THE COMPOSITION OF PROPERTY-PRESERVING EVENT SYSTEMS

BY

JOHN PAUL NESTOR

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

[This page should be blank.]

ABSTRACT

This thesis presents a modelling scheme based upon event traces that have a structure that is captured by formal languages and grammars. This provides a convenient and mathematically well-founded means for dealing with such traces. The external behaviour of a module, including causal and temporal relationships between events, and possible nondeterminism, is successfully modelled. The problem of composing a module from constituent modules is comprehensively studied. It is demonstrated that composition is possible in general. Restrictive properties are also studied and a large class of properties is shown to be preserved under composition.

ACKNOWLEDGEMENTS

A large number of persons and institutions were important in the development of this work. First and foremost, I would like to thank my two research supervisors. Professor E.S. Lee for his patient guidance, advice and assistance throughout my graduate studies and particularly during the development of this thesis; and Professor P.I.P. Boulton for providing that timely second opinion that helped keep things on track.

I would also like to acknowledge the work of the members of my Departmental and School of Graduate Studies Examination Committees: Professors T.S. Abdelrahman, P. Chow, V.D. Gligor, E.C.R. Hehner, S.R. Karpik, F.R. Kschischang, R.M. Needham, and M. Stumm.

In addition, I would like to thank all of my friends in the Computer Group who helped make being a graduate student more than just research and classes.

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

Last, but certainly not least, I would like to thank my parents and family for always encouraging me to strive for excellence, and providing all the support that I needed to achieve it.

Table of Contents

List of Figures ix

List of Appendices xi

Symbol Index xiii

1 Introduction and Overview 1

1.1 Introduction 1
1.2 Modelling 2
1.3 Composability 2
1.4 Event Traces 3
1.5 Causality 5
1.6 Nondeterminism 6
1.7 This Thesis 7
1.8 Overview 8

2 Related Research 11

2.1 Introduction 11
2.2 Confidentiality Models 11
2.2.1 The Bell-LaPadula Model 11
2.2.2 The Orange Book 12
2.2.3 Goguen and Meseguer's Noninterference 13
2.2.4 Sutherland's Deducibility 14
2.2.5 McCullough's Hook-Up Security and Restrictiveness 16
2.2.6 Security and Machine Composition 17
2.2.7 Lin's Behavioural Security 18
2.2.8 Nondeterminsitic Security Modelling 18
2.2.9 Parker's Foundations of Information Security 19
2.3 Integrity Models 20

TABLE OF CONTENTS

2.3.1 Biba's Integrity Model 20
2.3.2 The Clark-Wilson Integrity Model 20

2.4 FORMAL LANGUAGES 21
2.4.1 Turnbull's Parsing 22

2.5 SYSTEM SPECIFICATION 24
2.5.1 Control of Discrete Event Systems 25
2.5.2 Alpern and Schneider's Liveness 25
2.5.3 Abadi and Lamport's Composing Specifications 26
2.5.4 Petri Nets 27

2.6 SUMMARY 28

3 MODULAR SPECIFICATION 31

3.1 INTRODUCTION 31

3.2 MODULAR REPRESENTATION 32
3.2.1 Basic Production Set 36
3.2.2 Prerequisite Relation 39
3.2.3 Delay Production Set 41
3.2.4 Shuffle Production Set 42
3.2.5 Multi-State Representation 44

3.3 DECIDABILITY ISSUES FOR MODULAR GRAMMARS 45

3.4 SINGLE-STATE MODULE EXAMPLES 46
3.4.1 Basic Module 47
3.4.2 Fork Module 49
3.4.3 AND Module 50
3.4.4 Spontaneous Module 51
3.4.5 Sink Module 52
3.4.6 General Module 52
3.4.7 Nondeterministic Fork 54
3.4.8 OR Module 55
3.4.9 Parallel Module 55
3.4.10 Clocked Input Module 57

