad A,B \in \mathcal{F}ma\{\Phi(i(A),obs) \cup \Phi(i,mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B \end{aligned}$$

$$\begin{aligned} COI \equiv\ & \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists A : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot \\ & \quad \exists B : \mathcal{F}ma\{\Phi(i,obs) \cup \Phi(i(A),mod)\} \cdot \end{aligned}$$

$$A,B \in \mathcal{F}ma\{\Phi(i,obs) \cup \Phi(i(A),mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B$$

6.3.4 Strongly Optimistic

The meta-environment may contain unacceptable (observation, modification) events. The agents in the environment will not send harmful events to vulnerable components. The components in the meta-environment may produce unacceptable (obs, mod) events; these events will not be sent to any vulnerable components.

$$SOC \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists\, A,B : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot$$
$$A,B \in \mathcal{F}ma\{\Phi(i(A),obs) \cup \Phi(i,mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B$$
$$SOI \equiv \forall x : \mathbb{N} \cdot \forall i : \mathcal{N} \cdot \exists\, A,B : \mathcal{F}ma\{\Phi(obs) \cup \Phi(mod)\} \cdot$$
$$A,B \in \mathcal{F}ma\{\Phi(i,obs) \cup \Phi(i(A),mod)\} \wedge \sigma(\mathcal{N};i)(in) \models_x A \wedge \sigma(\mathcal{E};i)(in) \models_x B$$

We are optimistic about both the behaviour of the components (given the interconnections of the system) and the behaviour of the environment and therefore strongly optimistic about the system and meta-environment as a whole.

6.4 Necessity of the Basic Safety Properties

Safety is violated (by definition) when harmful (unacceptable or undefined) events are received at a component in a system. These events may be potentially received from known sources (components or the environment) or from unknown sources. The basic safety properties counter these possible situations and are therefore necessary for the safe operation of components and systems.

The details of the proofs of the necessity of these properties are given in Appendix B.

6.5 Meta-Properties of Safety Properties

Table 6.1 illustrates the meta-properties of the basic safety properties given different composite structures. The meta-property of reflexivity of a safety property is independent of the composite structure of a system. This is because reflexivity is defined over a single component, regardless of the composite structure in which it exists. The meta-property of symmetry follows automatically for a feedback composite structure for all safety properties, because the feedback structure is a symmetric structure.

Demonstration of Necessity:
Assume that safety is violated by a known event;
this known event may be acceptable or unacceptable.
    if the event is received from a known source **then**
        if the event is unacceptable **then**
            if the event is an observation event
                then Confidentiality is not satisfied
            else if the event is a modification event
                then Integrity is not satisfied
        else if the event is acceptable
            then Safety is not violated.
    else if the event is received from an unknown source
        then Provenance (see Chapter 7) is not satisfied
Assume that safety is violated by an unknown event
    if the event is received from the environment
        then Authenticity of environment is not satisfied
    else if event is not received from the environment **then**
        if event is received from a known component
            then Authenticity of component is not satisfied
        else if event is received from an unknown source
            then Provenance (see Chapter 7) is not satisfied

Figure 6.1: Necessity of Basic Safety Properties

The basic safety properties do not, in general, exhibit reflexivity. The outputs produced by a system, although they are defined to be acceptable, are acceptable *outputs*. They are not necessarily acceptable *inputs* to the system. Reflexivity may follow if all possible outputs are also acceptable inputs. The acceptability of the outputs as inputs must be assessed for each individual system; the reflexivity of a property cannot be predicted in advance.

The meta-properties of the basic safety properties reflect the role of the environment and the domain-dependence of the properties. None of the safety properties exhibit transitivity or closure for any of the composite structures. This is because of the restrictions and prescriptions placed on the environment by these properties. If the environments of two systems are the same and both systems satisfy the same level of optimism (either pessimistic (PC/I) or component optimistic (COC/I) confidentiality or integrity) then the properties of authenticity (AU), confidentiality and integrity will be transitive and closed under composition. The properties of environment optimistic (EOC/I) and strongly optimistic (SOC/I) confidentiality and integrity will not exhibit transitivity or closure unless the environments of two components are the same and the prescriptions on the users of the environment are the same. This is a direct result of the domain-dependence of these properties. The assessment of the meta-properties of the basic safety properties is included in Appendix C.

COMPOSITE STRUCTURE	SAFETY PROPERTIES				
Meta-Property	AU	PC/I	EOC/I	COC/I	SOC/I
Reflexive	~x	~x	~x	~x	~x
Product					
Transitive	~x	~x	~x	~x	~x
Symmetric	√	√	√	√	√
Closure	~x	~x	~x	~x	~x
Cascade					
Transitive	~x	~x	~x	~x	~x
Symmetric	~x	~x	~x	~x	~x
Closure	~x	~x	~x	~x	~x
Feedback					
Transitive	~x	~x	~x	~x	~x
Closure	~x	~x	~x	~x	~x

√ = Property exhibits meta-property

~x = Property exhibits meta-property if meta-environment unchanged, where meta-environment is usually affected given this composite structure

Table 6.1: Meta-Properties of Basic Safety Properties

6.6 Threat, Risk, Vulnerability and Trust

We have alluded to the threat, risk, and vulnerability associated with events and to the trust that is placed in users. These are distinct ideas, even though they may have similar effects with respect to the domain-dependent properties.

Trust in a user is independently assessed and quantified. The trust placed in a user is a reflection of our belief that they will or will not behave in a particular manner. If we do not trust a user, it behoves us to assume that they will always behave contrary to our safety assumptions.

Just as we trust (or do not trust) users, we may choose to trust or to mistrust a component. Because a component's behaviour is predictable, the trust we place in a component is a reflection of the confidence we place in the specification of the component's behaviour and the verification of its correct behaviour. For example, if a component is a commercial-off-the-shelf (COTS) component, we may choose not to verify the specification of its behaviour, instead choosing not to trust the component.

The measure of {Threat, Risk, Vulnerability} ({TRV}) is used to examine the threat that an event poses to a system if received, the risk of it being received by the system, and the vulnerability of the system at which it is received [Bony95]. A detailed exposition of this measure is given by Thompson et. al. [TLBS88].

We must consider the {TRV} associated with an event if we choose to allow an unacceptable event to be a possible event within a system. The {TRV} assessment does not affect the acceptability or unacceptability of an event. It is used to determine if some events are more harmful than others when examined in the context of the threat posed by the event, the risk associated with the event, and the vulnerability of the system to such an event. Events that are judged to be "less" harmful may be allowed within a system because the {TRV} indicates that they are not harmful "enough" to warrant protective action by the system.

We suggest that in addition to the recommendation proposed by the {TRV} assessment, we must include the notion of the amount of trust in the possible sources of an event. If there is no trust in the source, then we cannot allow a potentially unacceptable event to be a possible event, no matter what the associated {TRV} measure. If we have absolute trust in a source, then we may allow a potentially unacceptable event to be a possible event, even given an unfavourable {TRV} measure for the event.

These are not concepts that are easily quantified in formal properties or assertions. They must be assessed for individual systems, taking in to account the intended use of a system. Different levels of {TRV} and trust can be handled with different levels of optimism using the domain-dependent properties, and incorporated into a risk assessment.

6.7 Defining System Specific Safety Properties

The basic safety properties introduced in this work are necessary for the safe operation of a system. They are not always sufficient, however. The safe interaction of components in a system may require that we consider additional properties. These properties will in part depend on the application in question. For example, a multi-level security system may require possibilistic security properties such as non-interference, whereas a network system may require covert-channel preventing properties.

The definition of application-specific system safety properties will not be addressed in this work. We postulate that many, if not all, relevant properties will be definable based on the experience of the system designers given the overall goals of the system's behaviour. We also believe that many interesting safety properties are actually progress properties given the formal classification of properties. We examine the classification of system-specific progress properties in Chapter 7.

7. Progress Properties

Go while the going's good
Is my advice...
SOPHOCLES: Philoctetes
(tr . E.F. Whatling)

We know from previous work that there are a number of important progress properties. A progress property is one that includes the unary temporal operator, $\Diamond$, "eventually". The most general form of a progress property is a reactivity property, defined by Manna and Pnueli [MaPn92] as

$$\bigwedge_{i=1}^{n} (\circ \Diamond A \vee \Diamond \circ B)$$

where A and B are static formulae.

In this chapter we introduce a set of basic progress properties, given the general definition of properties discussed in Chapter 5. The basic progress properties define the minimum requirements for the progress of a system. We define and discuss the role of these properties in the progress of a system. In particular, we focus on the assumptions that are made about the behaviours of the system and its environment, and how these assumptions affect the satisfiability of a property.

We discuss the necessity of the basic progress properties in the overall progress of a system and examine the meta-properties of these properties. We also address the identification and specification of progress properties that are unique to a given system.

7.1 Basic Progress Properties

The basic progress properties are

- Accessibility
- Eventual Output
- Termination
- Eventual Progress
- Faithfulness
- Provenance
- Utility

These properties are based on the basic actions that every component will exhibit: receive an event, (input), process it, (process), and produce a result (output). These properties are necessary requirements for a component to progress. They apply to individual components, and the processes that run on them.

We have separated the idea of "input eventually produces an output" into the four properties of accessibility, eventual output, termination and eventual progress. This is intended to highlight the different progress related actions of a component and their individual roles in the overall progress of the system.

7.1.1 Accessibility

Every event that is sent as an input to a component is eventually accepted for processing by that component. Accessibility requires that all events will eventually initiate processing at the component.

$$AC \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \sigma\{\mathbf{i}\}(in) \models_x \alpha \;\Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \alpha$$

Accessibility implies that all inputs are state-independent, that is, can always be (eventually) processed by a component, regardless of the state of a component. If an event is state-dependent, then the satisfiability of accessibility requires that the input(s) required to enter the required state, from which a desired input may be processed, will always eventually be received.

7.1.2 Eventual Output

Every process executing at a component will eventually produce an output on one of the component's output channels.

$$EO \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \exists \beta : \Phi(i(out)) \cdot \sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Diamond \beta$$

This property may seem to be counter-intuitive in that there are processes that may not produce an output. At the level at which we are initially modelling a system, however, we are almost always interested in processes that do produce an output, if only as a means to notify a user that a process has completed.

The combination of the two basic progress properties, accessibility and eventual output, gives a composite property known as eventual response: every input event will eventually produce an output event.

$$\forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \exists \beta : \Phi(i(out)) \cdot$$
$$(\sigma\{\mathbf{i}\}(in) \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \alpha) \wedge (\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Diamond \beta)$$
$$= \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \exists \beta : \Phi(i(out)) \cdot$$
$$(\sigma\{\mathbf{i}\}(in) \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \alpha) \wedge (\sigma\{\mathbf{i}\} \models_x \Diamond \alpha \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Diamond \Diamond \beta))$$
$$= \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \exists \beta : \Phi(i(out)) \cdot (\sigma\{\mathbf{i}\}(in) \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Diamond \beta)$$

7.1.3 Termination

Every process executed by a component will eventually complete.

$$TR \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi \cdot \sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \alpha$$

The basic progress property of termination is used to prove that processes will eventually terminate. If we wish to prove that a process, such as an operating system, does not terminate, we prove that the termination property is not true for that process.

Termination implies that a process is self-contained, that is, independent of the inputs or other processes being received or executed by a component. If a process is not self-contained, then it cannot terminate without a required input. If the input is not received, the process will not terminate and the property is not satisfied. We cannot guarantee termination for such a process if we cannot guarantee that the required input will always be received.

7.1.4 Eventual Progress

Every process active at a component will eventually lead to a different process being active at the component.

$$EP \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi \cdot \exists \chi : \Phi \cdot \sigma\{\mathbf{i}\} \models_x \Diamond \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \chi$$

This property ensures that a component will continue to process events. This prevents a process from disabling a component for use by other processes.

The basic progress property of eventual progress is used to prove that the component will not halt, but continues to progress by always allowing for a different process to be active.

7.1.5 Faithfulness

Faithfulness requires that the component behaves as a predictable process; given the same inputs, we can predict the possible outputs.

$$FT \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \Phi(i(in)) \cdot \exists \beta : \Phi(i(out)) \cdot$$
$$((\alpha ; \beta) \wedge \sigma\{\mathbf{i}\}(in) \models_x \alpha) \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x \Diamond \beta$$

If the property of faithfulness is not satisfied by the behaviour of a component then we have not completely defined the component's behaviour or it is behaving in some new, unpredictable behaviour.

The interleaving of events on output is handled by the specification of eventually, $\Diamond\beta$. While we can predict the possible output events, we cannot predict when they will occur. This follows from $\Diamond\beta$; given $\models_x \Diamond\beta$ and $\models_x \Diamond\chi$ we cannot determine exactly when in the future β or χ will occur.

7.1.6 Provenance

Every event received as an input is the output of a known, defined source, either a component in the system or the environment.

$$PV \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \forall A : \mathcal{F}ma\{\Phi\} \cdot \sigma\{\mathbf{i}\}(in) \models_x \Diamond A \Rightarrow \sigma(\mathbf{N}^{+}\mathbf{E};i)(out) \models_x A$$

Provenance requires that all events have a known, defined source, the environment or a known component. Provenance may appear to have the same goal as authenticity yet they place distinct requirements on the behaviour of a system. It is possible for authenticity to be violated while provenance is satisfied, for example, if a system administrator introduces a new, previously unconceived module. In this case provenance is satisfied because the module is a legitimate component within a system and the provenance of any events sent by this component can be determined. Authenticity may be violated, however, if the domain required by the composite system disallows the known, acceptable outputs of the new module within the context of the composite system. It is also possible for provenance to be violated while authenticity is satisfied, for example, if an intruder initiates a well-known and understood `rm *.*` command within the system. The provenance of the event is not satisfied because the intruder is not defined component within the system, yet authenticity may be satisfied if the `rm *.*` is an acceptable event (in certain situations) within the system.

In many applications, cryptography is used to provide a combination authenticity and provenance function. Public and private encryption keys are used to verify that a user is who s/he claims to be (authentic). The idea of provenance is inherent in the use of the correct keys. Neither authenticity nor provenance, as we have defined them, is explicitly handled using cryptography. Any such approach must be examined to ensure that it provides the level of protection required by these individual properties.

7.1.7 Utility

Every output produced by a component is sent to a known, defined source, either another component or the environment.

$$UT \equiv \forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \forall A : \mathcal{F}ma\{\Phi\} \cdot \sigma(\mathbf{N}^{+}\mathbf{E};i)(out) \models_x A \Rightarrow \sigma\{\mathbf{i}\}(in) \models_x \Diamond A$$

Utility is the opposite of provenance. Utility requires that every output channel of all components is connected to a valid destination component, or the environment.

Provenance and utility together ensure the safety and correctness of the connections of a system. Together, they ensure that new, unknown sources are not inserted in a system and that outputs are not diverted from the system. These properties do not deal with situations such as passive eavesdropping or wiretapping. These are physical attacks that are external to the behaviour of a system.

7.1.8 Additional Properties

The additional property of *progress on interconnection* will ensure fairness on composition by requiring that connections behave as if they are live. Every output, β, sent by a component **i** as an input to component **j** will eventually reach that component.

$$\mathrm{POI} \equiv \forall x : \mathbb{N} \cdot \forall i,j : \mathbb{N} \cdot \forall \beta : \Phi \cdot \ (\sigma(i;j)(out) \models_x \beta \Rightarrow \sigma(i;j)(in) \models_x \Diamond \beta)$$

This property follows from the specification of an interconnection between two components, the properties of provenance and utility.

7.2 Liveness Characteristics

Progress, as defined by Manna and Pnueli [MaPn92] and used in this work, does not correspond to liveness, as defined by Alpern and Schneider [AlSc85a, AlSc87a] (see Chapter 2.5). Nevertheless, we refer to the liveness of a system as the sum of its progress properties. We believe that it is more intuitive to refer to a live system than a progressive system. A system that exhibits the basic progress properties will be said to be live.

7.2.1 Live Components

A live component is one that behaves in a known, predictable manner, satisfies the basic progress properties, and behaves in a manner consistent with any additional goals of live behaviour.

7.2.2 Live Environments

A live environment is defined to be one that is capable of generating events to be sent to a component or a system.

Within a composite system, a component's meta-environment is live if it is capable of generating events to be sent to the component. Thus a live meta-environment depends on the live behaviour of the users and the live behaviour of the components.

7.2.3 Live Systems

A live system is made up of live components, each within a live meta-environment, that itself has a live environment. The behaviour of a live system must satisfy the (necessary) basic progress properties. Additional, system-specific, progress properties may be required by a system for its overall behaviour. These properties will reflect such requirements as freedom from deadlock and livelock. This will be addressed in detail in a later section.

7.3 Domain-Independent Progress Properties

The progress properties are domain-independent properties; they do not depend on assumptions about the input events that will be sent to the components. Changes to a system that do not affect the overall environment (that do not introduce new events or remove existing events from the environment or meta-environment) will not affect the satisfiability of these properties.

7.4 Necessity of Basic Progress Properties

Demonstration of Necessity:

```
Assume an event is sent to a component
    if the event never initiates a process at the component
        then Accessibility is not satisfied
    else if the event initiates a process then
        if the process cannot execute then
            if there is a process preventing desired process
              from executing then
                then Termination is not satisfied
            else if there are no other processes executing
                then Eventual Progress is not satisfied
        else if the process can execute then
            if the process never produces an output event
                then Eventual Output is not satisfied
            else if process produces an output event then
                if output is not a predicted component output
                    then Faithfulness is not satisfied
                else
                    Progress is satisfied
```

Figure 7.1: Necessity of Basic Progress Properties

Progress is violated (by definition) when an input is sent to a component, yet the component is unable to process the event, produce a corresponding (predictable) output, and continue processing other events. The component must always eventually produce a predictable output; if we cannot predict the possible outputs, we cannot determine which outputs correspond to which inputs and cannot therefore determine if the component will exhibit progress.

The details of the proofs of necessity are given in Appendix B.

7.5 Meta-Properties of Progress Properties

Table 7.1 shows the meta-properties of the progress properties for the three types of composition of components. The assessment of the meta-properties of the basic progress properties is included in Appendix D.

The meta-properties of accessibility (AC), eventual output (EO), termination (TR), and eventual progress (EP) reflect the role of inputs to the system in the satisfiability of these properties. If the behaviours of the system are input-state independent then the meta-properties are satisfied regardless of the possible inputs or possible ordering of inputs. If the behaviours satisfying these properties are not input-state independent, then we must ensure that the interconnections of the composite system do not disallow the inputs required to maintain the satisfiability of these properties. This must be assessed on a system-by-system basis. Thus we must examine the behaviours of a composite system in order to evaluate the satisfiability of these properties.

COMPOSITE STRUCTURE	PROGRESS PROPERTIES						
Meta-Property	AC	EO	TR	EP	FT	PV	UT
Reflexive	~x	~x	~x	~x	~x	√	√
Product							
Transitive	~x	~x	~x	~x	~x	√	√
Symmetric	√	√	√	√	√	√	√
Closure	~x	~x	~x	~x	~x	√	√
Cascade							
Transitive	~x	~x	~x	~x	~x	√	√
Symmetric	~x	~x	~x	~x	~x	√	√
Closure	~x	~x	~x	~x	~x	√	√
Feedback							
Transitive	~x	~x	~x	~x	~x	√	√
Closure	~x	~x	~x	~x	~x	√	√

√ = Property exhibits meta-property

~x = Property exhibits meta-property if meta-environment unchanged and all behaviours are independent of state of component

Table 7.1: Meta-Properties of Basic Progress Properties

Faithfulness (FT) will exhibit reflexivity, transitivity, symmetry or closure if the progress properties of accessibility and eventual output also exhibit these meta-properties. This is because faithfulness combines these two properties together with information on the input-output relation between events. If these progress properties are satisfied, then the progress aspect of faithfulness will follow. The relation between input and output events at a component will not be affected by the interconnections of other components.

The properties of provenance (PV) and utility (UT) are dependent upon the interconnections within the system. We assume that when a system is created, all and only the required interconnections are made. Given this assumption, these properties exhibit reflexivity, symmetry, transitivity and closure.