3.5 MULTI-STATE MODULE EXAMPLES 59
3.5.1 Switch Module 59
3.5.2 Limited Buffer 60

3.6 SUMMARY 62

4 COMPOSITION OF MODULES 63

4.1 INTRODUCTION 63

4.2 FORMALIZING COMPOSITION 63
4.2.1 Specifying the Composition 64
4.2.2 Constructing the Composite 65
4.3 INTERNAL AND EXTERNAL EVENTS 75
4.4 DEADLOCK 76
4.5 COMPLEXITY ISSUES RELATED TO COMPOSITION 77
4.6 SINGLE-STATE COMPOSITION EXAMPLES 78
4.6.1 Simple Parallel Composition 78
4.6.2 Simple Serial Composition 81
4.6.3 Three Module Composition 85
4.6.4 Nondeterministic Example 87
4.6.5 Another Nondeterministic Example 88
4.7 MULTIPLE-STATE COMPOSITION EXAMPLES 90
4.7.1 Feedback Example 90
4.8 SUMMARY 92

5 PROPERTIES 93

5.1 INTRODUCTION 93
5.2 EVENT LABELS 93
5.3 MODULAR TRANSACTION PROPERTIES 97
5.4 COMPOSING WITH PROPERTIES 100
5.5 SUMMARY 104

6 APPLICATION EXAMPLES 105

6.1 INTRODUCTION 105
6.2 PREREQUISITE CONFIDENTIALITY 106
6.3 SUBJECTS AND OBJECTS 108
6.4 BELL-LAPADULA CONFIDENTIALITY 113
6.5 BIBA'S INTEGRITY 116
6.5.1 Strict Integrity 117
6.5.2 Low-Water Mark Policy on Objects 118
6.5.3 Low-Water Mark Policy on Subjects 120
6.6 DIGITAL LOGIC 121
6.7 NEEDHAM-SCHROEDER PROTOCOL 126

TABLE OF CONTENTS

6.8 SUMMARY 128

7 SUMMARY AND CONCLUSIONS 129

7.1 SUMMARY 129
7.2 CONCLUSIONS 130
7.3 DIRECTIONS FOR FUTURE WORK 132

Bibliography 135

LIST OF FIGURES

Figure 1.1: Relativistic Timing of Event Traces 4
Figure 2.1: Two Stack Machine 23
Figure 3.1: A Module with Two Blocks 33
Figure 3.2: Basic Module 47
Figure 3.3: Fork Module 49
Figure 3.4: AND Module 50
Figure 3.5: Spontaneous Module 51
Figure 3.6: Sink Module 52
Figure 3.7: m,n-General Module 52
Figure 3.8: Nondeterministic Fork Module 54
Figure 3.9: OR Module 55
Figure 3.10: Parallel Module 56
Figure 3.11: Clocked Input Module 57
Figure 3.12: Switch Module 59
Figure 4.1: Parallel Composition of Basic Modules 79
Figure 4.2: Parallel Composition with Internal Structures Removed 81
Figure 4.3: Serial Composition of Basic Modules 82
Figure 4.4: Serial Composition of Basic Modules with Internals Hidden 84
Figure 4.5: Composition of 1,3-GM with Two AND Modules 85
Figure 4.6: NDF Module Composed with Sink Module 87
Figure 4.7: OR Module Composed with NDF Module 89
Figure 4.8: OR Module with Fork Module Feedback 90
Figure 5.1: Addition of $k+1$ Connection 103
Figure 6.1: AND Module with Prerequisite Property 108
Figure 6.2: Object Modular Representation 111
Figure 6.3: Object Observation and Alteration Interface 112
Figure 6.4: Low-Water Mark Object Interface 119
Figure 6.5: Logical Inverter 122
Figure 6.6: Logical Inverter Timing Diagram 122
Figure 6.7: Logical Inverter State Diagram 122
Figure 6.8: Logical AND 123
Figure 6.9: Logical AND Timing Diagram 124
Figure 6.10: Logical AND State Diagram 124

[This page should be blank.]

List of Appendices

A GLOSSARY 139
B ANNOTATED ALGORITHMS 143

[This page should be blank.]

SYMBOL INDEX

$\{\alpha_1, \alpha_2, \ldots\}$	Set containing elements $\alpha_1, \alpha_2, \ldots$.
$x \in A$	x is an element of set A
$x \notin A$	x is not an element of set A
$\varnothing$	Empty set
$\|A\|$	Cardinality of (number of elements in) set A. $\|\varnothing\| = 0$.
$A \times B$	Cartesian product of sets A and B
$A \setminus B$	Complement of set B in set A
$A \cup B$	Union of sets A and B
$A \cap B$	Intersection of sets A and B
$[a,b]$	Set of integers between a and b inclusive (i.e. the interval $[a,b] = \{i : a \le i \le b\}$)
$\wedge$	Logical AND
$\vee$	Logical OR
$\neg$	Logical negation
$\forall$	For all (universal quantifier)
$\exists$	There exists (existential quantifier)
$\Leftrightarrow$	Logical equivalence
$\Rightarrow$	Logical implication
ε	Null (empty) string
A^*	Kleene closure of set A. A^* denotes the set of strings constructed by concatenating any number of elements from A.
A^+	Non-empty Kleene closure of set A. $A^+ = A^* \setminus \{\varepsilon\}$
$\rightarrow$	Produces (used in specification of production rules)
$\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$	α derives β in grammar G
$\alpha \underset{G}{\Rightarrow} \beta$	α directly derives β in grammar G
$\rhd$	Dominates operator for a partially-ordered set.
$\rhd_C$	Dominates operator for set of confidentiality labels.
$\rhd_I$	Dominates operator for set of integrity labels.

SYMBOL INDEX

ρ	Transitive event label relation $\rho \subseteq L \times L$.
${}^{l}\alpha_i$	Module input event with label l on input stream i.
${}^{l}\beta_j$	Module output event with label l on output stream j.
A_s	Start symbol for state s
b	Number of blocks comprising a module
$B_{k,s}$	Non-terminal symbol corresponding to block k for state s.
BP_s	Set of basic productions for state s. $BP_s = BPA_s \cup BPB_s \cup BPT_s \subseteq P_s$.
BPA_s	Set of common productions for state s. $BPA_s \subseteq BP_s \subseteq P_s$.
BPB_s	Set of block productions for state s. $BPB_s \subseteq BP_s \subseteq P_s$.
BPT_s	Set of transaction productions for state s. $BPT_s \subseteq BP_s \subseteq P_s$
connected	Connection relation between events.
DP_s	Set of delay productions for state s. $DP_s \subseteq P_s$.
E	Set of module event terminal symbols. $E = {}^{I}E \cup {}^{O}E$.
${}^{I}E$	Set of input event terminal symbols. ${}^{I}E^k$ is the set for module k.
${}^{O}E$	Set of output event terminal symbols. ${}^{O}E^k$ is the set for module k.
glb	Meet or greatest lower bound operator. If $l_A, l_B, l_C \in L$ and $l_C = \text{glb}(l_A, l_B)$ then $l_A \rhd l_C$ and $l_B \rhd l_C$ and $\neg\exists l_D \in L$ s.t. $l_A \rhd l_D, l_B \rhd l_D$, and $l_D \rhd l_C$.
I	Initial state for a module.
$L(G)$	Language generated by grammar G
$lhs(p)$	Left-hand side of production p
L	Set of event labels
L_C	Set of confidentiality (event) labels
L_I	Set of integrity (event) labels
m	Number of inputs to a module. m^k is the number for module k.
n	Number of outputs from a module. n^k is the number for module k.
P_s	Set of productions for state grammar G_s.
$prereq_s$	Prerequisite relation for state s
r_s	Interval specifying the possible number of transaction repetitions for state s
$rhs(p)$	Right-hand side of production p
S	Set of states for a module
SP_s	Set of shuffle productions for state s. $SP_s \subseteq P_s$.