7.6 Defining System Specific Progress Properties

The basic progress properties introduced in this work are necessary for a system to be live. The live interaction of components in a system may require that we consider additional progress properties. These properties will in part depend on the application in question. For example, a network system may require the eventual reliability of a lossy communication channel, whereas a database system may be required to be free from deadlock.

The definition of application-specific system progress properties will not be addressed in detail in this work. We postulate that many, if not all, relevant properties will be definable based on the experience of the system designers given the overall goals of the system's behaviour, and can be easily determined given the classification of the progress properties (see section 2.5.2 or [MaPn92]).

7.6.1 Classification of System Specific Progress Properties

In this section we will briefly highlight the progress classes of Manna and Pnueli [MaPn92] and the role of properties within these classes in defining a system-specific progress property. The basic progress properties belong to the response class of properties, which strictly includes the obligation class of properties, and the safety and guarantee classes of property. Unless we have defined *all* possible response properties, however, we cannot be sure that we have defined all relevant safety, guarantee or obligation formulae.

Guarantee properties, $\Diamond\beta$, can be used to specify that required events occur at least once. Because all events occurring at a component are defined by an input-output behaviour rule, and therefore a response formula, the guarantee properties describing component behaviour are already defined and included in the set of response formulae.

Guarantee formulae may also be used to describe the behaviour of the environment, in particular when there are certain events that we know *must* eventually occur, such as $\Diamond$logoff. Environment actions that *may* occur cannot be defined using guarantee formulae; we cannot place a guarantee on the prescribed behaviour of the environment. We may, however, choose to define such guarantee properties in order to clarify the implicitly made assumptions on an environment's behaviour. This will then allow us to determine what will happen if these assumptions are not met.

Obligation properties, $(\circ\, \alpha \vee \Diamond\beta)$ or $(\Diamond\alpha \rightarrow \Diamond\beta)$, require that if one event occurs then another corresponding event must eventually occur, or, one event is always true or another event eventually becomes true. Obligation formulae may be used to ensure that related events occur, where if one event occurs, we know that a related event must have already occurred or will eventually occur.

The response properties, $\circ\, \Diamond\alpha$ or $\circ\, (\alpha \rightarrow \Diamond\beta)$, describe the required behaviour of a component. These properties are determined by applying the progress requirements to the input-output behaviour rules defining the component behaviour, resulting in a set of response formulae describing the behaviour of a system. For example, the input-output rule $(\alpha\, ;\, \beta)$ gives the response property $\circ\, (\alpha \rightarrow \Diamond\beta)$.

Response properties may also be defined for related pairs of input events and related pairs outputs of events. For example, if a component produces a request to write to a file, the requirement that the same component eventually produce a request to close the file is represented by a response property:

$\circ$ (request_open_for_write_file $\rightarrow \Diamond$request_close_file)

We can also define response properties that place similar assumptions on the required behaviour of the environment, so that a user requesting to open a file will eventually close that file.

The class of persistence formulae, $\Diamond \circ \beta$, is the dual of the class of the response formulae. This property specifies that something eventually becomes true for all time, such as Lamport's requirement on stuttering [Lam77].

Reactivity properties, $(\circ\, \Diamond\alpha \vee \Diamond \circ\, \beta)$ or $(\circ\, \Diamond\chi \rightarrow \circ\, \Diamond\alpha)$ or $(\Diamond \circ\, \delta \rightarrow \Diamond \circ\, \beta)$, are used "to describe a response of a more complicated type, which does not guarantee a response to a single stimuli ... we want to commit the system to eventually respond to sufficiently many stimuli, but not specify a bound on how many stimuli may happen before the eventual response" (p. 294, [MaPn92]).

If we represent the reactivity schemata as $(\neg\circ\, \Diamond\alpha \rightarrow \Diamond \circ\, \beta)$, or, equivalently, $(\Diamond \circ \neg\alpha \rightarrow \Diamond \circ\, \beta)$, then if a sequence eventually stabilises to $\neg\alpha$, then it will eventually stabilise to β.

If we represent the reactivity schemata as $(\neg\Diamond \circ\, \beta \rightarrow \circ\, \Diamond\alpha)$, or, equivalently, $(\circ\, \Diamond\neg\beta \rightarrow \circ\, \Diamond\alpha)$, this can be represented as a dependency between two response formulae, $\circ\, (\neg\beta \rightarrow \Diamond\chi) \rightarrow \circ\, (\alpha \rightarrow \Diamond\delta)$. Thus if one response must trigger another response, we can represent this as a reactivity property.

We can represent the reactivity schemata in infinitely many equivalent forms. We introduce these two equivalent forms based on response and persistence in the belief that any reactivity properties containing new restrictions on the behaviour of the system will be due to the interrelationship between these two "lower" schemata.

When defining system-specific progress properties, we must do our utmost to ensure that a property is defined in the simplest class possible. As we move up the lattice of progress properties, we must ensure that each new property brings an additional, previously unconsidered requirement to the table. This makes it easier to understand its goals and to ensure its truth for a system's behaviour.

The definition of system-specific properties will require a detailed knowledge of the intended operation of the system. A system administrator should be able to use the general definitions of progress classes to help determine if there are specific requirements unique to a given system. This method is no more scientific that current hit-or-miss methods; it is hoped that the consideration of these progress classes will make the method more comprehensive than current approaches.

It is reasonable to expect that most progress properties required will be known from previous experience with system analysis. We must ensure that progress properties are correctly identified as such. Properties, such as faithfulness, that appear to be safety properties must be correctly identified as progress properties.

7.7 Bounded Response Time of a System

Progress and liveness properties take advantage of the temporal operator $\Diamond$. This implies only that something will happen eventually (in the future). It places no restrictions on how far into the future something will happen. Progress may therefore be theoretically satisfied, even if a required action takes ten years to complete. This indicates that there are some practical problems with considering progress within the context of its formal definition.

For example, the termination property requires that a process will terminate at some time in the future. Arguably, if a process does not terminate within some finite, bounded time then we may as well consider it as not ever terminating, even if it can theoretically finish in the distant future. Therefore we introduce the notion of *bounded* or *practical fairness*. Bounded fairness states that a required action always occurs within a finite bounded time τ.

Basic Assumption of Bounded or Practical Fairness
An atomic action will always terminate within a finite, predetermined time, τ.

We motivate the use of bounded fairness by noting that most users of computer systems have an intuitive notion of the response time that a system will exhibit for a given task. If the task greatly exceeds this time, the system is assumed to be dead, hung or otherwise malfunctioning; the standard reaction of many users is to reboot the system and start over. If the system does not respond within a specified time, it is not behaving correctly, regardless of whether or not it is theoretically considered to be live under the unbounded response time model.

Thus in terms of bounded fairness, the termination property is given as:

$$\forall x : \mathbb{N} \cdot \forall i : \mathbf{N} \cdot \forall A : \Phi \cdot \sigma\{\mathbf{i}\} \models_X \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_X \Diamond^{\tau} \neg\alpha$$

where $\Diamond^{\tau}$ is defined as a follows:

$$(\mathcal{M} \models_s \Diamond^{\tau} \beta) \text{ iff } (\mathcal{M} \models_s \beta) \vee (\mathcal{M} \models_s \bigcirc\beta) \vee \ldots \vee (\mathcal{M} \models_s \bigcirc^{\tau-1} \beta) \vee (\mathcal{M} \models_s \bigcirc^{\tau} \beta)$$

In previous work there have not been any bounds placed on the eventuality of an action. If an action may eventually occur at any time in the future (or the past) then we can deal with infinite sequences, simplifying the proofs required for progress and liveness. We maintain that this is not strong enough a justification for the separation of practice and theory. Any examinations of progress using an automated tool or manual analysis must include a set of guidelines based on bounded fairness.

When we compose progress properties and create system properties, we must have some method of determining the delay τ through the composite system. It is not feasible for an automated tool to examine all possible sequences of infinite length, just as it is not feasible for a user to wait for the end of time to get a response from a system.

The bounded delay can be determined by taking the delays incurred on the longest possible path through the system, the average path through the system, the average response time expected by a knowledgeable and reputable expert user of similar composite systems, and so on. Each method will have advantages and disadvantages that will make them more or less attractive for individual systems. As long as the delay is not short enough that the system can not be used in practical situations, the choice of the delay can be left up to the modeller and characteristics of the system that is being modelled.

We will continue to represent the behaviours of the components of a system with unbounded eventuality. When we assess the satisfiability of the basic progress properties for an actual component or system, we must replace all occurrence of $\Diamond$ within the basic progress properties with $\Diamond^{\tau}$ and then prove that the system behaviours continue to satisfy the bounded basic progress properties. We believe that the extra complexity introduced by the use of bounded eventuality will be more than offset by the practicality of this approach and its applicability to real-world systems.

8. Compatible Properties

There are characters which are continually creating
collisions and nodes for themselves in dramas
which nobody is prepared to act with them.
Their susceptibilities will clash against objects
that remain innocently quiet.
GEORGE ELIOT: Middlemarch

The order of the operations "putting your socks on"
and "putting your shoes on" is of practical importance.
JOHN ADDISON

In this chapter we introduce and discuss the compatibility of properties. The compatibility of policies and properties has been introduced in previous work by Hinton and Lee [HiLe94]. The ability to determine the compatibility of policies is of obvious importance when designing a system; we would like to ensure that the goals of the system are not contradictory. Knowledge of the conditions of use or types of goals that lead to incompatible policies will allow us to design useful systems with implementable policies.

We begin by defining the compatibility of properties and the assessment of the compatible property domain and compatible property specification. These indicate if the requirements and assumptions of the environment and the requirements on the behaviour of the system may be simultaneously satisfied. We define a method for assessing the compatibility of properties and examine the transitivity of the compatibility assessment.

The requirements for compatible basic safety and progress properties are discussed. We also discuss the phenomenon of order-of-application compatibility, which may result when mechanisms within a system are used to dynamically disallow behaviours that will violate the goals of a desired property.

8.1 Compatibility of Properties

Two properties are compatible if the goals of the two properties are not mutually exclusive. Properties that are compatible are simultaneously satisfied by the behaviour of a system.

Incompatibilities may not be apparent in the policy statements but will be detectable by analysing the formal properties. This will help indicate if policies become incompatible in the formal instantiation because of a discrepancy in the translation of policy to property.

8.2 Compatible Property Domain

The compatible property domain defines the common environment required by different properties. This environment may be a subset of the environments required by the individual properties, where there are events that are allowed by one property that are not allowed by another property.

Consider the general property domain,

$Q \equiv \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A : \mathcal{F}ma\{\Phi\}$

All properties are defined over all positions in the sequence, $\forall x : \mathbb{N}$. A property may be defined for a subset of the components in the system, $\exists i : \mathcal{N}$. If this is the case, we must examine the compatibility of this property with other properties that are required to be satisfied at that component. The compatibility of a property with properties that do not apply to the same component is irrelevant.

8.2.1 Compatibility of Property Domains

The compatible property domain Q, of two individual event domains, Q_i and Q_j is defined as follows:

$$Q = Q_i * Q_j$$
$$= (\forall x : \mathbb{N} \cdot \lambda_{i1}\, i : \mathcal{N}_i \cdot \lambda_{i2}\, A : \mathcal{F}ma\{\Phi\}) * (\forall x : \mathbb{N} \cdot \lambda_{j1} j : \mathcal{N}_j \cdot \lambda_{j2} B : \mathcal{F}ma\{\Phi\})$$
$$= (\forall x : \mathbb{N} \cdot \lambda_{i1}\, i : \mathcal{N}_i \cap \mathcal{N}_j \cdot {}_{j1} j : \mathcal{N}_i \cap \mathcal{N}_j \cdot \lambda_{i2}\, A, \lambda_{j2} B : \mathcal{F}ma\{\Phi_i \cap \Phi_j\})$$

Within the compatible domain, $\mathcal{N}_i \cap \mathcal{N}_j$ defines the components in the system to which the property applies. $\mathcal{F}ma\{\Phi_i \cap \Phi_j\}$ defines the event domain common to both properties.

The first step in the compatibility assessment of properties is to determine the compatible property domain. For example, given two domain-dependent properties, Q_i and Q_j,

$$Q_i = \forall x : \mathbb{N} \cdot \lambda_{i1}\, i : \mathcal{N}_i \cdot \lambda_{i2}\, A : \mathcal{F}ma\{\Phi(i(in))\}$$
$$Q_j = \forall x : \mathbb{N} \cdot \lambda_{j1}\, j : \mathcal{N}_j \cdot \lambda_{j2}\, B : \mathcal{F}ma\{\Phi(j(in))\}$$

If the properties P_i and P_j are compatible, they will be compatible given the property domains Q'_i and Q'_j as follows:

$Q'_i = \forall x : \mathbb{N} \cdot \lambda_{i1}\, i : \mathbb{N}_i \cap \mathbb{N}_j \cdot \lambda_{i2}\, A : \mathcal{F}ma\{\Phi(i(in),\Phi(j(in))\}$

$Q'_j = \forall x : \mathbb{N} \cdot \lambda_{j1} j : \mathbb{N}_i \cap \mathbb{N}_j \cdot \lambda_{j2}\, B : \mathcal{F}ma\{\Phi(i(in),\Phi(j(in))\}$

Thus if the system is defined to have environment specified by $\Phi(i(in)),\Phi(j(in))$ then the properties may be simultaneously satisfied. If the compatible property domain is defined over the empty set of events ($\varnothing$) then the two properties will not interfere with one another. They cannot place conflicting requirements on the behaviour of the system because they are defined for distinct sets of events. If the two properties Q_i and Q_j are defined as

$Q_i = \forall x : \mathbb{N} \cdot \lambda_{i1}\, i : \mathbb{N}_i \cdot \lambda_{i2}\, A : \mathcal{F}ma\{\Phi(in)\}$

$Q_j = \forall x : \mathbb{N} \cdot \lambda_{j1}\, j : \mathbb{N}_j \cdot \lambda_{j2}\, B : \mathcal{F}ma\{\Phi(out)\}$

then the common event domain is $\Phi(in,out)$. If $\Phi(in,out) = \varnothing$ then the properties will not interfere with each other; they are therefore compatible.

If the compatible event domain is defined such that, $Q \neq Q_i$ or $Q \neq Q_j$ then there are events that are possible according to one property but are not possible according to the other property. Thus it is possible that behaviours that satisfy one property will not satisfy the other property.

Properties that are incompatible because of the property domain can be made compatible if both properties can be shown to be valid properties, corresponding to their original policy goal, given the restricted compatible domain, and if the domain of the component in question can be shown to correspond to the compatible property domain.

8.3 Compatible Property Specification

The compatible property specification is defined as the logical conjunction of individual property specifications. If the compatible property specification is trivially false, then the individual properties are mutually exclusive: the truth of one property requires the falsity of the other property.

Consider the general property domain, S,

$S \equiv W \wedge R$

The compatible property specification, S, of two individual property specifications, S_i and S_j is defined as follows:

$S = S_i \wedge S_j = S_j = S_i$

$\quad = (\alpha_i \in \mathcal{F}ma\{W_i\} \wedge R_i) \wedge (\alpha_j \in \mathcal{F}ma\{W_j\} \wedge R_j)$

$\quad = (\alpha_i \in \mathcal{F}ma\{W_i\} \wedge \alpha_j \in \mathcal{F}ma\{W_j\}) \wedge (R_i \wedge R_j)$

The world $(W_i \wedge W_j)$ is the *required world*, defining the allowable events for the compatible property. The common required world, W, is defined as:

$W = (W_i \wedge W_j)$

Because the conjunction of logical formulae is equivalent to the intersection of sets [Lips81, MaPn92], the required world is equivalent to:

$W = (\alpha_i , \alpha_j \in \mathcal{F}ma\{W_i \cap W_j))$

Compatible properties must have a common required world, W.

If $W = W_i = W_j$ then both properties are defined over the same sets of allowable events for the system. If there is a common event domain ($Q \neq \emptyset$) and no common required world, ($W=\emptyset$), then the properties are not compatible. These properties are both applied to the same subset of possible events within the environment; $W=\emptyset$ implies that one property does not allow the allowable events of the other property.

Given the two required worlds, W_i and W_j,

$W_i = \Phi(i(A,in))$

$W_j = \Phi(j(A,in))$

the common required world is given as

$W = \Phi(i(A,in)), \Phi(j(A,in))$

If the common required world is such that $W \neq W_i$ and $W \neq W_j$ the different properties allow for different allowable events within the system. As with the compatible property domain assessment, properties that can be shown to be satisfied given the set of common allowable events, can be shown to be compatible.

The required behaviour, R, of two individual properties, R_i and R_j is defined as follows:

$R = (R_i \wedge R_j)$

If the behaviours are such that $(R_i \Rightarrow R_j)$ then the required behaviour R_j will be satisfied if the required behaviour R_i is satisfied. This indicates that R_i is a more specific requirement on the same form of behaviour than R_j. If $(R_i \wedge R_j) = \perp$ then the required behaviours replace conflicting requirements on the behaviour of the system and the corresponding properties are not compatible.

8.3.1 Compatibility of Property Specifications

To determine the compatibility of the property specification we take the conjunction of the two required behaviours, fix the antecedent to the same event values at the same time, and determine the effects, if any, on the consequence of the behaviours. If one of the properties is a static formula of the form $\sigma \models_x \alpha \wedge \sigma \models_x \beta$, then we must consider the effects with both α and β as the fixed antecedent value.

We may need to take advantage of the standard substitution theorem [BeMa92]:

Substitution Theorem:
If $\forall z[\beta(x/z)]$ arises from $\forall x\beta$ by alphabetic change, then these two formulae are logically equivalent.

The substitution theorem is used in the compatible property assessment to fix the variables of the property specifications to search for incompatibilities.

Suppose we have two properties, P_1 and P_2, that have already been shown to have a common, compatible property domain and required world:

$$P_1 : \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \mathcal{F}ma\{\Phi\} \cdot \exists \chi : \mathcal{F}ma\{\Phi\} \cdot \sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \chi$$

$$P_2 : \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \mathcal{F}ma\{\Phi\} \cdot \sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \alpha$$

We use the substitution theorem to define property P_2 over a distinct set of variables, using the substitution $\forall \beta \, [P_2 \, (\alpha/\beta)]$

$$P_2 : \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \beta : \mathcal{F}ma\{\Phi\} \cdot \sigma\{\mathbf{i}\} \models_x \beta \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \beta$$

The compatible property specification is therefore

$$P_1 \wedge P_2 = \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha, \beta : \mathcal{F}ma\{\Phi\} \cdot \exists \chi : \mathcal{F}ma\{\Phi\} \cdot$$
$$(\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \chi)) \wedge (\sigma\{\mathbf{i}\} \models_x \beta \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \beta)$$

We use the substitution theorem a second time, to fix the variables of P_2 to one of the variables of P_1, $\alpha = \beta$

$$= \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \mathcal{F}ma\{\Phi\} \cdot \exists \chi : \mathcal{F}ma\{\Phi\} \cdot$$
$$(\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \chi) \wedge (\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \alpha)$$

The resultant property is not trivially false when simultaneously applied at the same component. We again use the substitution theorem, to fix the variables of P_2 to one of the variables of P_1, $\chi = \beta$:

$$= \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \mathcal{F}ma\{\Phi\} \cdot \exists \chi : \mathcal{F}ma\{\Phi\} \cdot$$
$$(\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \chi) \wedge (\sigma\{\mathbf{i}\} \models_x \chi \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \chi)$$
$$= \forall x : \mathbb{N} \cdot \forall i : \mathbb{N} \cdot \forall \alpha : \mathcal{F}ma\{\Phi\} \cdot \exists \chi : \mathcal{F}ma\{\Phi\} \cdot$$
$$(\sigma\{\mathbf{i}\} \models_x \alpha \Rightarrow \sigma\{\mathbf{i}\} \models_x \Diamond \neg \chi)$$

Again, the resultant property is not trivially false when simultaneously applied at the same component. Therefore, the two properties P_1 and P_2 are compatible.