SYMBOL INDEX

s.t.	Such that
$T_{k,s}$	Non-terminal symbol corresponding to a single complete transaction for block k while in state s.
V_s	Set of non-terminal symbols for state grammar G_s. $V_s = \{A_s\} \cup \bigcup_{k\in[1,b]} \{B_{k,s}, T_{k,s}\} \cup \hat{V}_s$
$\hat{V}_s$	Set of all input and output non-terminal symbols for state s. $\hat{V}_s = \bigcup_{\substack{l\in L \\ i\in[1,m]}} \{{}^{l}Y_{i,s}\} \cup \bigcup_{\substack{l\in L \\ j\in[1,n]}} \{{}^{l}Z_{j,s}\}$
${}^{l}Y_{i,s}$	Non-terminal symbol corresponding to input event terminal symbol ${}^{l}\alpha_i$ for state s.
${}^{l}Z_{j,s}$	Non-terminal symbol corresponding to output event terminal symbol ${}^{l}\beta_j$ for state s.

[This page should be blank.]

1 INTRODUCTION AND OVERVIEW

1.1 INTRODUCTION

Since large-scale computer systems were first introduced in the 1950s, four decades of rapid development have occurred. Systems have become more powerful, and at the same time have become extremely complicated in both design and functionality. To compound this, individual systems can now be interconnected with a variety of networks to allow data to move freely among systems. Information, particularly information that is stored electronically, is now one of our society's most valuable resources. Nearly all businesses, governments, and individuals depend on the availability of timely and accurate information to support their activities.

When we combine this dependence with the growth in the complexity of computer systems, it quickly becomes apparent that there is a great need to provide assurance that the guarantees made about how systems behave are valid (in an existing system) or achievable (in a system being designed). A great deal of research has investigated the problem of creating models of computer systems. Such models must be able to describe systems hierarchically, in order to address issues relating to how systems behave when connected together. They must also handle the true or apparent unpredictability or nondeterminism that is present in many systems. Most of the modelling approaches also include the ability to represent desired system properties and to decide whether specific systems have these properties. The remainder of this introductory chapter will provide background information on these fundamental issues, and then state the thesis of the work.

1.2 MODELLING

When the first computer systems were designed and built, it was relatively easy to verify that the systems had all of the properties intended by their designers. One could merely exercise the system through manual testing. As the complexity of the systems increased, and as different types of systems were interconnected, a convincing demonstration that a system had certain properties became difficult. Researchers turned to a variety of formal modelling techniques to describe systems and the desired properties. These techniques were often based on mathematical proofs that were used to increase confidence that the system goals were met.

No modelling strategy can provide complete assurance that a real system behaves in a certain way. The act of modelling itself involves a variety of simplifications of the real system in order to create a workable model. It always remains possible that a simplification made for modelling purposes will allow for invalid operation when the modelled system is implemented. The key is to show that the simplifications made are not unreasonable, and do not adversely affect the result of the exercise. In other words, recognizing an appropriate degree of abstraction is important in successfully representing a real system.

The first step in developing a good modelling technique for computer systems is to create a representation that reflects how a system is "put together" from its basic components. The second step is to create a means of describing the way that these components communicate or otherwise interact with each other and the surrounding environment.

1.3 COMPOSABILITY

The term *composability* is used to describe a situation in which a property fulfilled by an overall system is equivalent to properties of the individual components that make up the composite system. For example, one might consider two components, A and B, that satisfy properties P^A and P^B, respectively. By employing the notion of composability and by making certain assumptions about the environment, we may be able to conclude that the composition $A \cap B$ satisfies an (emergent) property P. Obviously, we must also make some *a priori* assumptions about the manner in which A and B are interconnected to form the composite system, before such conclusions may be reached. Notice that it is not necessary that $P = P^A = P^B$, although such a situation represents an important sub-class of composability problems that have received considerable attention. It is this sub-class that will receive the most attention in this thesis.

With this understanding, the utility of employing composable properties becomes apparent. Instead of having to deal with a large, complex system as a monolith, we can break it down into small sub-components that are more easily described and analyzed. These components can then be interconnected to form the composite system, and the desired property of the overall system follows by the composition principle. Another form of flexibility results when we consider an existing system to which a new component is to be added or an existing component is to be replaced or modified. If we have a composable property, it is only necessary to evaluate the new component and its connectivity to the original system. Non-composable properties and modelling approaches require a complete re-evaluation of the modified overall system in such cases. Even apparently standalone systems can be considered to be "composed" with the environment or external user with which they interact.[1]

1.4 EVENT TRACES

One way of representing the behaviour of a system uses *event traces*. In this approach, the significant happenings are represented by a time-ordered sequence of *events*. The modeller is free to choose those events that are of interest and ignore those that are not relevant to a particular analysis. What is considered "important" depends on the particular situation being analyzed.

In such a representation, the absolute timing of a particular event is not important, but its timing relative to the occurrence of other events in the

[1]This is an example of *closed-system* modelling, since the activities of the environment are included in the model. *Open-system* modelling concentrates on the system of interest, with little attempt to characterize the behaviour of the surrounding environment.

sequence is significant. If one wishes to introduce the notion of absolute timing into the event trace formalism, this can generally be done by creating "clock-tick" events. This relativistic timing notion corresponds closely to the practical operation of most real computing systems. For example, a typical C program running on a multi-user system is usually guaranteed to execute its statements in a particular order[2], but the absolute timing of the execution of each statement usually does not greatly concern the user.

Another way of describing the relativistic timing of event traces relates to the possible "real" executions that such traces capture. An event trace places no constraints on the absolute timing of particular events. All executions with the same relative ordering of events are captured and represented by a single event sequence. Thus, when we consider the properties of event sequences, we are really considering the properties of a whole class of executions. This is represented graphically in Figure 1.1. In this case, the labelled circles represent different events, and their absolute timing displayed on the scale from left to right. The first three sequences have identical event trace representations, "A, B, C, B", while the fourth sequence differs.

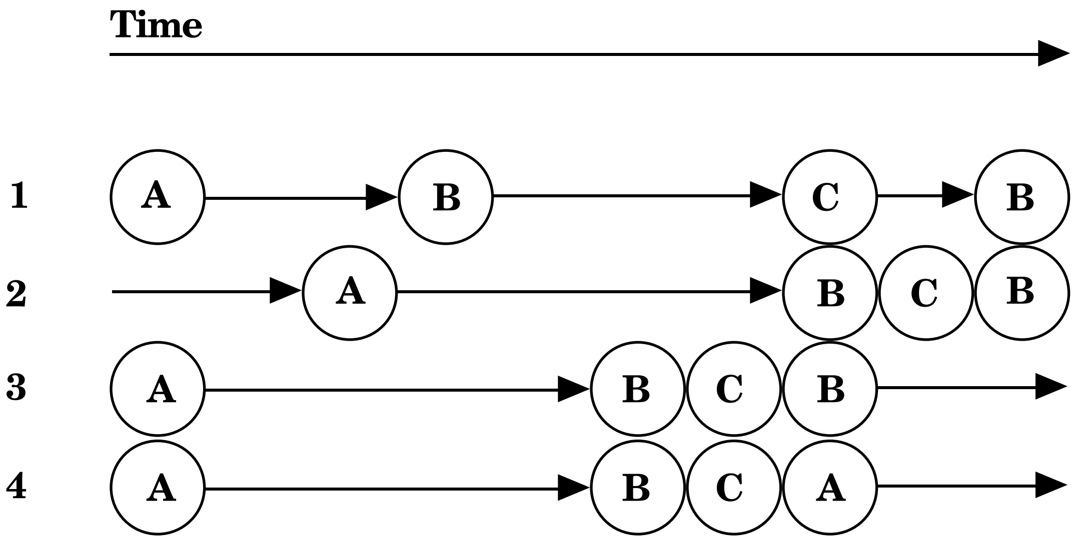

Figure 1.1: Relativistic Timing of Event Traces

[2]This would not be the case with programs coded in a language that supports parallel execution.