We use the substitution theorem to examine the restrictions that properties simultaneously place on events occurring at a given time in the sequence. If there are **n** variables specified in the compatible event domain then we must consider at most **n!** different substitutions, or mappings.

In the previous assessment, we examined the compatibility of two properties, fixing the values at a specific time in the sequence by setting $\alpha = \beta$. If the property is shown to be trivially false then it does not matter what the value of α (or β) is: the properties will be incompatible for all possible values.

8.4 Compatibility Assessment of Properties

The assessment of the compatibility of properties has three distinct parts: the assessment of the compatible property domain, the compatible required world and the evaluation of the properties applied simultaneously to a system. Figure 8.1 describes the process of evaluating the compatibility of two components.

If two individual properties have the same property domain, required world, and a non-trivial required behaviour, then they are compatible. Compatible properties may be treated as a single property instead of individual properties. We will often continue to treat these properties as individual properties, eschewing the more complex multi-goal compatible property.

Determine Compatible Property Domain $Q_i * Q_j$
    if $Q_i * Q_j \neq Q_i \neq Q_j$ **then**
        properties require incompatible domains
    else if $Q_i * Q_j = Q_i \neq Q_j$ **then**
        if $Q_j \Rightarrow Q_i$ **then**
            if P_j not satisfied given Q_i **then**
                properties require incompatible domains
            else
                properties may be compatible: continue
Determine Compatible Required World $W_i \wedge W_j$
    if $W_i \wedge W_j \neq W_i \neq W_j$ **then**
        properties place conflicting requirements / assumptions
        on meta-environment of component
    else if $W_i \wedge W_j = W_i \neq W_j$ **then**
        if $W_j \Rightarrow W_i$ **then**
            if P_j not satisfied given W_i **then**
                properties place conflicting requirements /
                assumptions on meta-environment of component
            else
                properties may be compatible: continue
        else if $W_i \wedge W_j = W_i = W_j$ **then**
            properties may be compatible: continue
Evaluate Compatible Properties: $(Q_i \cdot W_i \wedge R_i) \wedge (Q_j \cdot W_j \wedge R_j)$
    for all combinations of variables of P_i at P_j
        if $P_i \wedge P_j = \perp$ **then**
            properties not compatible: stop
    otherwise properties compatible

Figure 8.1: Compatibility Assessment of Properties P_i and P_j

8.4.1 Using Sub-Properties in the Compatibility Assessment

There are properties that specify requirements on "compound" sequences in a system. These properties can be separated into sub-properties. For example, the property P_i, defined below, can be separated into subproperties P_{i1} and P_{i2} as follows:

$$
\begin{aligned}
P_i &= \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A : \mathcal{Fma}\{\Phi\} \cdot \sigma(\mathcal{N}^{+}\mathcal{E};i) \models_x A \\
&= \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A_1, A_2 : \mathcal{Fma}\{\Phi\} \cdot \sigma(\mathcal{N};i) \models_x A_1 \wedge \sigma(\mathcal{E};i) \models_x A_2 \\
&= \{ \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A_1\ \mathcal{Fma}\{\Phi\} \cdot \sigma(\mathcal{N};i) \models_x A_1 \} \\
&\quad \wedge \{ \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A_2 : \mathcal{Fma}\{\Phi\} \cdot \sigma(\mathcal{E};i) \models_x A_2 \} \\
&= P_{i1} \wedge P_{i2}
\end{aligned}
$$

This property can be further divided with a subproperty for each component in the set of components, $\mathcal{N}$, if necessary.

Properties may also define different subsets of possible or allowable events over different subsequences in a system, given different characteristics of events. These properties are separated into sub-properties as follows:

$$
\begin{aligned}
P_i &= \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A : \mathcal{Fma}\{\Phi(\text{char1})\} \cdot \lambda B : \mathcal{Fma}\{\Phi(\text{char2})\} \cdot \\
&\qquad \sigma(\mathcal{N};i) \models_x A \wedge \sigma(\mathcal{E};i) \models_x B \\
&= \{ \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda A : \mathcal{Fma}\{\Phi(\text{char1})\} \cdot \sigma(\mathcal{N};i) \models_x A \} \\
&\quad \wedge \{ \forall x : \mathbb{N} \cdot \lambda i : \mathcal{N} \cdot \lambda B : \mathcal{Fma}\{\Phi(\text{char2})\} \cdot \sigma(\mathcal{E};i) \models_x B \} \\
&= P_{i1} \wedge P_{i2}
\end{aligned}
$$

Sub-properties of properties allow us to examine the compatibility of relevant aspects of a property. For example, if one property places a specification on the events received from the environment, we can exclude from the compatibility assessment the sub-property of a second property that applies only to events received from other components.

8.5 Transitivity of Compatibility Assessment

As with the composition of properties and systems, transitivity of the compatibility assessment will allow us to pair-wise examine a set of properties. If the compatibility assessment is not transitive then we must examine the compatibility of all possible properties simultaneously.

The compatibility of the required behaviour is transitive: $(R_i \wedge R_j) \wedge (R_j \wedge R_k) \Rightarrow (R_i \wedge R_k)$. This follows from the transitivity of conjunction.

The compatibility of the required world and the compatible property are not transitive. This follows because the intersection of sets is not transitive. For three properties, P_i, P_j, P_k, transitivity of the compatibility assessment would require $(Q = Q_i * Q_j) \wedge (Q = Q_j * Q_k) \Rightarrow (Q = Q_i * Q_k)$. If $Q_j \subset Q_i$ and $Q_j \subset Q_k$, it follows that $Q_i \cap Q_k$ may include events excluded by Q_j. The same follows for the required world, where $(W = W_i \cap W_j) \wedge (W = W_j \cap W_k) \not\Rightarrow (W = W_i \cap W_k)$.

There are conditions where the transitivity of the compatible property domain and required world will follow. If $(Q_i \cap Q_j) = (Q_j \cap Q_k) = (Q_i \cap Q_k)$ and $(W_i \cap W_j) = (W_j \cap W_k) = (W_i \cap W_k)$ then the transitivity of the compatibility assessment follows. Transitivity also follows (trivially) if $Q_i = Q_j = Q_k$ and $W_i = W_j = W_k$.

The compatibility assessment of the domain-independent properties exhibits transitivity because they are all quantified over all $\mathbb{N}$ and all Fma{Φ}. The compatibility assessment of the domain-dependent properties does not in general exhibit transitivity because these properties are quantified over different event domains and required worlds, such that $Q_i \neq Q_j \neq Q_k$ and $W_i \neq W_j \neq W_k$.

8.6 Compatibility of Basic Properties

The compatibility of the basic properties is given in Table 8.1. The details of the assessments are included in Appendix F. With the basic safety and progress properties it is the required domain that leads to incompatibilities of properties, not the requirements on the system behaviours.

The properties of confidentiality and integrity are compatible, given that each fulfils the requirements of the other in terms of the observation and modification events. The compatibility of different levels of optimism is of interest as it describes the relationships between these properties. If the property of pessimistic confidentiality (integrity) is satisfied by the behaviour of a system, then so will the properties of weakly environment optimistic, weakly component optimistic and strongly optimistic confidentiality (integrity).

The pessimistic property defines the most restrictive domain, in which there are no allowable unacceptable observation (modification) events. Thus we can safely assume that no user will send an unacceptable event to the system and no component will send an unacceptable event to a vulnerable component. Likewise, strongly optimistic integrity (confidentiality) implies that both weakly environment optimistic and weakly component optimistic integrity (confidentiality) apply, as strongly optimistic incorporates the assumptions of these two individual properties.

The properties of weakly environment optimistic and weakly component optimistic integrity (confidentiality) are not in general compatible with each other. That is, one requires that we trust the environment, and one requires that we trust the components. If we were able to trust both the environment and the components then strongly optimistic integrity (confidentiality) would apply. If we were unable to trust either, then pessimistic integrity (confidentiality) would apply.

The property of authenticity is compatible with the pessimistic and weakly component optimistic properties. It is not necessarily compatible with the weakly environment optimistic or

	PI	EOC	EOI	COC	COI	SOC	SOI	AU	AC	EP	TR	EO	FT	PV	UT
PC	◆	⇑	◆	⇑	◆	⇑	◆	√	✣	✣	✣	✣	◆	✣	✣
PI		◆	⇑	◆	⇑	◆	⇑	√	✣	✣	✣	✣	◆	✣	✣
EOC			◆	✗	◆	⇒	◆	◆	✣	✣	✣	✣	◆	✣	✣
EOI				◆	✗	◆	⇒	◆	✣	✣	✣	✣	◆	✣	✣
COC					◆	⇒	◆	√	✣	✣	✣	✣	◆	✣	✣
COI						◆	⇒	√	✣	✣	✣	✣	◆	✣	✣
SOC							◆	◆	✣	✣	✣	✣	◆	✣	✣
SOI								◆	✣	✣	✣	✣	◆	✣	✣
AU									✣	✣	✣	✣	◆	✣	✣
AC										√	√	√	√	√	√
EP											√	√	√	√	√
TR												√	√	√	√
EO													√	√	√
FT														√	√
PV															⇔

√ = Compatible

◆ = Compatible given common property domain

✣ = Compatible if domain-independent property satisfied over domain-dependent domain

⇑ = Row property implies column property if satisfied by behaviour (compatible)

⇒ = Column property implies row property if satisfied by behaviour (compatible)

⇔ = Dependent properties; if one fails, other will also fail

✗ = Not compatible

Table 8.1: Compatibility of Basic Safety and Progress Properties

strongly optimistic properties, both of which make assumptions about the behaviour of the output of the components.

The basic progress properties (except faithfulness) are compatible with the domain-dependent basic safety properties when defined over the property domain specified by the safety properties. The required behaviours themselves are compatible.

Faithfulness is compatible with the domain-dependent safety properties provided that faithfulness is defined over the property domain required by the safety property and that this domain does not disallow the input or output half of any of the component behaviour rules specified by faithfulness.

If faithfulness is satisfied, then so are accessibility and eventual output: FT $\Rightarrow$ EO and FT $\Rightarrow$ AC. The converse is not true: EO $\not\Rightarrow$ FT and AC $\not\Rightarrow$ FT because faithfulness places restrictions on the relationship between the input events (which cause the state of the component) and the output events.

The progress properties of provenance and utility are interdependent physical properties. If one fails, then the other will also fail. The usefulness in these two separate properties is in clearly identifying the separate goals of the two properties, and identifying the form of the failure. If PV $\Rightarrow$ UT but UT $\not\Rightarrow$ PV then there is a channel in the system that is being diverted from its intended destination. If UT $\Rightarrow$ PV but PV $\not\Rightarrow$ UT then there is an unknown channel acting as an input source within the system.

The conditions of the compatibility of the basic safety and progress properties need to be considered when we are evaluating the properties of a system. When we evaluate the satisfiability of these properties we must also ensure that the conditions for compatibility, as described above, are met. In particular, we must ensure that the domain-dependent properties do not place conflicting requirements on the behaviour of the system. We must also examine the compatibility of any system-specific properties to ensure that we have not inadvertently defined properties that conflict with each other or with the existing basic safety and progress properties.

8.7 Order-of-Application Compatibility

Properties may be incompatible because of potentially conflicting requirements on the environment or behaviour of the system. This incompatibility may manifest itself as a form of incompatibility resulting from the order in which properties are applied to a system.

It may be that a system will require minor modifications so that it will exhibit a given property, either to the system behaviour, allowable events or connections within the system. We must ensure that these modifications do not result in subtle changes to the system such that it no longer satisfies previously satisfied and previously assessed properties. These modifications introduce incompatibility because they modify the system and its behaviour so that other properties are no longer satisfied.

If a system is modified so that it satisfies a policy with a specific goal, we cannot then re-modify it to satisfy a more general policy goal without violating previously satisfied properties. This is an example of *order-of-application* compatibility. If a more general goal is satisfied and a more specific property is applied, we may reduce the scope of the general goal. This will not violate the existing general property.

9. Emergent Properties

The doubt of future foes exiles my present joy,
And with me warns to shun such snares as threten mine annoy.
For falsehood now doth flow, and subject faith doth ebb
Which would not be, if reason rules or wisdom weaved the web.
But clouds of toys untried do cloak aspiring minds,
Which turn to rain of late repent, by course of changed winds.
The top of hope supposed, the root of ruth will be,
And fruitless all their graffèd guiles, as shortly ye shall see.
QUEEN ELIZABETH I: The Doubt of Future Foes

In this section we introduce and discuss the emergence of properties. Emergent properties are those that are gained or lost when systems are composed. The idea of emergent properties was initially discussed by Bertalanffy [Bert68] with respect to biological systems. Nancy Leveson introduced the idea of emergent behaviours with respect to computer system behaviour when she observed that "Safety is [such] an emergent property" [Lev91]. Leveson defines emergent properties as "properties of a complex system that emerge or arise only when components are put together and are not visible or identifiable by considering only individual components" [Lev91].

When we interconnect systems, new behaviours become possible. These new behaviours are relevant to the composite system and can be said to emerge; they were not relevant or possible for any of the individual components. New behaviours may not be governed by the properties of the component systems, which in turn may not represent the overall goals of the composite system. Thus we need to define, identify, and prove properties that emerge for the composite system and its emergent behaviour. In this chapter we define the emergent properties, investigate how to identify them and prove their satisfiability for a given composite system.

9.1 Emergent Properties

Properties reflect the desired behaviour of a system; if a system is changed, its behaviour may change and its desired behaviour may also change. The composition of components results in a system with *emergent behaviour,* behaviour not possible for the individual components but exhibited by the interconnection of these components. Emergent properties reflect the desired goals of this emergent behaviour.

Emergent properties may be gained or lost when components are composed. A gain of properties occurs when an individual component that does not exhibit a certain property does exhibit that property when composed with another component (which may or may not exhibit that property). The loss of a property at a component (de-emergence) occurs when a component that does exhibit a certain property no longer exhibits that property when composed with other components (which may or may not exhibit the property). There is often an implicit assumption that the properties of a component will continue to be satisfied by that component when interconnected with other components. If this assumption is shown to be false, the properties in question are said to de-emerge.

There are many real-world examples of systems that exhibit emergent behaviours, described by emergent properties. For example, the logic gates *and*, *or*, *no*t have simple, easily expressed properties. These properties alone give little information about the possible behaviours of the possible interconnections of these gates. Without additional knowledge of how these logic gates are to be interconnected, we have no way of knowing, *a priori*, what properties or behaviours a composite system will satisfy. We cannot determine if the properties of a binary adder will emerge on the composition of these logic gates; if we have created a series of flip-flops, the composite system may have exhibited the properties of a shift register. The meta-stable behaviour of an interconnection of logic gates is another example of emergent behaviour that is not easily predictable without the detailed knowledge of the behaviour of the gates, their interconnections, and the behaviour of the interconnected components.

If we have defined all relevant properties then there can be no new ones. As we are not yet at the stage (and it is not likely that we ever will be) where we are able to define all relevant properties, or even identify that we have defined all relevant properties, it seems likely that new properties will always come into existence on composition.

9.2 Representation of Emergent Properties

Consider two components, C_i and C_j with the sets of properties, $\mathbf{P}(C_i)$ and $\mathbf{P}(C_j)$ respectively. The notation $\mathbf{P}(C_i)$, reads as "the set of properties of component C_i, where a property that is satisfied by the behaviour of C_i belongs to the set $\mathbf{P}(C_i)$

$\exists P_k \cdot \mathbf{sat}(C_i, P_k) \Rightarrow P_k \in \mathbf{P}(C_i)$

where $\mathbf{sat}(C_i, P_k) : C_i \times P_k \rightarrow \{\top,\perp\}$. There may be properties of a component C_i that belong to $\mathbf{P}(C_i)$ yet are not satisfied by the component's behaviour; it is common to require that such properties be satisfied. A composite system has the set of properties

$\mathbf{P}(\boldsymbol{\&}(C_i, C_j))$

where the properties may be composed as product, cascade, feedback, or some general composition of components, $\boldsymbol{\&} = \{\&^P, \&^C, \&^F, \&\}$. The conditions for emergent properties are given in Table 9.1.

	$P_k \in$			
	$\mathbf{P}(C_i)$	$\mathbf{P}(C_j)$	$\mathbf{P}(\&(C_i, C_j))$	Property Type
0	$\top$	$\top$	$\top$	Composable
1	$\top$	$\top$	$\perp$	Not Composable
2	$\top$	$\perp$	$\top$	Emergent (Type 1)
3	$\top$	$\perp$	$\perp$	Not Emergent
4	$\perp$	$\top$	$\top$	Emergent (Type 1)
5	$\perp$	$\top$	$\perp$	Not Emergent
6	$\perp$	$\perp$	$\top$	Emergent (Type 2)
7	$\perp$	$\perp$	$\perp$	Irrelevant

Table 9.1: Classes of Emergent Properties

If there is a property P_k common to the individual components and the composite system and satisfied by the behaviour of the individual components and the composite system,

$$\exists P_k : \mathbf{P}(C_i) \cap \mathbf{P}(C_j) \cap \mathbf{P}(\boldsymbol{\&}(C_i, C_j) \cdot$$
$$(\mathbf{sat}(C_i,P_k) \wedge \mathbf{sat}(C_j,P_k) \wedge \mathbf{sat}(\boldsymbol{\&}(C_i, C_j),P_k))$$

then the property P_k is *composable for this composite system*. This property may not be composable for different combinations of systems: the composability of this property for all possible combinations of components does not automatically follow from this specification. This corresponds to line 0 in Table 9.1.

If there is a property, P_k, such that

$$\exists P_k : \mathbf{P}(C_i) \cap \mathbf{P}(C_j) \cdot$$
$$(\mathbf{sat}(C_i,P_k) \wedge \mathbf{sat}(C_j,P_k) \wedge \neg\mathbf{sat}(\boldsymbol{\&}(C_i, C_j), P_k))$$

then the property P_k is *not composable* (line 1 in Table 10.1).

There are six other possible relations between properties of components and systems that must be considered. Line 7 describes properties that are not relevant to the individual systems nor the composite system. We do not need to consider such properties.

Lines 2 and 4 in Table 9.1 define a property P_k such that

$$\exists P_k : \mathbf{P}(C_i) \cap \mathbf{P}(\mathbf{\&}(C_i, C_j) \cdot$$
$$(\ \mathbf{sat}(C_i,P_k) \wedge \neg\mathbf{sat}(C_j,P_k) \wedge \mathbf{sat}(\mathbf{\&}(C_i, C_j), P_k)\ )$$

Property P_k is an emergent property. This property may or may not be relevant to C_j within the context of the composite system (as opposed to C_j as an individual, stand-alone system). This property is defined to be a *type 1 emergent property*.

Line 6 in Table 9.1 defines a property P_k is such that

$$\exists P_k : \mathbf{P}(\mathbf{\&}(C_i, C_j) \cdot$$
$$(\ \neg\mathbf{sat}(C_i,P_k) \wedge \neg\mathbf{sat}(C_j,P_k) \wedge \mathbf{sat}(\mathbf{\&}(C_i, C_j), P_k)\ )$$

Property P_k is an emergent property. P_k does not apply to any of the individual components of the system but does apply to the behaviour of the composite system. P_k is defined to be a *type 2 emergent property*.

The final relation between properties (lines 3 and 5 of Table 9.1) defines the property P_k

$$\exists P_k : \mathbf{P}(C_i) \cdot$$
$$(\ \mathbf{sat}(C_i,P_k) \wedge \neg\mathbf{sat}(C_j,P_k) \wedge \neg\mathbf{sat}(\mathbf{\&}(C_i, C_j),P_k)\ )$$

where the property P_k is applicable to the behaviour of one (or more) but not all of the components and is not applicable to the behaviour of the other components in the system nor to the behaviour of the composite system.

In the above discussion it has been assumed that the properties of C_i and C_j continue to be satisfied by the components within the composite system.

9.3 Types of Emergent Properties

Type 1 emergent properties are those that are known properties satisfied by individual components. Given the properties of the individual components, we must decide which are applicable to the composite system and ensure that they are satisfied by the composite system. If these properties are also required to be satisfied by (some or all of) the components within the composite system, we must assess their satisfiability at these components.

A type 2 emergent property, like a type 1 emergent property, is gained on composition. Unlike a type 1 emergent property, it is not satisfied by any of the individual components. As with type 1 properties, these emergent properties may be required to be satisfied by the components within the context of the composite system. If so, we must assess the satisfiability of these properties at the relevant components. The recognition of previously identified properties satisfied on composition, while tedious, is simply a matter of determining which, if any, of the previously identified properties are satisfied by the composite system behaviour.

The difficulty with type 2 emergent properties is in the identification and recognition of *theoretically relevant but previously unknown properties*. These properties are referred to as *pedantic properties*[5]. These properties formally specify the goals of the desired behaviour for a composite system. They ensure that no previously unconsidered type of behaviour will violate these goals. For example, before 1981 (approximately), non-interference [GoMe82, GoMe84] was a pedantic property. Pedantic properties that do not affect the desired behaviour of the system can be ignored.

Known relevant properties have been designed (some by trial and error and some through bitter experience) to deal with known problems. If we are not aware of a potential problem then we can not identify it, nor can we identify any sequences that would violate this problematic behaviour, nor can we identify any properties that would describe only the allowed behaviours. Therefore, the key to the identification of pedantic properties is a knowledge and understanding of the threats, risks and vulnerabilities of a system, and those introduced through composition.

9.4 De-Emergent Properties

While new properties and behaviour may emerge on composition, we also need to ensure that desired properties do not de-emerge. A property is said to de-emerge if it is no longer satisfied by a component within the context of a given composite system. De-emergent properties are not composable, although properties that are not composable need not be de-emergent. Composability of properties will be discussed in detail in Chapter 10. De-emergent properties are of interest because they indicate that components may lose their functionality or security features when connected with other components.

Within the context of emergent properties, a composable property is one that is satisfied by two components, by the interconnection of the two components, and continues to be satisfied by the individual components within the composite system. A de-emergent property is one that is not satisfied by the interconnection of the components, and is no longer satisfied by at least one of the individual components. A non-composable property, by contrast, will not be satisfied by the interconnection of the components but will continue to be satisfied by the individual components within the context of the composite system. Therefore we must take care to consider the composability, non-composability and possible de-emergence of a property in a composite system.

---

5 **pedant n.** 1 a person who insists on strict adherence to formal rules or literal meaning at the expense of the wider view. 2 a person who rates academic learning or technical knowledge above everything. 3 a person who is obsessed by a theory; a doctrinaire. (p. 977, The Concise Oxford Dictionary of Current English, Eighth Edition, 1990)

9.5 Assessment of Emergent Properties

An informal algorithm to determine emergent properties for a composite system, $\mathfrak{C}$, with two components, C_i, C_j, is given in Figures 9.1 and 9.2. There are several Boolean relations that are used in this algorithm. They represent the characteristics of the composability of a property and relevance of a property to the required goals of a component's behaviour. These are defined as:

composable (P_i)	$: P_i \rightarrow \{\top,\perp\}$	The property P_i is composable
relevant (C_i,P_i)	$: C_i \times P_i \rightarrow \{\top,\perp\}$	The property P_i is relevant to the behaviour of component C_i

We also define the following notational constructs:

$\mathfrak{C}(C_i)$	:	Component C_i within composite system $\mathfrak{C}$
$\bigcup_{i=1}^{\mathfrak{C}} \mathbf{P}(C_i)$	:	Set of all properties for components in $\mathfrak{C}$
$\bigcap_{i=1}^{\mathfrak{C}} \mathbf{P}(C_i)$	:	Set of properties common to components in $\mathfrak{C}$
$\mathbf{P}(\mathfrak{C}) \cup= P_k$	$:\mathbf{P}(\mathfrak{C}) = \mathbf{P}(\mathfrak{C}) \cup P_k$	Set of properties $\mathbf{P}(\mathfrak{C})$ union property P_k
$\mathbf{G}(\mathfrak{C})$	:	Goals of behaviour of composite system $\mathfrak{C}$ represented as a set of properties

We adopt the following approach to the determination of emergent properties:

Part I: Examine all (known and relevant) properties of components of composite system. Ensure that emergent properties are satisfied by the relevant components (are not de-emergent). Ensure that all type 1 emergent properties are satisfied by the composite system.

Part II: Represent the desired system behaviour as properties of composite system. If the goal is entailed by an existing system property, it will have been examined in Part I. If the goal is entailed by a known property of a similar system, assess this (type 2) property for the composite system. Otherwise, the goal represents a pedantic property.

The properties that are considered in Figure 9.1 are all known, defined properties. This algorithm defines a concise method for assessing the role of these properties, including non-composable properties, as potential emergent properties. Figure 9.2 describes the process of identifying and assessing the type 2 emergent properties, those properties that are not also properties of the individual components of the system.

Once a pedantic property is identified, it is no longer, by definition, a pedantic property. For the purposes of this work (and the sanity of the system analyst) we will classify such a property as pedantic until after we have completed our analysis, at which point it becomes a known property of similar systems.

```
PART I: COMPOSABLE + TYPE 1 EMERGENT PROPERTIES
∀P_k : ∪_{i=1}^{ℭ} P(C_i)
    if P_k ∈ ∩_{i=1}^{ℭ} P(C_i) then
        property common to all components
        if ¬composable(P_k) then
            if relevant(ℭ,P_k) then
                P(ℭ) ∪= P_k          add to properties of composite system
                if ¬sat(ℭ,P_k) then
                    required property not satisfied by composite system
                    add to list of not-satisfied, required properties
                else required property satisfied: okay
            else not relevant to composite system: okay
            ∀C_i : ℭ
                if ¬sat(ℭ(C_i),P_k) then
                    required property de-emergent at component C_i
                    within composite system
                    add to list of not-satisfied, required properties
                else required property continues to be satisfied: okay
        else required property composable: okay
    else property not common or relevant to all components
        ∀C_i : ℭ
            if P_k ∈ P(C_i) then
                if ¬sat(ℭ(C_i),P_k) then
                    required property de-emergent at component C_i
                    within composite system
                    add to list of not-satisfied, required properties
                else
                    required property satisfied: okay
            else if relevant(ℭ(C_i),P_k) then
                if ¬sat(ℭ(C_i),P_k) then
                    property relevant to component within system,
                    yet property not satisfied by component within system
                    add to list of not-satisfied, required properties
        if relevant(ℭ,P_k) then
            P(ℭ) ∪= P_k              add to properties of composite system
            if ¬sat(ℭ,P_k) then
                type 1 emergent property not satisfied by system
                add to list of not-satisfied, required properties
```

Figure 9.1: Assessment of Composable and Type 1 Emergent Properties

PART II: TYPE 2 AND PEDANTIC PROPERTIES

Represent overall goals as properties of composite system: $\mathbf{G}(\mathbb{C})$

$\forall P_k : \mathbf{G}(\mathbb{C})$

    if $P_k \in \mathbf{P}(\mathbb{C})$ **then**

        P_k entailed by an existing property: examined in Part I.

    else

        if $P_k \in \mathbf{P}$(similar systems) **then**

            P_k entailed by a known property

            $\mathbf{P}(\mathbb{C}) \cup= P_k$ add to properties of composite system

            if $\neg\mathbf{sat}(\mathbb{C}, P_k)$ **then**

                type 2 emergent property not satisfied by system
                add to list of not-satisfied, required properties

        else

            $\mathbf{P}(\mathbb{C}) \cup= P_k$ add to properties of composite system

            if $\neg\mathbf{sat}(\mathbb{C}, P_k)$ **then**

                pedantic property not satisfied by composite system
                add to list of not-satisfied, required properties

        $\forall C_i : \mathbb{C}$

        if *relevant*$(\mathbb{C}(C_i), P_k)$ **then**

            relevant to components within system

            if $\neg\mathbf{sat}(\mathbb{C}(C_i), P_k)$ **then**

                type 2 emergent property not satisfied by components
                add to list of not-satisfied, required properties

Figure 9.2: Assessment of Type 2 Emergent Properties

9.6 Composability and Emergence

If we compose a component that satisfies a known property with a component that does not satisfy this property, resulting in a composite system that does satisfy this property, then this property exhibits the characteristics of an emergent property.

Consider a system created by cascading a counter and one of two modulo counters: a modulo-2 and a modulo-3 counter. We use a clock to supply the inputs to the system. Each counter on its own has the property that it will produce an output on every (second) (third) clock tick.

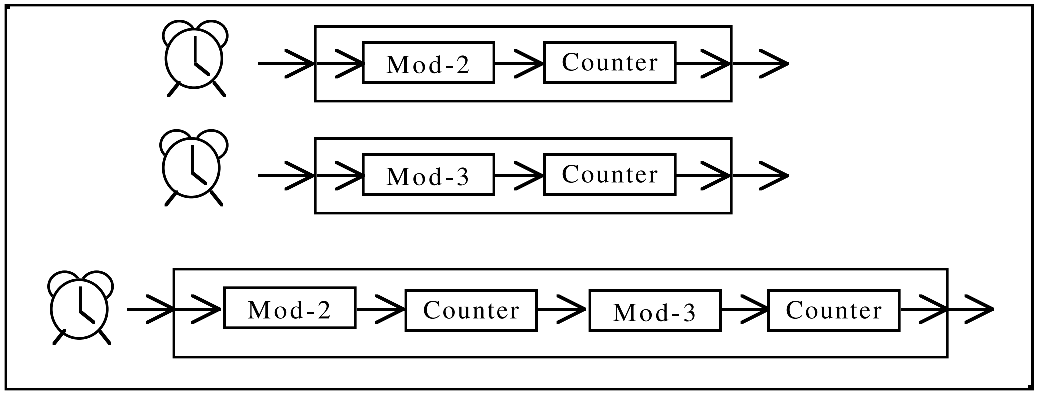

Figure 9.3: Composition of Clocks and Modulo Counters

The interconnected systems are shown in Figure 9.3.

The modulo-2 counter has the property that the output is a 0 if $n \bmod 2 \neq 0$, where n is the number of clock ticks received, and 1 otherwise. Thus if the modulo-2 counter is connected to a clock and a counter, the resulting system will "count" every second clock tick. A similar system with the modulo-3 counter has a similar property, "counting" every third clock tick.

If the modulo-counters are connected in cascade, the resultant system will not exhibit either of the individual properties; these properties are not composable. Instead, a new property emerges for the composite system. Using a clock to supply an input to the composite system, we find that the composite system will count *one of every six* clock ticks. This emergent property is independent of the order in which we cascade the two modulo-counters.

Let C_1 represent the (modulo-1) counter, where $\mathbf{P}(C_1) = \{ P_1 \}$

Let C_2 represent the modulo-2 counter, where $\mathbf{P}(C_2) = \{ P_2 \}$

Let C_3 represent the modulo-3 counter, where $\mathbf{P}(C_3) = \{ P_3 \}$

Given the properties P_i (and behaviours, B_i) of these components

$$P_1 = B_1 = \forall x : \mathbb{N} \cdot (\sigma\{\mathbf{i}\}(in) \models_x tick \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x tick)$$

$$P_2 = B_2 = \forall x : \mathbb{N} \cdot ((\sigma\{\mathbf{i}\}(in) \models_x tick \wedge x \bmod 2) \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x tick)$$

$$P_3 = B_3 = \forall x : \mathbb{N} \cdot ((\sigma\{\mathbf{i}\}(in) \models_x tick \wedge x \bmod 3) \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x tick)$$

The assessment of properties of the composition of the mod-2 and mod-3 counters according to the algorithms given in Figures 9.1 and 9.2 easily determines that the property P_6 emerges on composition.

$$P_6 \equiv \forall x : \mathbb{N} \cdot (\sigma\{\mathbf{i}\}(in) \models_x tick \wedge x \bmod 6) \Rightarrow \sigma\{\mathbf{i}\}(out) \models_x tick)$$

Neither P_1, P_2 nor P_3 is a type 1 emergent property for the composite system. There are no composable or type 1 emergent properties of the composite system. If we compare property P_6 with the behaviour of the composite system, it is easy to see that the actual behaviour of the system represents the desired behaviour of the system. Thus the property of this composite system is an (easily identified) pedantic property.

9.7 Compatibility and Emergence

Compatible properties may act as emergent properties. Consider the composition of two components, satisfying different strengths (levels of optimism) of the same property. A simple compatibility assessment of these two properties [HiLe94] indicates that the less optimistic property is common to both systems. On composition, the more optimistic (stronger) property may no longer be satisfied by the individual component, nor by the composite system. Thus the stronger property is a de-emergent property; the contradiction of the stronger property acts as an emergent property for the composite system. This agrees with an observation made by John McLean that "a [possibilistic] property seems to be preserved by being cascaded [composed] with itself or with a stronger property." [McL94]

10. Composition of Properties

At night there came into the hostelry
Some nine and twenty in a company
Of sundry folk happening them to fall
In fellowship, and they were pilgrims all
That towards Canterbury meant to ride
GEOFFREY CHAUCER: The Canterbury Tales
(tr. Nevil Coghill)

In this chapter we discuss the composition of properties describing the desired behaviour of individual and composite systems. The goal of, and difficulty with, composition is to determine before a system is composed if the properties of the components will also be properties of the composite system. At present this is at best a difficult task; for many compositions it is not practical. Composition is of interest because it allows us to evaluate properties for individual components and then compose large, complex systems from these components such that the properties will be satisfied by the complex system. Knowledge of the composability of properties obviates the need to re-evaluate properties for a complex system; the properties are known from construction.

In this chapter we examine the composability of the basic safety and progress properties and the threats to their satisfiability for a composite system. We propose a method for assessing the satisfiability of non-composable properties. We conclude with a Composition Principle. This Composition Principle is a circular set of rules used to assess the composability and satisfiability of properties for a composite system.

10.1 Composition of Properties

The basic safety and progress properties must be satisfied by the individual components *and* by the interconnection of the components, the composite system. Properties that apply to both components and systems, be they the basic properties proposed in this work, possibilistic security properties, or other system-specific properties, must be satisfied by all relevant components.

A property is *composable* if it is satisfied by the behaviour of the general composition structure of the feedback, cascade or product composition of components. *Non-composable* properties are not satisfied by the behaviour of feedback, cascade or product compositions. A property that is not composable may be satisfied for a composite system. However, such a property will not be known, *a priori*, to be satisfied by the composite system.

The composability of a property is dependent on the type of property (domain-dependent or independent), its meta-properties (transitive, closed, *et cetera*), and the type of composition (product, cascade, feedback). Properties that are transitive and closed for a given construct will be composable for compositions using this construct. If a property is reflexive, then the property is satisfied by the component and the composition of the component with a self-feedback loop. Reflexivity of a property is independent of structure. If a property is symmetric, it will be composable for a system with feedback. Symmetry of properties for feedback systems follows by the definition of a feedback composite structure. Domain-dependent properties are not composable if the individual components do not have identical environments and environment assumptions.

If a property is composable for a desired composition construct, then this property is known to be satisfied by the behaviour of the composite systems with this structure. If a property is not composable for the given construct and is relevant to the composite system, we must assess the satisfiability of this property given the behaviour of the composite system.

10.2 Composition of Basic Safety Properties

The basic safety properties are not composable: they are not satisfied by the behaviour of a product, cascade, or feedback composition of components satisfying these properties. This follows because they are domain-dependent, and is verified by the meta-properties of the basic safety properties. Table 10.1 lists the basic safety properties and threats to their satisfiability within a system. Knowledge of the potential threats to a system will help us to design and interconnect systems such that they are not susceptible to such threats and to identify such threats within a composite system.

Property	**Threats to Property on Composition**

Confidentiality / Integrity	Events that are unacceptable and previously not possible may be received by a vulnerable component from either the environment or from another component.
P.C. / I.	All potentially harmful events from either the environment or from system components must be disallowed.
EOC. / I.	Potentially harmful events that may be received from other components must be disallowed. Such events are assumed not to be received from the environment.
COC. / I.	Potentially harmful events that may be received from the environment must be disallowed. Such events are assumed not to be received from the system components.
SOC. / I.	Potentially harmful events are assumed not to be received from the system components or from the environment.
Authenticity	Component may produce unknown outputs in response to unknown inputs that become possible on interconnection.

Table 10.1: Threats to and Remedies for Safety on Composition

10.2.1 Assessment of Safety Properties

The composite and meta-environment of a system and its components must correspond with the domain required by the domain-dependent properties. Figure 10.1 defines the process of evaluating the domain of a composite system with respect to the confidentiality and integrity properties, presented in 2-system form for simplicity. This process is easily extended to deal with more than 2 systems.

This process defines the conditions under which the various confidentiality and integrity properties are not satisfied. If an event common to both components is acceptable to one component and unacceptable to the other component, the pessimistic property cannot be satisfied. If the connections within the system are such that this unacceptable event will be sent to one component by the other component, the component optimistic property is not satisfied. If this unacceptable event may be received from the environment, and the environment cannot be trusted to not send this event, then the environment optimistic property is not satisfied. If both the component optimistic and environment optimistic properties are satisfied then the strongly optimistic property will be satisfied.

It is possible that none of the confidentiality or integrity properties will be satisfied given the composite system's environment. We may decide to *configure* the system and its environment to satisfy one of the properties. We may decide to place more trust in the users of a

Assume that we are given two systems, i and j.
The composite environment of the two systems is $\Phi^i \cup \Phi^j$.
$\forall\ \alpha \in \{\Phi^i \cup \Phi^j\}$

    if $\{\ \exists\alpha \in \Phi(i(A),obs) \cap \Phi(j(U),obs)\ \}$
        then PC = FALSE
    else if $\{\ \exists\alpha \in \Phi(i(U,obs) \cap \Phi(j(A,obs)\ \}$
        then PC = FALSE
    else if $\{\ \exists\alpha \in \Phi(i(A,out),obs) \cap \Phi(j(U,in),obs)\ \}$
    and $(output_i \rightarrow input_j)$ **and** $\neg\sigma(i;\not E) \models_x \alpha$
        then COC = FALSE
    else if $\{\ \exists\alpha \in \Phi(i(U,in),obs) \cap \Phi(j(A,out),obs)\ \}$
    and $(output_j \rightarrow input_i)$ **and** $\neg\sigma(j;\not E) \models_x \alpha$
        then COC = FALSE
    else if $\{\ \exists\alpha \in \Phi(i(A,obs) \cap \Phi(j(U,in),obs)\ \}$
    and $(environs \rightarrow input_j)$ **and** {agents not trustworthy)
        then EOC = FALSE
    else if $\{\ \exists\alpha \in \Phi(i(U,in),obs) \cap \Phi(j(A,obs)\ \}$
    and $(environs \rightarrow input_i)$ **and** {agents not trustworthy)
        then EOC = FALSE
    else if {EOC} **and** {COC}
        then SOC = TRUE
    else
        Confidentiality not satisfied at any level of optimism

The assessment of integrity is the {obs/mod} dual.

Figure 10.1: Assessment of Composite System Environment

system not to send harmful events to vulnerable components. This may allow the system to satisfy a weakly environment optimistic property.

If we require either the pessimistic or component optimistic property to be satisfied, we must find a way to remove or nullify any potentially harmful events received from other components. We may choose to disallow a given connection between components, changing the structure of the composite system. We may choose to disallow events that are produced as outputs but are unacceptable as inputs. To do this, we redefine the given output, and the inputs that cause it, as unacceptable events. This may have a cascade-type effect in that the newly disallowed input may in turn have been received as an acceptable input from another component in the system, where it is now defined as unacceptable. We must cascade this effect back to the "original" event received from the environment, at which point we either disallow the event or trust the users not to send the event.

```
check_events( Φ, Φ*, i, j, assume )
    if Φ(i(out),mod) ∩ Φ(j(U,in),mod) = ∅ then
        no common {obs,mod} events
        print "Disallow Connection out_i → in_j"
        return
    ∀α ∈ Φ(i(out),mod) ∩ Φ(j(U,in),mod)
        if assume = trust component then
            okay to allow event as output of i
        else if assume = remove event then
            Φ*(i(out),mod) = Φ(i(out),mod) \ α
            Φ*(i(A,in),mod) = Φ(i(A,in),mod) \ β
            given i(α;β)
        else assume = disallow connection
            print "Disallow Connection out_i → in_j"
            return
    return
Φ = ∪_{i=0}^{N} Φ^i
Product Composition:
∀j : N · (environs → in_j)
    check_events(Φ, Φ*, environs, j, assume )
Cascade and Feedback Composition:
∀j : N · from output-level to input-level
    ∀i : N · (out_i → in_j
        check_events(Φ, Φ*, i, j )
    check_events(Φ, Φ*, environs, j, assume )
    Φ(i) = Φ*(i)
The assessment of confidentiality is the {obs,mod} dual.
```

Figure 10.2: Domain Modification for Composite System Satisfiability

If we introduce a mechanism to remove events from a component's input stream, we must take care not to violate progress. For example, we cannot remove an event "close file" without affecting the behaviour associated with "open file".

The process in Figure 10.2 may be used to remove potentially harmful events from the system. The module check_events is used to determine the resultant domain given the assumptions about the environment and other components. These assumptions are represented as the assumption to trust the component (where the environment may be represented as a component), the decision to remove the event, or the decision to disallow the connection. The process then produces as an output a listing of all offending events and the suggested action, based on the assumptions made.

The order in which we examine the components of a product structured composite system is irrelevant. The components do not communicate with each other and therefore their events cannot affect each other. We must however simultaneously consider both components when we decide to disallow an event that is possible from the environment, as such an event may be possible at both components.

The order in which we examine the components of a cascade structured system is critical: before we examine any component we must ensure that we have already examined all components that are reachable from this component. Thus we begin with components that send events only to the environment (there must be at least one such component in a cascade system). This is to prevent an event that has already been evaluated as allowable at a different component from being disallowed within the system, an action that may have repercussions for the previously assessed component. We use the same process for a feedback structure.

The complexity of the process given in Figure 10.2 depends on the composite structure. For a product structure composite system with $\mathbf{N}$ components, there will be at most $\mathbf{N}+1$ calls to check_events. For a cascade composite system structure, there will be at most $\frac{\mathbf{N}(\mathbf{N}+1)}{2}$ calls to check_events. Thus for the composite system structures of product and cascade, the complexity of the process is $O(\mathbf{N})$ and $O(\mathbf{N}^2)$, respectively, assuming that the valuation of check_events is constant.

For a feedback system, the complexity of the process given above will also be $O(\mathbf{N}^2)$. The algorithm given, however, will not remove all events $\alpha \in \Phi(i(out),\{obs,mod\}) \cap \Phi(j(U,in),\{obs,mod\})$ for a feedback composite system. This is because a component may be reachable from itself. The assessment of acceptable events given feedback loops in the system may have the effect of "rendering unacceptable" events at components that have already been considered to be acceptable. Thus we must use iterations of the cascade/feedback process to remove all potentially harmful events from a feedback system, stopping when there are only acceptable events remaining, the system has been disallowed (all connections removed), there are no remaining allowable events or some user defined limit has been reached. This results in a potentially unbounded algorithm with an unbounded complexity.

10.3 Failure of Progress

The failure of progress in a composite system (with or without feedback) results in livelock, where a component no longer exhibits progress, or deadlock, where no component may progress because each is dependent on another, non-progressing component. Such failures can be attributed to one of two sources: resource deadlock caused by resource contention, or communication deadlock caused by the lack of a required input. Resource contention occurs when a shared resource cannot be accessed. If the resource in question is never made available, any process that depends on this resource will not be able to gain this resource and will therefore not be able to progress.

Communication deadlock may occur when an input, required to terminate a process or initiate a new process, is not received, either because it was never sent or because a component is not in a state where it is able to receive the event. Thus the lack of a required input can also prevent a component from satisfying progress.

Resource contention and lack of input may simultaneously contribute to a lack of progress. For example, a process holding a resource may be unable to release that resource until a required input, authorising the resource's release, is received. Resource contention may be independent of lack of input, for example, if a required self-terminating process fails to terminate, thereby failing to release a resource. Lack of input may be independent of resource contention, for example, if a process enters a state from which it may act on only one required input event that it never receives.

If a component does not satisfy progress then it cannot be included in a composite system that is intended to satisfy progress.

10.3.1 Resource Contention and Progress

Resource contention ultimately hinges on the termination of a process owning a resource. If this process does not terminate, then the resource is not released and resource contention may lead to a failure of progress. If a process is self-terminating, we know from the basic progress properties that such a process will be self-terminating within the composite system. If a process is not self-terminating then resource contention may occur. We choose to classify this as a communication deadlock because the resource contention is released by a required input.

If resource contention occurs because there are not enough resources for a process to complete, even if it owns all possible resources, then the system is inherently flawed and will not progress. In our examination of progress we assume that there are sufficient resources available within the system; the resources available to a component are sufficient for any processes at a component to complete.

10.3.2 Other Sources of Progress Failure

Progress may fail due to the failure of a component. The issue of component failure and its effects on progress encompasses the area of fault-tolerant systems, where redundancy of components is used to ensure that the failure of a component will not affect the behaviour of a system. Fault-tolerant considerations such as these apply to systems that have already been composed. It is possible for a fault-tolerant system with triple redundancy to exhibit deadlock. Alternatively, failure of progress due to communication deadlock may not be viewed as a component failure, as the component will continue to have the ability to progress, provided that the required input is received. This examination of the failure of progress on composition is distinct from the issues of failure of progress within a fault-tolerant system.

10.4 Composition of Basic Progress Properties

The progress properties are composable provided that the behaviours of the component are input-state independent. If all behaviours are input-state independent, then progress is composable. For product and cascade systems, progress can be shown to be composable given the progress-restoring behaviour of the environment. For a feedback system with input-state dependent behaviours, progress may not be composable. Table 10.2 lists the general form to the threats to progress on composition.

If the behaviour axiom is input-state independent then the behaviours governed by this axiom will not stall progress. If the behaviour axiom is not input-state independent and the component never returns to the required input-state then the component is livelocked. If a series of components that comprise a feedback path through a system are all livelocked, then the system will exhibit deadlock. We must determine the input required to return the state of each component to the required input-state and ensure that this input remains a possible input.

Progress	**Threats**
Accessibility	Process cannot be initiated, because component not in correct state to accept input or because resources occupied.
Eventual Output	Process is unable to produce an output, because unable to execute (stalled waiting for input or waiting for resource).
Termination	Process is unable to complete due to resource contention or stuck-in-state waiting for a particular input.
Eventual Progress	Process completes and leaves component in a state where no more processes may be initiated. Process unable to complete, but owns all resources while waiting to complete, preventing other processes from executing.
Faithfulness	Fails if AC, EO, TR, EO fail, or unknown event (with unpredictable output) received.
Provenance	Physical threats such as severing or splicing connecting cables, adding new input channel.
Utility	Physical threats such as severing or splicing connecting cables, redirecting or adding output channels.

Table 10.2: Threats to and Remedies for Progress on Composition

10.4.1 Assessment of Progress

We must examine the behaviour axioms of each component and determine for which inputs, if any, progress may stall, and the circumstances under which this will occur. Figure 10.3 gives a general search technique to locate potential progress violating behaviours within the system.

If a component is designed to receive inputs only from the environment, then progress will be satisfied by all behaviour axioms, input-state dependent or independent. This may require restrictions on the behaviour of the environment, for example, by requiring a certain ordering of events received from the environment. Progress will nevertheless follow because the environment is always able to send a progress-enabling event to the component should the component stall.

Consider a component that is able to receive inputs from the environment and other components. Progress may not be satisfied by all possible inputs to such a component. This is because the interconnection of components may render impossible some progress-enabling inputs. For each behaviour axiom we must list all potential progress-enabling inputs to be examined within the context of the composite system.

If progress fails for a product system, it can be "restarted" by the environment. Recall that the only source (and destination) for events in a product system is the environment. If progress fails because of lack-of-a-required-input, it is because the unpredictable environment has not sent the required input event. Because the environment always has the ability to send such events, however, we can assume that the environment will always eventually send the required inputs, once it becomes clear that the system has *stalled* (temporarily ceased to progress).

If progress fails for a cascade system, a required input expected from the environment or from another component is not received. If a required input may be received directly from the environment, then as with the product structure, we assume that the environment will send the required input and restore progress. If the required input is received from a component, then this previous component is unable to produce, as an output, the input required by the component in question. With a cascade system, all components are finitely reachable from the environment. Thus we can ultimately cause the required progress-restoring event for any component in the system from the environment.

Such an event will pass through at most ($\mathbf{N}$-1) components before it is received at a given component in an $\mathbf{N}$ component system. If all components satisfy the basic progress properties then no component can cause another component to never progress without the collusion of the agents in the environment.

As with a cascade system, if progress fails for a component within a feedback system, we should be able to correct this with an event from the environment which will eventually be received by the component in question. Unlike the cascade system, however, such an event may pass through any number of components before it is received at the component in question. Thus we must examine the behaviour of the individual components and assess their potential interactions with other components before we interconnect them to create a composite system.

$\forall i : \mathbb{N}$

  $\forall$b.a.$_j$: { input-state dependent behaviour axioms b.a.$_j$ }

    $\forall \alpha : \{\Phi(i(\text{input})\}$

      if α is not possible input to component when in input-state of b.a.$_j$

        then progress will not fail because of α

        continue

      if input-state of b.a.$_j$ $\neq$ input-state required for α

        then progress may fail if input required to return component to required α input-state is not received.

      $\forall \beta : \{\Phi(i(\text{input})\}$

        if β will return component to α input-state

          then progress may stall at component

            system may deadlock if β not received:

            determine when and if β is not received.

Figure 10.3: Detecting Potentially Stalled Paths in Composite System

Failure of progress in systems with feedback is the most difficult to detect of the three composite structures. There has been a great deal of research in deadlock detection algorithms for systems with feedback (see [Singh89] for a survey of some of these algorithms). This work does not propose yet-another-deadlock-detection-algorithm. Instead, we examine why and when deadlock, or lack-of-progress, occurs within a composite system so that we can predict where progress may fail in a composite system. Any of the standard distributed deadlock-detection algorithms may be used to determine if deadlock is possible [CMH83, Gold77, IsMa78, MiMe84, Sing89].

These guidelines do assume that we have a detailed knowledge of the design of the system and its intended operation. We contend that this is only realistic to expect: we cannot determine if a system will be free from deadlock if we cannot tell from the system operation if progress is being violated or not.

10.5 Composability of Compatible Properties

The basic safety properties are not, in general, composable, and the progress properties are, in general, composable for product and cascade systems and for feedback systems given input-state independent behaviours. The compatible-versions of the basic safety and progress properties may not be composable. The restricted domain required by all properties (corresponding to the physical domain of the system) may result in events that are required to satisfy the basic properties no longer being possible events within the composite, compatible system. The analysis of the composability of the compatible-version of properties is the same as the process discussed in the preceding sections.

10.6 Composability of System-Specific Properties

The composability of system-specific properties is dependent upon their meta-properties, domain-dependence and the composite structure in question. The composability of system-specific safety and progress properties requires the same sort of safety, progress and deadlock analysis as described for the basic progress properties.

The satisfiability of system-specific properties on composition can be determined by examining their meta-properties for the desired composite structure, as was done with the basic properties. It is also possible to re-evaluate the system-specific property within the context of the composite system, although this defies the goal of composition, which is to determine *a priori* if a system will continue to be satisfied on composition. If a system-specific property does not exhibit the meta-properties that allow us to do this, then we will have to resort to a reassessment of the property for the composite system.

10.7 Composition Principle

The fundamental problem of composing specification, as stated by Abadi and Lamport, is "to prove that a composite system satisfies its specification if all its components satisfy their specifications" [AbLa90]. On the basis of the Composition Principle of Abadi and Lamport, we introduce a (circular) Composition Principle that is relevant for both safety and progress assumptions of the environment's behaviour. This principle is presented for two components for simplicity. It is easily extended to handle multiple components.

Properties that are relevant to a component or system are said to belong to M, the set of machine properties. Those properties that are assertions about the environment belong to E, the set of environment properties. M_{reqd} is the set of properties that are known to be relevant to a component or system but are not composable and must therefore be evaluated for the composite system.

1.

$$\forall P_j : \mathbf{P}(C_1) \cup \mathbf{P}(C_2) \cdot P_j \in M$$
$$\textbf{if } P_j \in \mathbf{P}(C_1) \cap \mathbf{P}(C_2) \wedge \mathit{composable}(P_j) \Rightarrow \mathbf{sat}(\mathbb{C}, P_j)$$
$$\textbf{if } P_j \in \mathbf{P}(C_1) \cap \mathbf{P}(C_2) \wedge \neg\, \mathit{composable}(P_j) \Rightarrow P_j \in M_{reqd}$$
$$\textbf{if } P_j \in \mathbf{P}(C_1) \setminus \mathbf{P}(C_1) \cap \mathbf{P}(C_2) \wedge \mathit{relevant}(\mathbb{C}, P_j)$$
$$\Rightarrow \mathit{emergent}(P_j) \wedge P_j \in M_{reqd}$$
$$\textbf{if } P_j \in \mathbf{P}(C_2) \setminus \mathbf{P}(C_1) \cap \mathbf{P}(C_2) \wedge \mathit{relevant}(\mathbb{C}, P_j)$$
$$\Rightarrow \mathit{emergent}(P_j) \wedge P_j \in M_{reqd}$$
$$\forall P_j : M_{reqd} \cdot \mathbf{sat}(\mathbb{C}, P_j) \Rightarrow P_j \in M$$

A system $\mathbb{C}$ is known to satisfy the property P_j if each component C_i in the system $\mathbb{C}$ satisfies P_j and P_j is composable. A system $\mathbb{C}$ will satisfy the property P_j if at least one component C_i in the system $\mathbb{C}$ satisfies P_j and P_j is relevant to the composite system; P_j is an emergent property for the composite system (pedantic properties are outside the scope of the composition principle).

2.

$$\textbf{if } \forall P_j : \mathbf{P}(\mathbb{C}) \cdot P_j \in E \wedge \mathbf{sat}(\mathbb{C}, P_j)$$
$$\textbf{and } \forall P_k : \mathbf{P}(C_2) \cdot P_k \in M_2 \wedge \mathbf{sat}(C_2, P_k)$$
$$\textbf{then } \forall P_l : \mathbf{P}(C_1) \cdot P_l \in E_1 \wedge \mathbf{sat}(C_1, P_l)$$
$$\textbf{if } \forall P_j : \mathbf{P}(\mathbb{C}) \cdot P_j \in E \wedge \mathbf{sat}(\mathbb{C}, P_j)$$
$$\textbf{and } \forall P_k : \mathbf{P}(C_1) \cdot P_k \in M_1 \wedge \mathbf{sat}(C_1, P_k)$$
$$\textbf{then } \forall P_l : \mathbf{P}(C_2) \cdot P_l \in E_2 \wedge \mathbf{sat}(C_2, P_l)$$

A component C_i in the system $\mathbb{C}$ satisfies the relevant environment properties (including environment assumptions) if the composite system environment satisfies the required properties and assumptions and the remaining components $C_k \neq C_i$ satisfy the relevant component properties.

Figure 10.4: Composition Principle, Steps 1 and 2

Step 1 of the principle proves the properties of the composite system based on the properties of the system's components. This step considers the emergent properties (if any) of the composite system. Step 2 of the principle allows us to prove that each component satisfies its relevant properties by proving that the properties of the other components and the environment assumptions of the composite system environment are satisfied. Thus these properties ensure the required behaviour of a component's meta-environment. In Step 3, we prove that each component satisfies the relevant component properties by proving that the assumptions of the component's environment are satisfied.

3.

$$\textbf{if } \forall P_j : \mathbf{P}(C_i) \cdot P_j \in E_j \wedge \textbf{sat}(C_i, P_j)$$
$$\textbf{then } \forall P_k : \mathbf{P}(C_i) \cdot P_k \in M_i \wedge \textbf{sat}(C_i, P_k)$$

Every component C_i satisfies the relevant component properties if the relevant environment properties are satisfied.

Figure 10.5: Composition Principle, Step 3

Given the meta-properties of the component properties and knowledge of the composite structure, we can determine which properties, if any, are composable. By 1 of the Composition Principle, these properties will be satisfied by the composite system. We may also examine all properties that may act as emergent properties, using the algorithm of Figure 9.1 for multiple component systems.

For a given component, if the composite system and remaining components satisfy the required environment properties and assumptions, then by 2 of the Composition Principle, the environment properties and assumptions at the component will be satisfied. If the properties and assumptions of the environment of a component are satisfied, then the safety and progress properties of the same component will be satisfied, by 3 of the Composition Principle. Steps 2 and 3 can be verified using the approaches described in Figures 10.1, 10.2, and 10.3.

The Composition Principle is circular in that each step implies the next steps. Thus we assess the Principle by treating one of the steps as a hypothesis and then proving that each hypothesis implies the next step. We must also prove that one of the hypotheses is true for the system in question. Hypothesis 3 should follow because we will normally compose components with known, satisfied properties. Thus the proof of hypothesis 3 should involve the least amount of effort.

Given the proof of 3 of our Composition Principle, we then prove 1, given 3, use 1 to prove 2, and then use 2, to verify the correct assumption of 3. In section 11.4 we use the Composition Principle to evaluate the composability and security of a network system.

10.7.1 Comparison with Composing Specifications

The 1990 "Composing Specifications" technical report by Abadi and Lamport [AbLa90] is the first comprehensive exposition of the requirements of the specification of composable systems and programs. Like the Abadi and Lamport Composition Principle, our definition of the

Composition Principle defines the circumstances for the properties of a set of components to be properties of a composite system.

Abadi and Lamport require that "a specification should assert that the system performs properly if the environment does; it should not assert that the environment performs properly" [AbLa90]. They also require that a specification is *realizable* if the system can always behave in a manner required by its safety and liveness properties, no matter what the environment does. In terms of game theory, this requires that "a specification is said to be *realizable* if and only if the system has a winning strategy -- that is, if and only if the system can always win no matter what moves the environment makes" [AbLa90].

These two statements taken together seem to require that the system specification is sufficiently robust that the system can deal with *any* possible input. Otherwise, this requires that we do in fact know how the environment will act, in which case we are implicitly defining how it should behave, or what is more important, how it will not behave. In our approach, we do assert that parts of the environment will behave in a particular manner when we consider the predictable components in a component's meta-environment. We do not make any claim about the realizability of the system, recognising that there will be situations that cannot possibly or feasibly be protected against.

The environment portion of the safety properties defined in this dissertation ($\sigma(\mathcal{E};i) \models_x A$ or $\sigma(i;\mathcal{E}) \models_x A$) correspond to E, the Abadi-Lamport assumptions about the environment. These assumptions must be true if the machine (system) properties, M, are to be true. The component aspects of the safety properties and the progress properties defined in this dissertation correspond to the machine properties, M of the Abadi-Lamport Composition Principle.

Our Composition Principle does account for progress assumptions of the environment. We claim that the progress properties do not allow or disallow events that are disallowed or allowed by the input-output behaviour rules, a rationale for excluding progress from the Abadi-Lamport Composition Principle. This is a more accurate representation of reality in that users often cancel processes, change priority of processes, etc. Our Composition Principle also takes into account the emergence of properties on composition, and the different properties of a component that may not be common to all components within the system. This principle does not consider type 2 emergent properties. These properties are outside the composability analysis of a system and must be evaluated individually for the composite system.

The Composition Principle given is not constrained only to properties classified according to the Alpern-Schneider safety-liveness classification. Instead the properties of this work are classified according to the Manna and Pnueli safety-progress classification, which contains properties represented as safety and liveness properties. Thus the Composition Principle presented is flexible in that it will handle a broader classification of properties.

11. Examples and Applications

As there is Justice in heaven,
And fire in the hand of God,
The reckoning must be made in the end.
SOPHOCLES: Electra
(tr. E.F. Whatling)

In this chapter we will illustrate and apply the work of previous chapters in several detailed examples. The compatibility of two well-known security properties, mandatory and discretionary access control, are examined in section 11.1. Section 11.2 examines the emergence and de-emergence of the popular possibilistic property of generalised non-interference. In section 11.3 we apply the basic progress properties of this work to several individual components and examine the composability of these properties for the interconnection of these components. We apply the basic safety properties and the composition approach of this work to the specifications and composition rules of the Trusted Network Interpretation [TNI] in section 11.4. This section also examines the role of compatible and emergent properties for the composition of network systems. We include only the highlights of each example. The details of these examples are included in Appendix F.

11.1 Compatibility of Properties

In this example we examine the compatibility of two well-known security properties, Mandatory Access Control, MAC, and Discretionary Access Control, DAC. These properties have been successfully, simultaneously implemented on multi-level security systems. We show, however, that these policies are not compatible as defined. We examine their compatible implementation and the covert channel that results from an incorrect order-of-application. This example illustrates the need to determine the compatibility of properties, even those that appear to compatible in existing implementations.

DAC requires that a "TCB [Trusted Computing Base] shall define and control access between named users and named objects (e.g., files and programs) .... The enforcement mechanism ... shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or both, and shall provide controls to limit propagation of access rights" [TCSEC].

Mandatory access control requires that the "TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control (e.g., processes, files, segments, devices)" [TCSEC]. This access control is based on the comparison of *security levels* assigned to all subjects and objects. A subject's level must dominate the object's level in order to read from it and must be dominated by the object's level in order to write to it.

The details of the analysis of MAC and DAC are included in Appendix F.2, where we formally define MAC and DAC and show that MAC $\wedge$ DAC = $\perp$, indicating that both properties cannot be simultaneously true. We also show that MAC $\not\Rightarrow$ DAC and DAC $\not\Rightarrow$ MAC. A system that satisfies MAC is not guaranteed to imply DAC and vice versa. Thus neither property places a strictly dominant requirement on the behaviours required by the other property. MAC and DAC are *partially compatible* in that there are behaviours allowed by both properties, disallowed by both properties, and that are allowed by one property and not the other property.

MAC cannot be implemented at a system that satisfies DAC without modifying the granting behaviour of the system and increasing the scope of MAC. DAC is not compatible with MAC as it allows for less restrictive access to objects in a system. DAC cannot be implemented at a component that satisfies MAC without modifying the granting behaviour of the system and reducing the scope of DAC. MAC is not compatible with DAC as it requires a more restrictive access to objects in a system. For MAC and DAC to be compatible either the scope of MAC must be increased to accommodate DAC-allowable behaviours, or the scope of DAC must be reduced to accommodate MAC-disallowed behaviours.

In practice, MAC and DAC are simultaneously implemented using mechanisms that filter, or disallow, behaviours that violate either property. These mechanisms use access control lists as look-up tables to determine the acceptability of a given access request. MAC and DAC are forced to be compatible using these mechanisms. Thus MAC and DAC, as implemented, do not correspond to the informal definitions given above.

11.1.1 Compatibility and Covert Channels

There is a covert-channel inherent in the use of MAC and DAC as described above. This is best illustrated by an example. Consider a user A who owns an object Z such that both A and Z strictly dominate another user B. According to MAC, B may write to Z but may not read from Z. According to DAC, user A may grant and revoke B's access to Z. The MAC-allowable accesses to Z by B will be static throughout the life of the system. That is, it is common to assume that the security ratings of objects and subjects within a system will not change. The DAC-allowable accesses to Z by B may change, if A changes the permission she chooses to grant to B (for example, using the UNIX command chmod). A and B may collude to allow B to read information from A's security level by using object Z and DAC.

Assume that A and B decide that a DAC-allowable write to Z corresponds to binary 0 and a DAC-unallowable write to Z corresponds to binary 1. If B repeatedly attempts to write to Z while A is constantly changing B's write privileges, a covert channel is created through which B may read information from a higher security level.

We postulate that this covert channel is a result of the incompatibility of MAC and DAC together with the mechanisms used to enforce MAC and DAC. If MAC and DAC allow and disallow the same behaviours, this covert channel would not be possible. It is because they do not always agree on the behaviours to be allowed or disallowed, together with the potentially dynamic nature of DAC-allowable accesses and the static nature of MAC-allowable accesses, that leads to this covert channel.

11.2. Emergent Properties

In this example we examine the composition of components and examine the role of GNI as an emergent and de-emergent property for a cascade and feedback composition of components using the algorithms of Chapter 9. The example used is a well-known and well-used example of a non-GNI secure composite system [Rush91, TLBS88]. We find that GNI security can act as both an emergent and a de-emergent property. Given the conditions for the de-emergence of GNI, we propose a means of modifying and composing systems such that GNI-security will be composable. This approach allows us to identify the sources of the non-composability of GNI, rather than simply proposing quick fixes that solve this problem for this particular system, but not necessarily for all systems.

The composite system consists of two GNI-secure systems, a simple wire (A) used to create a feedback connection, and a composite system (B) that combines two high-level inputs to create a low-level output. This system is shown in Figure 11.1.

Both systems are GNI-secure in that any changes to the high-level inputs cannot be detected as such by the low-level users; it is possible that a high-level output may be added or deleted to nullify any changes. The details of this analysis are included in Appendix F.2.1, where B2 and B3 produce the exclusive-or of their inputs and B1 acts as a random noise generator.

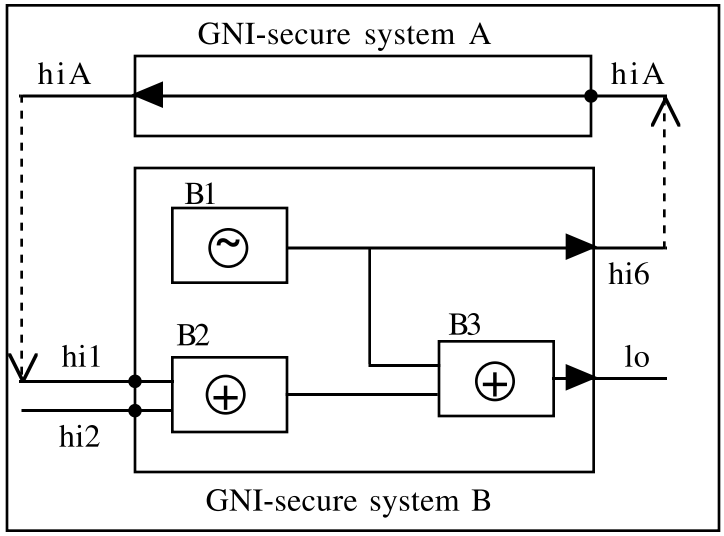

Figure 11.1: GNI-Insecure Composite System

When we examine composite system B we find that the components of B are *not* all GNI-secure. We also find that there is a fundamental problem with the definition of these components. *Two behaviourally identical components are not defined as such.* Components B2 and B3 are defined to take two high-level inputs and produce an output that is the exclusive-or of these inputs. Component B2 produces this output as a high-level output and is clearly GNI-secure. Component B3 produces this output as a low-level output and is just as clearly not GNI-secure; GNI does not allow for down-grading of information. Any change in the output of B3 can be directly attributed to a change in the high-level inputs of B3. We show that this is the true source of the GNI-insecurity of the feedback composite system A and B (see Appendix F.2.2). Thus the GNI-insecurity of this composite system is inherent in the components of the system.

We postulate that many cases of GNI-insecurity are *not* due to the non-composability of GNI. Instead, we believe that this is a direct result of allowing non-GNI-secure components to be a part of a system that is desired to be GNI-secure. It is the composition of such systems with other systems, with or without feedback, that allows the GNI-insecurity of sub-components to affect the overall behaviour of the component.

This is a powerful result that indicates that we must ensure that all components in a system are GNI-secure, at the lowest level possible. This contradicts the guidelines of the Trusted Network Interpretation, which states that

> The primary difference between a processing component, regarded as part of a larger network system, and regarded as a stand-alone ADP [automated data processing] system is that as a stand-alone system all of the TCSEC requirements for a particular class must be met ... In the context of a larger system, however, it may well be (and usually is) the case that the set of policy-related features to be supported by the component need not be the complete set required for a stand-alone system: *features not supplied by one component for the system are supplied by another* [Italics added] (p. 194, [TNI])

As an alternative to disallowing this composite system because of the behaviour of component B3, we may choose to verify that B3 will correctly "write down", producing a low-level output such that no information about the high-level inputs can be determined by a low-level user. An example of such a system is an encryption device that creates a low-level encrypted output for transport of high-level information across an insecure network. We must take care however, to ensure that this does not result in a situation such as discussed above where the composition of such a device with other components will result in an insecure system.

It has been argued that introducing a delay element into component A will have the effect of allowing the composite system to be GNI-secure. The same effect would be realised by introducing a delay into the components of B, requiring the output of each component to occur at least one unit after the input, instead of the simultaneous behaviour described above. It has been proven that if this is the case then we can no longer determine anything about the high-level inputs through feedback for this system.

The failure of GNI has been attributed to feedback, in particular the instantaneous feedback of the system [McC87, McC88, TBLS88, Rus91, McL94, ZaLe95]. While this does solve the problem given in this example, we believe that it gives a false security. Our first priority must be to ensure that GNI (or any relevant property) is satisfied at the lowest possible level. Thus a delay introduced into component B3 would be a proper use of delays to solve the non-composability of GNI.

11.3. Progress and Liveness

This example considers the well-known dining philosopher's problem. We begin by examining the basic progress properties with respect to the individual philosopher and fork components. We examine the satisfiability of the basic progress properties for individual components and the composability of these properties. Using the algorithms of Chapter 10, we examine the composite system for deadlock given the input-state-dependent and input-state-independent behaviours of the composite system. We show that our approach can be applied to a well-understood problem and that it will provide new insights into the failure of progress in this system.

This example can be shown to be analogous to a multi-level security system that prohibits processing unless a component owns its next-lowest and next-highest level security components (the left and right-hand forks). We have chosen to use this example because it is well understood and has been studied from a program point of view using a number of different formalisms ([Hoa78], [OwGr76], [Pete81]).

11.3.1 Problem Definition

The problem of the dining philosophers revolves around a number of P-components (philosophers) who have two types of action: internal processing (thinking) and processing a request from the environment (eating). A P-component that receives a request from the environment must gain possession of two (predefined) F-components (forks) in order to process the request. Once the P-component has finished processing, it will release both of the F-components and it will return to its internal processing tasks.

There are the same number of F-components and P-components, so that each F-component may be granted to one of two P-components. This may lead to every P-component possessing one F-component, so that no component is able to progress: each component is livelocked and the system is deadlocked. We consider a system with six components, three P-components and three F-components.

The problem is depicted below in Figure 11.2 using a block diagram of philosopher and fork components. Each of the n components is numbered. The P-components are numbered from 0 to $(\frac{n}{2} - 1)$, as are the F-components.

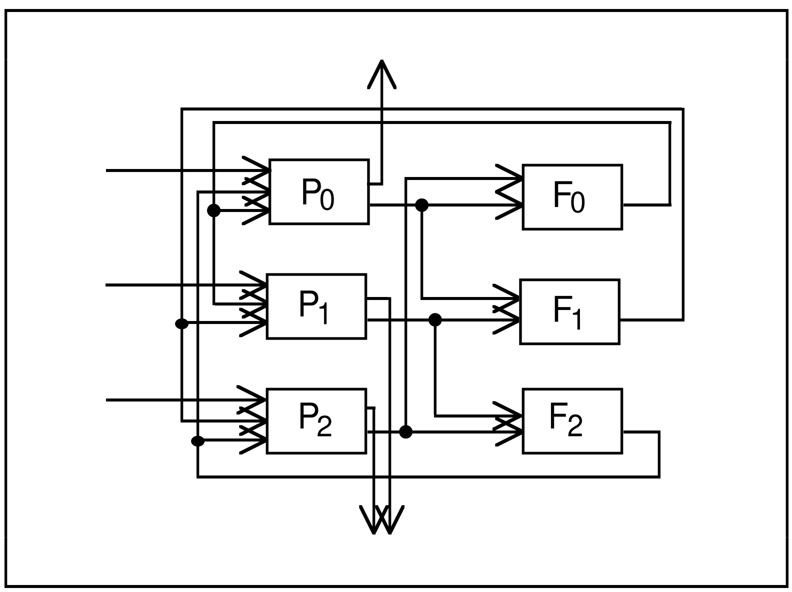

Figure 11.2: Dining Philosophers Problem Represented as Composite System

We examine the conditions for progress of the individual components and then examine these conditions and the progress of the components within the composite system. The details are included in Appendix F.3.

Appendix F.3.1 specifies the behaviours of the philosopher and fork components that make up the composite system. In Appendix F.3.2 we prove the satisfiability of the basic progress properties for the individual components and highlight behaviours that may cause problems on the composition of the components. By examining the input-state dependent behaviours, we see that progress may fail if a philosopher requests access to a fork that is held by another philosopher. This is an example of resource contention. Because the fork may be released if the appropriate input is sent to the fork, it is classified as a lack-of-input failure.

Appendix F.3.3 specifies the behaviour of the composite system. We examine the system behaviour axioms, given the possible failures highlighted in F.3.2 and the algorithm of Figure 10.3. We find that the fork component will not be able to satisfy a request $getF_i$ if the fork is in the state *granted* and the input $releaseF_i$ is never received, indicating a lack-of-input failure of progress.

Using a deadlock detection algorithm based on the Chandy-Misra-Haas algorithm [CMH83], we probe the paths through the system to determine if the system may enter the state where a component F_i is in the state *granted*, receives a request $getF_i$ and yet never receives the required input $releaseF_i$.

Appendix F.3.4 gives a solution to the deadlocking behaviour of the composite system by modifying the behaviour of the philosopher components, based on the observations of F.3.2 and F.3.3. We then prove that the resultant system is deadlock-free. This approach to progress and failure of progress on composition can be applied to realistic systems, for example, those modelled by the dining philosophers problem. The use of the basic progress properties allows us to easily identify potential problems, simplifying the process of evaluating deadlock and failure of progress.

11.4. Network Security

In this example we examine the composition rules proposed in the Trusted Network Interpretation [TNI]. These are high-level rules describing the high-level goals of a network system composed of individual components. We show how these security requirements, defined in the TNI, relate to the basic safety properties defined in this dissertation. We also examine the assessment of emergent, de-emergent and composable properties within the approach of the TNI. We evaluate a simple composite network system, given the domain-dependent safety properties, and the role of the Composition Principle of Chapter 10 in the assessment of the composite system properties. The Composition Principle allows us to systematically evaluate the security of a composite system, using the meta-property approach described in this dissertation.

11.4.1 TNI Composition Rules

The TNI rules define how to combine components that are categorised as M (MAC), D (DAC), A (Audit) or I (Identification and Authentication), based on the role of the component. These rules state the overall requirements that must be met in terms of policies of the composite system given policies of the component systems. How these policies or properties are to be assessed is not addressed, nor is it made clear how properties are to be assessed for composite systems, other than "[i]t is expected that the composition of a [D,I,A,M]-Component will require significant engineering and system architectural consideration" [TNI].

The TNI requirements specify that the individual policies of different components must remain valid for these components on composition. Thus the policies defining the component as one of (D, I, A, M, DI, DA, IA, MD, MI, MA, IAD, MDA, MDI, MIA, MIAD) must not act as de-emergent properties on composition. When composing components for use within a network, the composition rules require that the target Network-(D,I,A,M) Policy must be supported by the composite system. Thus the Network Policy must emerge on the composition of individual components.

On the composition of two D-Components it is required that the combination of the individual DAC Policies, and the subjects and objects that they control, must support the target Network DAC Policy. Thus the individual DAC properties must be compatible and the desired Network Policy must emerge on composition.

The composition of M-Components requires that components may only be connected if the sensitivity levels (the most restricted security access level and the least restricted security access level) associated with the components are the same. Thus the TNI composition rules do not allow a component rated for Confidential to Secret to be composed with a component rated for Secret to Top Secret, even if it is legitimate for the Secret users to write to the Top Secret information of one component and read from the Confidential information of the other component.

The reason given for this is because of the vulnerability of the system should a Confidential user gain access to Secret information, and then use this to gain information to the Top Secret information. We maintain that there may be systems where this is an acceptable risk and the interconnection can be allowed, or the overall threat of such an action is not great enough to disallow such a connection. Thus while it may be reasonable to require a packet switch to be able to handle all possible levels of information if it is to be included in a system, it may not be reasonable to exclude the higher-security medical records storage from an overall networked hospital system, as is allowed given the optimistic safety properties of Chapter 6.

The component classifications given in the TNI, and the attendant component properties, fall outside the basic safety and progress properties. Components that are evaluated according to the requirements of the TNI must also satisfy the basic safety and progress properties, according to the requirements of this work. The M, D, I and A properties represent system-specific properties that are required by the TNI for the security of the composite network system.

11.4.2 Composite Network System

Let us compose two network components, A and B, using the switch S to direct network traffic between the components. The composite system is illustrated in Figure 11.3. In this example we will focus on the safety properties of the system and use the Composition Principle to demonstrate the safety of the composite system. The assessment of progress was illustrated in the example of section 11.3 and will not be addressed in this example.

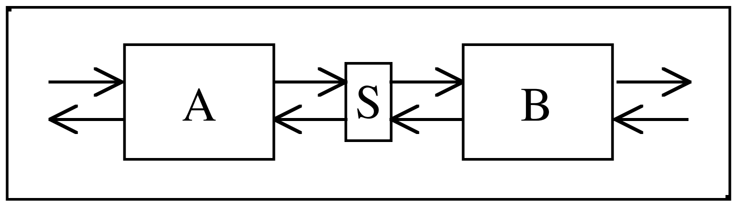

Figure 11.3: Composite Network System

Component A is defined to handle inputs and outputs at the {Confidential, Secret} level. Component B is defined to handle inputs and outputs at the {Secret, Top Secret} level. Thus the switch S must be able to handle inputs at all three levels {Confidential, Secret, Top Secret} according to the requirements of the TNI. This also implies that events that may be unacceptable to component A (Top Secret level events, for example) will be acceptable to the switch S, and likewise for component B given Confidential level events.

Let us assume that the individual components have been evaluated and shown to satisfy the basic safety properties, such that:

$$\forall \text{bsp} : \{\text{AU}, \text{COC}, \text{COI}\} \cdot \textbf{sat}(\text{A}, \text{bsp}) \wedge \textbf{sat}(\text{B}, \text{bsp})$$

$$\forall \text{bsp} : \{\text{AU}, \text{PC}, \text{PI}\} \cdot \textbf{sat}(\text{S}, \text{bsp})$$

Components A and B satisfy the component optimistic confidentiality and integrity properties. Thus it is assumed that the switch component will not send unacceptable events to either component. We make no such assumptions about the behaviour of the environment. The switch component S is assumed to satisfy pessimistic confidentiality and integrity. This implies that all events received by the switch are acceptable. If we consider the security levels given their acceptability and unacceptability, we see that

$$\Phi(\text{A}(\text{A})) = \Phi(\text{Confidential, Secret})$$

$$\Phi(\text{B}(\text{A})) = \Phi(\text{Secret, Top Secret})$$

$$\Phi(\text{S}(\text{A})) = \Phi(\text{Confidential, Secret, Top Secret})$$

Using the Composition Principle, we begin by assuming rule 3 of the principle, that is, that the components will satisfy their relevant component properties if the environment properties (assumptions) are satisfied. This is given as:

$$\textbf{sat}(\text{A}, \text{COC}) \wedge \textbf{sat}(\text{A}, \text{COI}) \Rightarrow \textbf{sat}(\text{A}, \text{AU})$$

$$\textbf{sat}(\text{B}, \text{COC}) \wedge \textbf{sat}(\text{B}, \text{COI}) \Rightarrow \textbf{sat}(\text{B}, \text{AU})$$

$$\textbf{sat}(\text{S}, \text{PC}) \wedge \textbf{sat}(\text{S}, \text{PI}) \Rightarrow \textbf{sat}(\text{S}, \text{AU})$$

Using rule 3, we prove rule 1, then prove rule 2, and then verify the assumption that rule 3 is satisfied given rules 1 and 2. Each rule proves the next rule; the circular Composition Principle indicates that the system is safe, given that the hypothesis of rule 3 is true for the system. The details of the assessment of the system are given in Appendix F.4.1.

The composite system satisfies authenticity, component optimistic confidentiality and component optimistic integrity. In this composite system, the switch is assumed not to send Top Secret events to component A and not to send Confidential events to component B. The events produced by components A and B are acceptable to the switch, so that the properties of pessimistic confidentiality and integrity at S are the same for this implementation as the properties of component optimistic confidentiality and integrity applied to S. That is, we can be optimistic about the behaviours of component A and B with respect to S because we know that by definition of the acceptable events of the switch, these components cannot send unacceptable events to the switch.

12. Summary and Conclusions

But enough of tales --
I have wept for these things already
EURIPIDES: Helen
(tr. Philip Vellacott)

In this section we summarise the work of this dissertation and compare it with the Composing Principle introduced by Abadi and Lamport. We reiterate the contributions of this dissertation and briefly discuss some areas of future work based on the work of this dissertation.

12.1 Summary

This dissertation presents an examination of the properties of systems and the behaviour of these properties when systems are composed. This provides a solution to the open problem in computer security, that of being able to determine, *a priori*, the properties of a composite system based on the properties of the component systems.

The basic safety and progress properties defined in this dissertation represent necessary (but not always sufficient) requirements for the safe and live behaviour of a system. These properties are classified as safety and progress. They are common to all systems and provide a basis for ensuring the minimum level of security of a system.

We describe how to define system-specific goals, based on the individual requirements of specific systems. This will allow us to tailor our approach to the different security requirements, such as those of computer networks, distributed databases or systems and so on, to meet their specific needs. Thus we have a common approach to security and to the determination of system-specific security requirements to be used as a basis in different systems.

Meta-properties indicate how properties will behave when systems are composed. If properties are not satisfied on composition, knowledge of these meta-properties will indicate where and why properties will no longer be satisfied. This in turn will allow us to address these situations and create secure composite systems with the desired properties.

The meta-properties that are considered are transitivity, reflexivity, symmetry, and closure. Each of these meta-properties is considered within the context of different composite system structures, product, cascade and feedback. A property may exhibit different meta-properties, dependent on the composite structure that is employed. Again, foreknowledge of potential problems will allow us to create secure composite systems, eschewing troublesome structures.

The properties as defined allow for different levels of trust in the behaviour of the environment. Thus we explicitly include the behaviour of the environment in our specification of system security. This more accurately reflects reality, where the cost (monetary or temporal) of complete security, independent of the environment, is often prohibitive. Properties that are domain-dependent are those that rely on the assumptions that we make about the behaviour of the environment. As long as these assumptions are reasonable and justifiable, we can create secure composite systems based on these properties and assumptions.

We consider the compatibility of multiple properties that are required to be simultaneously satisfied by the behaviour of a system. Compatible properties ensure that properties do not impose conflicting goals on the behaviour of the other (compatible) properties. We examine one common application of standard security requirements that are shown to be incompatible. We postulate that the covert channels that may result from the implementation of these security requirements is a result of their incompatibility.

Emergent properties, based on the possible behaviours of composite systems, define required system-specific properties. Emergent properties define the security goals corresponding to the behaviours that emerge for a composite system, given the behaviours of the individual components and their interconnections. Emergent properties are also examined to ensure that properties that are required of a system do not act as de-emergent properties. This would imply that the desired security functionality of a system is lost when it is created. Thus we can use the examination of emergent and de-emergent properties to ensure that the desired properties of a system emerge and are maintained on composition.

Composability is a form of emergent property, where the property is satisfied by each of the individual components and the interconnection of the components. Composability of properties is desired as it obviates the need to assess the more complex, composite system. We examine the composability of the basic properties and provide a means of assessing the satisfiability of non-composable properties.

Given this detailed examination of the role and behaviours of properties, we propose a circular Composition Principle that will allow us to evaluate the security properties of a composite system. This Composition Principle encompasses the aspects of security and security assessment introduced in this work.

12.2 Real-World Applications

The 1992 paper "Issues in the Specification of Composite Trustworthy Systems" [HeGa92] identifies several issues that must be addressed in the modelling and specification of secure composite systems as well as criteria for evaluating a model with respect to a real-world application. In this section we examine the ability of our approach of this work to handle the issues. We also evaluate our model and approach given the criteria identified by Hemenway and Gambel [HeGa92].

12.2.1 Issues for Real-World Application

Real-world applications of the modelling and specification of security must consider the issues of component roles, component dependency relationships, granularity of model elements, label consistency and granularity, security polices, secondary or supporting policies, and granularity of execution. In addition, there are two practical issues that must also be considered: component modifications and automated tool support.

Security functions and security-relevant data of a distributed Trusted Computing Base (TCB) may require different components to replicate these functions or they may be distributed across components based on the role of the component.

- Composability and compatibility of the security functions will be used to ensure that they are correctly implemented at the required components.
- Distributed functionality is examined using the emergent properties; this will ideally correspond to the required composite TCB functionality.
- Compatibility and emergence of properties are used to examine the dependencies of properties for given component dependency relationships.

"It is desirable to map security policies from different components to a common abstract policy" [HeFe94].

- This can be accomplished by ensuring the compatibility of the policies and the emergence of the required common policy.

Different components from different vendors may use their own definition and format for security labels. This is external to the model or the properties of the system. Label consistency must be maintained for any modelling and specification approach. We must ensure that once consistently labelled, the security policies enforced with these labels are compatible.

Hemenway and Fellows state that "a collection of independently developed COTS [commercial off-the-shelf] components may differ with respect to what security policies the component enforces." [HeFe94] This applies to both security polices and secondary polices, those polices that depend on other polices.

- We must ensure the compatibility of these polices and their compatibility for those components that are to be interconnected.

Granularity of execution occurs when atomic system actions are actually sequences of actions at the component level.

- This is handled by the choice of the level of abstraction used to represent the system components and its internal and external sequences of events.

The modelling considerations that are required by Hemenway and Gambel are addressed by the consideration of the compatibility, composability and emergent of properties. This is encouraging, as it indicates that there are no substantial issues that must be addressed that cannot be handled by the approach of this dissertation.

The practical issues of allowing for component modifications is addressed in this work. We must ensure that system modifications are consistent with or reflected in the goals of the system. Given this, "the system integrator needs the capability to iterate through the decomposition/composition effort, examining the effects of various changes on both the overall system requirements and the individual component capabilities." [HeFe94]

Hemenway and Fellows also state the importance of automated tool support for as much of the modelling effort a possible. We agree with this requirement; future work in this area will include the automation of the algorithms and approaches presented.

12.2.2 Criteria for Real-World Applications

In applying the Abadi-Lamport Composition Principle to real-world systems, Hemenway and Fellows consider four criteria [HeFe94]. Evaluating our model and specification according to these criteria, we find that our approach fits these criteria at least as well as the Composition Principle. The evaluation of these consideration is given as:

1. Ability to handle general properties

- The properties that are handled in our Composition Principle include both safety and progress properties, a more general class of properties than the safety-liveness classification handled by the Abadi-Lamport Composition Principle.

2. Use of a state machine model, with state hiding

- Our approach permits us to hide or reveal internal structures as needed for the design and analysis based on the level of abstraction chosen to model the system. We do not explicitly consider a state machine model, although our work can be easily extended or translated to such a model.

3. Representation of both system and environment. "The need to address explicitly such environmental assumptions becomes critical in composite systems, where each component has as part of its environment the actions of all the other components." [HeFe94]

- We also account for the actions of the users and components within a given component's meta-environment.

4. Ability to handle both sequential and parallel composition

- We allow for sequential (cascade), parallel (product) and feedback composite structures. Thus we can handle most, if not all, large integrated systems.

Thus according to the criteria used by Hemenway and Fellows to evaluate the Abadi-Lamport Composition Principle, the approach presented in this dissertation is also suitable for use in the assessment of real-world systems and applications.

Hemenway and Fellows implement the Abadi-Lamport Composition Principle using the formal verification tool, FDM. This approach did not include liveness or progress because FDM does not have the capability of handling progress. Any application of the Composition Principle given in this dissertation must include progress and safety. This implies we must find a formal development methodology that will allow us to consider progress and progress properties.

12.3 Conclusions

This work provides a solution to the problem of the composability of security. This is a necessary step towards designing and building large, integrated, secure systems. In particular, we are now able to solve the problem posed by McLean (section 1.3). Given a system satisfying properties X (including the basic safety and progress properties) and another system satisfying the properties Y (again including the basic safety and progress properties), composed according to a given composite structure Z (product, cascade, feedback or a combination of these structures), we are now able to predict and correctly identify the properties that the composite system will satisfy.

Security properties, classified according to the safety-progress classification, are able to represent a diverse set of properties. The basic security properties clarify the minimum goals required for all systems, regardless of the level or type of security required for the system. This will allow us to represent more accurately the required security goals of a system.

Knowledge of the meta-properties of these properties, together with the composite structure, allows us to determine, *a priori*, the composability of these properties. Those properties that emerge on composition can also be identified.

The detailed examination of where and why properties may be non-composable can be used to ensure that non-composable properties will still be satisfied by composite systems. The compatibility, emergence and composability of properties will allow us to ensure that properties are satisfied on composition, whether they are composable or non-composable while ensuring that the properties do not exclude or falsify other required properties.

The circular Composition Principle encompasses these aspects of security and provides a methodical approach to the assessment of security on composition, given the meta-properties, composability, and emergence of properties (and compatible properties). This approach is more general than previous approaches and can be applied to the different types of security requirements and properties. This approach is shown to be practical for real-world applications.

This work has allowed us to highlight the incompatibilities of well-known properties, examine the failure on composition of a well-known possibilistic property and properly attribute this failure, and formally examine and assess the composability of the basic safety and progress properties when applied to well-known examples. The analytic Composition Principle allows us to accurately and easily assess the security of a composite system.

12.4 Future Work

The work of this dissertation points to several areas of future work and research.

- The algorithms and approach given should be automated, and incorporated into an existing formal verification and specification system. This will allow us to examine formally and automatically both correctness and security.
- This work can be incorporated into federal criteria for secure composite systems, as well as into the risk-analysis packages used to assess the security risks of given systems.
- This work should be extended to deal with dynamic connections, such as library links. This is especially relevant with the predominance of networked, distributed systems.
- This approach should be applied to a detailed examination of secure, networked systems. The current concentration on authentication protocols ignores many of the issues of secure (networked) interconnections.
- The relationship between compatible properties and covert channels within a system should be examined in more detail.

Bibliography

[Aba87] Abadi, M., "Temporal-Logic Theorem Proving," Ph.D. thesis, Technical Report STAN-CS-87-1151, Stanford University, Stanford, California, 94305, March 1987.

[Aba88] Abadi, M. The Power of Temporal Proofs. Technical Report 29, Digital Systems Research Center, Palo Alto, California, August 1988.

[AbLa88] Abadi, M. and Lamport, L. The Existence of Refinement Mappings. Technical Report 29, Digital Systems Research Center, Palo Alto, California, August 1988.

[AbLa90] Abadi, M. and Lamport, L. Composing Specifications. Technical Report 66, Digital Systems Research Center, Palo Alto, California, October 1990.

[AbPl92] Abadi, M. and Plotkin, G.D. A Logical View of Composition. Technical Report 86, Digital Systems Research Center, Palo Alto, CA, May 1992.

[AlSc85a] Alpern, B. and Schneider, F.B. Defining Liveness. *Information Processing Letters*, 21(4):114–133, October 1985.

[AlSc87a] Alpern, B. and Schneider, F. Proving Boolean Combinations of Deterministic Properties. In *1987 IEEE Symposium on Logic in Computer Science,* 1987.

[BaKu91] Bacic, E.M. and Kuchta, M. Considerations in Preparation of a Set of Availability Criteria. In *Proceedings of 3rd Canadian Computer Security Symposium,* 1991.

[BeMa77] Bell, J.L. and Machover, M. *A Course in Mathematical Logic*. Amsterdam, The Netherlands: Elsevier Science Publishers, B.V., 1977.

[Bert68] von Bertalanffy, L. *General System Theory: Foundations, Development, Applications*. Volume Revised, George Braziller, Inc, 1968.

[BFGG89] Barringer, H., Fisher, M., Gabbay, D., Gough, G., and Owens, R. METATEM: A Framework for Programming in Temporal Logic. In *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness,* volume 430de Bakker, J.W., de Roever, W.P., and R., G.R., Springer-Verlag, 1989, pages 94-129, Lecture Notes in Computer Science, vol 430.

[Biba77] Biba, K.J. Integrity Considerations for Secure Computer Systems. Technical Report TR-3153, MITRE Corporation, Bedford, MA, April 1977.

[BLP75] Bell, D.E. and LaPadula, L.J. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, Tech. Rep., MITRE, Bedford, MA, July 1975.

[Bony95] Bonyon, D., personal communication, October 1995.

[Brko95] Brkovic, J., *An Automated Approach to the Composition of Event Systems,* Bachelor's Thesis, University of Toronto, April 1995.

[Bu62] Buchi, J.R. On a Decision Method in Restricted Second Order Arithmetic. In *Proc . Internat. Congr. Logic, Method and Philos. Sci.,* Stanford University Press, 1960, pages 1-12.

[ChBe95] Cheswick, W.R. and Bellovin, S.M. *Firewalls and Internet Security*. Professional Computing Series, Addison-Wesley, 1995.

[ClWi77] Clark, D.D. and Wilson, D.R. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy,* IEEE Computer Society Press, April 1987, pages 184–194.

[CMH83] K.M. Chandy, J. Misra, and Haas, L.M. Distributed Deadlock Detection. *ACM Transactions on Computer Systems*, :144-156, 1983.

[CTCPEC] Communications Security Establishment. The Canadian Trusted Computer Product Evaluation Criteria, Version 3.0e. Technical Report 3.0e, Canadian System Security Centre, Governement of Canada, Ottawa, Canada, January 1993.

[DACTS] Department of Defense [U.S. of A]. A Guide to Understanding Discretionary Access Control in Trusted Systems. Technical Report NCSC-TG-003-87, National Computer Security Center,1987.

[DBY94] G. W. Dinolt, L.A. Benzinger, and Yatabe, M.G. Combining Components and Policies. In *Proceedings of the Computer Security Foundations Workshop VII,* IEEE Computer Society Press, 1994.

[DDG] Damm, W., Dohmen, G., Gersetner, V., and Josko, B. Modular Verification of Petri Nets, A Temporal Logic Approach. In *Stepwise Refinement of Distributed Ssytems: Models, Formalisms, Correctness*. Springer-Verlag.

[Denn76] Denning, D.E. A Lattice Model of Secure Information Flow. *CACM*, 19(5):236-243, May 1976.

[FiWo93] Fisher, M. and Wooldridge, M. Executable Temporal Logic for Distributed A.I. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence,* May 1993.

[GaSp91] Garfinkel, S. and Spafford, G. *Practical Unix Security*. Computer Security, O'Reilly & Associates, Inc., 1991, June 1994 printing.

[Gli83] Gligor, V.D. A Note on the Denial of Service Problem. In *Proceedings 1983 Symposium on Security and Privacy,* IEEE Computer Society, IEEE Computer Society Press, 1983, pages 139-149.

[Glig93] Gligor, V.D., personal communication, 1993 November.

[Gold77] Goldman, B. Deadlock Detection in Computer Networks. Technical Report MIT/LCS/TR-185, Tech Report, MIT,September 1977.

[Gold92] Goldblatt, R. *Logics of Time and Computation*. Lecture Notes, Center for the Study of Language and Information, 1992.

[GoMe82] Goguen, J.A. and Meseguer, J. Security Policies and Security Models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy,* 1982, pages 11–20.

[GoMe84] Goguen, J.A. and Meseguer, J. Unwinding and Inference Control. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy,* 1984, pages 75–86.

[Gray90] Gray, J.W. Probabilistic Interference. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy,* IEEE Computer Society Press, May 1990, pages 170–179.

[HeFe94] Hemenway, J.A. and Fellows, J. Applying the Abadi-Lamport Composition Theorem in Real-World Secure System Intergration Environments. June 1994, submitted to Dec. 94 ACSAC Conference.

[HeGa92] Hemenway, J. and Gambel, D. Issues in the Specification of Composite Trustworthy Systems. In *Proceedings 4th Annual Canadian Computer Security Symposium,* 1992.

[Heh93] Hehner, E.C.R. *A Practical Theory of Programming*. Gries, D. and Schneider, F.B., Eds., Springer-Verlag, 1993.

[HiLe93] Hinton, H.M. and Lee, E.S. Secure Composition Based on Environmental Considerations. In *Fifth Annual Canadian Computer Security Symposium,* Canadian System Security Centre, May 1993, pages 463-473.

[HiLe94] Hinton, H.M. and Lee, E.S. The Compatibilty of Policies. In *Proceedings of the Second ACM Conference on Computer and Communications Security,* 1994.

[Hin93] Hinton, H.M. An Environment-Based Approach to the Composition of Safe Systems. Master's thesis, University of Toronto, Toronto, Ontario, Canada, January 1993.

[Hoa78] Hoare, C.A.R. Communicating Sequential Processes. *Communications of the ACM*, 21(8).

[Hoa93] Hoare, C.A.R. Algebra and Models. In *Software Engineering Notes,* volume 181993, pages 1-8, Proceedings of First ACM Sigsoft Symposium on the Foundations of Software Engineering.

[HoSn88] Hoffman, D. and Snodgrass, R. Trace Specifications: Methodology and Models. *IEEE Transactions on Software Engineering*, 14(9):1243-1252, September 1988.

[HuCr68] Hughes, G.E. and Cresswell, M.J. *An Introduction to Modal Logic*. London: Routledge, 1968.

[Hun73] Hunter, G. *MetaLogic: An Intrduction to the Metatheory of Standard First Order Logic*. University of California Press, 1973.

[IsMa78] Isloor, S.S. and Marsland, T.A. An Effective On-line Deadlock Detection Technique for Distributed Database Management Systems. In *Proc. Compsac 78,* November 1978, pages 283-288.

[ITSEC] German Information Security Agency. IT Security Criteria: Criteria for the Evlaution of Trustworthiness of Information Technology (IT) Systems. Technical Report, 1989.

[JoTh88] Johnson, D.M. and Thayer, E.J. Security and the Composition of Machines. In *Proceedings of the Computer Security Foundations Workshop,* IEEE Computer Society Press, June 1988, pages 72–89.

[Kem85] Kemmerer, R.A. Testing formal specification to detect design errors. *IEEE Transactions on Software Engineering*, SE-11(1):32-43, .

[KiSo89] King, S. and Sorensen, I.H. From Specification, Through Design to Code: A Case Study in Refinement. In *Formal Methods: Theory and Practice*, Scharbach, P.N., Ed. BSP Professional Books, 1989.

[Lam77] Lamport, L. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, SE-3(2):125-143, March 1977.

[Lam89] Lamport, L. A Simple Approach to Specifying Concurrent Systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[Lam90] Lamport, L. A Temporal Logic of Actions. Technical Report 57, Digital Systems Research Center, Palo Alto, California, April 1990.

[Lan86] Lansky, A.L. A Representation of Parallel Activity Based on Events, Structure, and Causality. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop at Timberline, Oregon,* Morgan Kaufmann, Los Altos, California, 1986, pages 123-159.

[Lap93] Laprie, J.C. Dependability: from Concepts to Limits. In *Safecomp '93:12th International Conference on Computer Safety, Reliability, and Security,* Gorski, J., Springer-Verlag, November 1993, pages 157-168.

[LaRe92] Landauer, J. and Redmond, T. A Framework for Composition of Security Models. In *Proceedings of Computer Security Foundtions Workshop V,* IEEE Computer Society Press, 1992.

[LBTS92] Lee, E.S., Boulton, P.I.P., Thomson, B.W., and Soper, R.E. Composable Trusted Systems. Technical Report, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, May 1992, Final Report of Contract W2213-9-6351/01-SZ. Available via anonymous ftp as CSRI272 from csri.toronto.edu.

[Lev91] Leveson, N.G., *Issues in Assuring Safety in Composite Trustworthy Systems,* 1992 NATO Workshop on Composite Trustworthy Systems, October 1991.

[Lip81] Lipson, J.D. *Elements of Algebra and Algebraic Computing*. Benjamin Cummings, 1981.

[MaPn81] Manna, Z. and Pnueli, A. Verification of concurrent programs: the temporal framework. In *The Correctness Problem in Computer Science*, Boyer, R.S. and Moore, J.S., Eds. Academic Press, 1981.

[MaPn92] Manna, Z. and Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems*. New York, NY: Springer-Verlag, 1992.

[McC87] McCullough, D. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy,* IEEE Computer Society Press, May 1987, pages 161–166.

[McC88] McCullough, D. Noninterference and the Composability of Security Properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy,* IEEE Computer Society Press, May 1988, pages 177–186.

[McHa69] McCarthy, J. and Hayes, P.J. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence: 4*, Meltzer, B. and Michie, D., Eds. New York, New York: American Elsevier, 1969, pages 463-502.

[McL83] McLean, J., *A Complete System of Temporal Logic for Specification Schemata*, volume 164, Lecture Notes in Computer Science. Springer Verlag, 1983.

[McL87] McLean, J. Reasoning About Security Models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy,* IEEE Computer Society Press, April 1987, pages 123-131.

[McL88] McLean, J. The Algebra of Security. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy,* 1988, pages 2–7.

[McL94] McLean, J. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In *Proceedings of 1994 Computer Society Symposium on Research in Security and Privacy,* IEEE Computer Society, IEEE Computer Society Press, 1994.

[Mil89] Millen, J.K. Foundations of Covert Channel Detection. Technical Report MTR10538, MITRE,1989.

[MiMe84] Mitchell, D.P. and Merritt, M.J. A Distributed Algorithm for Deadlock Detection and Resolution. In *Proc. ACM Conf. Principles of Distributed Computing,* August 1984, pages 282-284.

[MoMa83] Moszkowski, B. and Manna, Z., *Reasoning in Interval Temporal Logic*, volume 164, Lecture Notes in Computer Science. Springer Verlag, 1983.

[NeGo92] Neumann, P. and Gong, L., *Minimizing Trust in Multilevel-Secure Systems,* submitted to National Computer Security Conference, Oct 94, unrevised copy distributed by Peter Neumann .

[Neu94] Neumann, P.G., *Architectural Structure,* Handout Matertials for Workshop on Formal Methods Security Architecture and Separation Kernels, March 1994.

[OHal90] O'Halloran, C. A Calculus of Information Flow. In *Proceedings of European Symposium on Research in Computer Security,* Toulouse, France, 1990.

[Ora90] Orava, F. Verifying Safety and Deadlock Properties of Networks of Asynchronously Communicating Processes. In *Protocol Specification, Testing and Verification, IX,* E. Brinksma, G.S. and Vissers, C.A., IFIP, Elsevier Science Publishers B.V. (North-Holland), 1990.

[OwGr76] Owicki, S. and Gries, D. Verifying Properties of Parallel Programs: An Axiomatic Approach. *Communications of the ACM*, 19(5).

[OwLa82] Owicki, S. and Lamport, L. Proving Liveness Properties of Concurrent Programs. *ACM Transcations on Programming Languages and Systems*, 4(3):455-485, July 1982.

[Par91] Parker, D.B. Restating the Foundation of Information Security. In *Second Annual North American Information System Security Symposium,* Toronto, Ontario, October 1991.

[Par92] Parker, D.B. 26 Principles for Information Security Controls. In *Compsec 92 International: The Ninth World Conference on Computer Security,* November 1992.

[Pete81] Peterson, J.P. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall Inc, 1981.

[Pint94] Pinto, J.A. Temporal Reasoning in the Situation Calculus. Technical Report KRR-TR-94-1, Knowledge Representation and Reasoning, University of Toronto,February 1994.

[Pnu77] Pnueli, A. The Temporal Logic of Programs. In *Proceedings of the 18th Symposium on the Foundations of Computer Science,* IEEE, June 1977, pages 46–57.

[Pnu81] Pnueli, A. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science*, :46-57, 1981.

[Pnu86] Pnueli, A. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. In *Current Trends in Concurrency, Lecture Notes in Computer Science*, J.W. de Bakker, W.P.R. and Rozenberg, G., Eds. Springer-Verlag, 1986, pages 510-584.

[PrNe92] Proctor, N.E. and Neumann, P.G. Architectural Implications of Covert Channels. In *Proceedings 15th National Computer Security Conference,* 1992, Computer Science Lab, SRI International , Menlo Park, CA 94025.

[Rus91] Rushby, J. Composing Trustworthy Systems: A Position Paper. In *Proceedings of the NATO RSG2 Working Conference on Composability,* October 1991.

[Sand93] Sandhu, R. Lattice-Based Access Control Models. *IEEE Computer*, 26(11):9-19, November 1993.

[Sin89] Singhal, M. Deadlock Detection in Distributed Systems. *IEEE Computer* , 22(11):37-48, November 1989.

[Sis85] Sistla, A.P. On Characterization of Safety and Liveness Properties in Temporal Logic. In *Proceedings of Fourth Annual ACM Symposium on Pinciples of Distributed Computing,* ACM, 1985, pages 39-48.

[TCSEC] Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria. Technical Report DOD 5200.28-STD, Department of Defense Standard, National Computer Security Centre, Washington, D.C., U.S.ofA., December 1985.

[TLBS88] Thomson, B., E.S.L., E.S.L., Boulton, P.I.P., Stumm, M., and Lewis, D.M. A Trusted Network Architecture. Technical Report, Computer Systems Research InsitituteUniversity of Toronto, Toronto, Ontario, Canada, October 1988.

[TNI] Department of Defence [U.S. of A.]. Trusted Network Interpretation in Trusted Computer Systems Evaluation Criteria. Technical Report, National Computer Security Center,1987.

[TSLB91] Thomson, B., Soper, R., Lee, E.S., and Boulton, P.I.P. Requirements for a Composable Security Property. In *Proceedings of the Composite Trustworthy Systems Workshop,* Naval Research Labs, Washington, D.C., October 1991.

[Web89] Weber, D.G. Formal Specification of Fault-Tolerance and its Relation to Computer Security. In *Proceedings of the 5th International Workshop on Software Specification and Design,* Association for Computing Machinery, 1989, pages 273–277.

[WiJo90] Wittbold, J.T. and Johnson, D.M. Information Flow in Nondeterministic Systems. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy,* IEEE Computer Society Press, May 1990, pages 144–161.

[Wol89] Wolper, P. On the Relation of Programs and Computations to Models of Temporal Logic. In *Temporal Logic in Specification*, Banieqbal, B., Berringer, H., and Pnueli, A., Eds. Springer-Verlag, 1989, pages 75-123, Lecture Notes in Computer Science.

[Woo87] Wood, D.. *Theory of Computation*. John Wiley and Sons, 1987.

[YuGl88] Yu, C.F. and Gligor, V.D. A Formal Specification and Verification Method for the Prevention of Denial of Service. In *Proc. 1988 Symposium on Security and Privacy,* IEEE Computer Society, 1988, pages 187-202.

[YuGl90] Yu, C.F. and Gligor, V.D. A Specification and Verification Method for Preventing Denial of Service. *IEEE Transactions on Software Engineering*, 16(6).

[ZaLe95] Zakinthinos, A. and Lee, E.S. The Composability of Non-Interference. In *8th IEEE Computer Security Foundations Workshop,* 1995.

Glossary

Glossary of Terms

Assertion	An prescription on unpredictable behaviour, usually of the users in the environment
Behaviour	The sequence of inputs and outputs at a component or system over time.
Behaviour Axiom	Defines the behaviour of a component or system.
Compatible	Not mutually exclusive
Compatible Properties	Two properties that are not mutually exclusive, that is, that can be simultaneously satisfied by the same behaviour of a component or system.
Compatible Property Domain	The set of possible events that are common to two properties that are compatible.
Compatible Property Specification	The conjunction of the property specifications of two individual property specifications
Completeness (of Logic)	A logic is complete if it defines all possible truths
Completeness (of Set of Properties)	The set of properties is complete if it contains all properties relevant to the desired goals (safety and progress) of a system.
Composite Structures	The different interconnections between components.
Product Structure	There are no interconnections between the components
Cascade Structures	The outputs of one component are sent to another component as inputs. The second component does not send (directly or indirectly) an output to the first component.
Feedback Structure	The outputs of one component are sent to a second component as an input. The second component will send outputs to the first components as inputs.
Composite System	The interconnection of two or more individual systems or components
Component	A functional element that may be interconnected to form a composite system or may exist as an individual, unconnected element.
Composite Environment	The environment of a composite system
Composition	The process of interconnecting components and systems to create larger, more complex systems.
Component Behaviour	The sequence of inputs and outputs occurring at a component.
Component Behaviour Axiom	A rule defining how a component will respond to a given input. This response may be dependent upon the state of the component.
Correctness	Implies that a system or program exactly corresponds to the desired specifications for that system or program.
Dependable System	One that will behave in predictable manner, even in the presence of faults.
Domain-Dependent Property	A specification of desired behaviour that depends on assumptions or prescriptions about the behaviour of the unpredictable environment.
Domain-Independent Property	A specification of desired behaviour that depends only on the predictable behaviour of components in a system.
Emergent Behaviour	A behaviour that is possible for an interconnection of components that is not possible or relevant to the individual components.
Emergent Property	A property that defines the desired behaviour of the emergent behaviours of a system.

Emergent Type 1 Property	A property that is defined and relevant to some, but not all, components in a system and is relevant to the interconnection of these components.
Emergent Type 2 Property	A property that is not relevant to any component in a system but is relevant to the interconnection of these components.
Environment	The source of inputs and destination of outputs to and from a component.
Event	A model-relevant action.
Acceptable/Unacceptable	If an event will never cause a bad thing to happen it is acceptable, otherwise it is unacceptable
Harmless	A harmless event is an unacceptable event whose harmful effect has been nullified.
Input/Output Event	An event that either received by a component (input) or produced by a component (output).
Relevant /Irrelevant	A relevant event may act as a stimulus. An irrelevant event will have no direct or indirect effect.
Feedback	The output of a component that is sent to another component (from which the component receives inputs) as an input.
Filtration	The process of removing unwanted events.
Input-Output Behaviour	The predictable relationship between input and output events at a component or system.
Interconnection	A channel between components in a system.
Interconnection Rule	The definition of possible interconnections within a system.
Liveness	The requirement that "something good" will eventually happen.
Meta-Environment	The source of all inputs and destination of all outputs, including any interconnected (predictable) components and the original (unpredictable) environment of a component.
Pedantic Property	A property that is not previously known that becomes relevant and is identified for a composite system.
Process	The execution by a component of an event.
Progress Property	A property, containing the temporal operator, ™, "eventually" that defines the eventuality of an action.
Property	The formal instantiation of a policy, representing the goal or desired behaviour of a component or system.
Reflexivity (of a property)	The satisfiability of a property for a component when the component has one of its outputs fedback to itself as an input.
Risk	A measure of the possibility that an event will occur and the worth, threat and vulnerability associated with the event.
Safety	The requirement that nothing bad ever happen within a system.
Safety Critical System	A system whose failure has catastrophic consequences, for example, the controller of a nuclear power plant.
Safety Property	A property that specifies those aspects of a component's or system[s behaviour that must always be true.
Simple System	A system that is represented as a stand-alone component, without reference to the internal connections or behaviours of the system.
Secure System	A system that satisfies a given set of safety and progress properties.
Sequence	An ordered set of input and output events occurring at a component or system.
Specification	The description of a component, system, or behaviour.
Stand-Alone Component	A component that has interconnections to and from the environment and no connections to and from other components.
State	The internal variables of a component or system.
System	A stand-alone component or an interconnection of such components.
System Behaviour	The sequence of inputs and outputs occurring at a system.

Threat	The potential damage to a system caused by the occurrence of an event.
Transitivity (of a property)	The satisfiability of a property for the composition of two systems given the satisfiability of the same property for the composition of each of the systems with another system
Trust (in a user)	The belief that a user will behave in a defined or predictable manner and will not intentionally cause harmful events to be received by a system.
Verification	The process of proving that the implementation of a system or property corresponds to the specification of that system or property.
Vulnerability (of a system)	The susceptibility of a system to certain, potentially harmful events.
Required World	Those events that are allowed to occur at a system. Events that are excluded from the required world are those that are known to be harmful yet may be received by the system.

Glossary of Notation

Temporal Logic:

$\circ$	Always in the future, a unary temporal connective
$\Diamond$	Eventually in the future, a unary temporal connective
$\bigcirc$	Next time in the future, a unary temporal connectives.
$\mathcal{U}$	Until, a binary temporal connective.
$\exists$	Existence, a existential quantifier.
$\forall$	Universality, a universal quantifier.

General Notation:

$\mathcal{N}$	The components in a system, where there are $\mathcal{N}$ components, numbered from 0 to **n**-1
$\mathcal{E}$	The environment of a system
Φ	The set of events defined for a system
σ	A sequence of events
p	An atomic formula
A	A formula
$\mathcal{F}ma(\Phi)$	The set of formulae derived from the set of events Φ
$(\alpha ; \beta \vee \chi)$	Component behaviour rule, input α causes outputs β or χ.
$(i \rightarrow j)$	Interconnection rule, output of **i** sent to component **j** as an input
A\B	Set exclusion: those members of set A that are not members of set B.
$\mathfrak{C}$	Composite system
$\mathfrak{C}(C_i)$	Component C_i within composite system $\mathfrak{C}$
$\cup_{i=1}^{\mathfrak{C}} \mathbf{P}(C_i)$	Set of all properties for components in $\mathfrak{C}$
$\cap_{i=1}^{\mathfrak{C}} \mathbf{P}(C_i)$	Set of properties common to components in $\mathfrak{C}$
$\mathbf{P}(\mathfrak{C}) \cup= P_k$	Set of properties $\mathbf{P}(\mathfrak{C})$ union property P_k

G($\mathfrak{C}$) Goals of behaviour of composite system $\mathfrak{C}$ represented as a set of properties

Events:

Φ(in) The set of input events defined for a system

Φ(out) The set of output events defined for a system

Φ(A) The set of acceptable events

Φ(U) The set of unacceptable events

Φ(i) or Φ^{i} The set of events relevant to component **i**

Φ(i,j) or Φ^{i+j} The set of events relevant to components **i** and **j**

Φ(i,(char)) The set of events with characteristic char relevant to component **i**

Sequences:

σ(i; j) A sequence of events from component **i** to component **j**

σ(in) A sequence of events consisting only of input events

σ(out) A sequence of events consisting only of output events

$\sigma[k]$ A sequence of events occurring on the channel k, where the first ℓ channels are input channels at a component and the next m-ℓ are output channels.

$\sigma\{\mathbf{i}\}$ The sequence of states of component **i**

$\sigma\{\mathbf{i}\}$(in) The sequence of input events received at component **i**

$\sigma\{\mathbf{i}\}$(out) The sequence of output events produced by component **i**

Several common sequences are defined as:

$$\sigma(i;j)(in) = \langle\, \langle in_0^j \rangle, \langle in_1^j \rangle, \ldots \rangle$$

$$\sigma(i;j)(out) = \langle\, \langle out_0^i \rangle, \langle out_1^i \rangle, \ldots \rangle$$

$$\sigma(i;j)[k] = \langle \langle in_0^i[k] \rangle, \langle in_1^i[k] \rangle, \ldots \rangle \quad \text{iff } k < \ell$$

$$\langle \langle out_0^i[k] \rangle, \langle out_1^i[k] \rangle, \ldots \rangle \quad \text{iff } k \geq \ell$$

$$\sigma(i;j)(in)[k] = \langle \langle in_0^i[k] \rangle, \langle in_1^i[k] \rangle, \ldots \rangle$$

$$\sigma(i;j)(out)[k] = \langle \langle out_0^i[k] \rangle, \langle out_1^i[k] \rangle, \ldots \rangle$$

$$\sigma\{\mathbf{i}\}(in)[k] = \langle \langle in_0^i[k] \rangle, \langle in_1^i[k] \rangle, \ldots \rangle \quad \text{iff } k < \ell$$

$$\langle \langle null \rangle, \langle null \rangle, \ldots \rangle \quad \text{otherwise}$$

$$\sigma\{\mathbf{i}\}(out)[k] = \langle \langle out_0^i[k] \rangle, \langle out_1^i[k] \rangle, \ldots \rangle \quad \text{iff } k \geq \ell$$

$$\langle \langle null \rangle, \langle null \rangle, \ldots \rangle \quad \text{otherwise}$$

$$\sigma(in)[k] = \langle\, \langle in_0[k] \rangle, \langle in_1[k] \rangle, \ldots \rangle \quad \text{iff } k < \ell$$

$$\sigma(out)[k] = \langle\, \langle out_0[k] \rangle, \langle out_1[k] \rangle, \ldots \rangle \quad \text{iff } k \geq \ell$$

The notations σ(i;j)(in) and σ(i;j)(out) are used indicate the relevant aspect of the sequence (the destination or source) to be considered.

$\sigma \models_j A$	The sequence σ entails the formula A at time j.
$\sigma(\mathbb{S};\mathbb{D}) \models_j p$	The sequence from source $\mathbb{S}$ to destination $\mathbb{D}$ entails the event p at time j, where A is a formulae defined as A $\in \mathcal{F}ma\{\Phi(\mathbb{S}(out),\mathbb{D}(in))\}$
$\sigma(i+j;\mathbb{E})$	The sequence of events from components **i** and **j** to the environment

Composite Systems:

$\&(C_i, C_j)$	Composition of components C_i and C_j
$\&^P(C_i, C_j)$	Product composition of components C_i and C_j
$\&^C(C_i, C_j)$	(General) Cascade composition of components C_i and C_j
$\&^{\sim C}(C_i, C_j)$	Non-general cascade composition
$\&^F(C_i, C_j)$	(General) Feedback composition of components C_i and C_j
$\&^{\sim F}(C_i, C_j)$	Non-general feedback composition
C_i^R	The reflexive composition of component C_i

Properties:

$\mathbf{P}(\&(C_i, C_j))$ The set of properties of the composite system $\&(C_i, C_j)$

Q_k	The property domain of a property P_k, $\forall x : \mathbb{N} \cdot \lambda i : \mathbb{N} \cdot \lambda A : \mathcal{F}ma\{\Phi\}$
S_k	The property specification of a property P_k, $\lambda A \in \mathcal{F}ma\{W_k\} \wedge R_k$
W_k	The required world of property P_k, $W_k \subseteq \Phi$
R_k	The specification of the behaviour
$\mathbf{sat}(C_i, P_k)$	True if the behaviour of component C_i satisfies the requirements of property P_k, where $C_i \times P_k \rightarrow \{\top,\perp\}$
$Q_i * Q_j$	Compatible property domain, defined as $(\forall x : \mathbb{N} \cdot \lambda_{i1}\ i : \mathbb{N}_i \cap \mathbb{N}_j \cdot \lambda_{j1} j : \mathbb{N}_i \cap \mathbb{N}_j \cdot \lambda_{i2}\ A\ , \lambda_{i2}\ B : \mathcal{F}ma\{\Phi_i \cap \Phi_j\})$
$S_i \wedge S_j$	Compatible property specification, defined as $(\alpha_i \in \mathcal{F}ma\{W_i\} \wedge \alpha_j \in \mathcal{F}ma\{W_j\}) \wedge (R_i \wedge R_j)$
W	Required world, given as $W = (\alpha_i\ , \alpha_j \in \mathcal{F}ma\{W_i \cap W_j\})$
composable(P_i)	Proposition defining composability of property P_i
relevant(C_i,P_i)	Proposition defining relevance of property P_i to the behaviour of component C_i