One advantage of the event trace approach is that it is capable of describing a system based only upon the externally visible behaviour. As a result, it does not implicitly constrain the internal behaviour of the modelled system in the same way that other methods (such as state machines) sometimes do.

However, a major problem is that the representational framework for event traces can be easily bogged down with unnecessary detail. To alleviate this, the specific representation that is chosen must allow the modeller to isolate those events that are particularly important or relevant to the modelling exercise, and pay little attention to the remainder.

Another problem relates to the loss of structure in event sequences. Without careful representational techniques, causal and temporal relationships between events are easily lost. What started out as the execution of a well-structured program can be turned into a completely unstructured sequence of events. Alleviating this problem requires the use of a finite representation of event sequences. One such representation uses well-known symbolic languages and grammars. A related technique for state-based representations uses a hierarchy of state machines to help manage the structure in a system's behavioural complexity.[3]

1.5 CAUSALITY

Given an event trace representation of a system, the most significant aspect to consider is how the various events interrelate. There are three possible classes of relationships between events that can be identified:

- *Unrelated events*: If event α is unrelated to event β, the occurrence of event α has no impact whatsoever on whether event β occurs.

- *Temporally related events*: If α and β are temporally related, then they must appear in a specific time order. It may, for instance, be required that event α happen before event β. We say that event α is a *prerequisite* for event β. As

[3]John P. Nestor, Security Modelling Using Hierarchical State Machines. Master's Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ontario, December 1990.

with all prerequisites, the occurrence of event α does not imply that β will occur in the future, but only that the occurrence of β becomes possible.

- *Causally related events*: If α and β are causally related, then there is a true functional relationship between them. The occurrence of event α implies that β is guaranteed to occur eventually. It may also be the case that another type of event can cause β, or that β might occur spontaneously.

In the development of this thesis, we do not differentiate between the latter two types of event relationships. Temporal and causal relationships are treated in the same manner in the development of the modelling methodology.

1.6 NONDETERMINISM

A deterministic system is one in which the inputs to the system completely determine the behaviour of the system. This is a strong definition of determinism, since it regulates *all* behaviour of the system, including internal activities that are not directly observable. Weaker definitions, such as that for *input-output determinism*,[4] require only that the inputs determine the visible output behaviour of the system.

A nondeterministic system is one in which the inputs do not completely determine the behaviour of the system, because the system contains some element of unpredictability.

The way in which a particular modelling strategy handles the existence of nondeterminism greatly affects its ability to represent real systems. Invalid conclusions can be drawn if the likelihood of nondeterministic events is not modelled accurately. On the other hand, a complete description of all possibilities may be imponderable, or may unduly complicate the working of the model.

[4]E.S. Lee, P.I.P. Boulton, B.W. Thomson, and R.E. Soper. Composable Trusted Systems. Technical Report, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, May 1992, p. 169.

This difficulty has led to two different classes of models that handle nondeterminism. *Possibilistic models* are concerned with whether a particular event can or cannot happen, regardless of the likelihood of the event. As a result, such models are often extreme[5] in their conclusions. *Probabilistic models* provide further information by using the probability distribution of the occurrence a particular event. Although their conclusions may be more realistic, it is difficult in practice to obtain the probability information such models require as input information.

1.7 THIS THESIS

Given the discussion contained in this chapter, it should be evident that any modelling methodology that is likely to be effective must have the following characteristics:

- Representations of real systems must be tractable, usually through some finite representation.
- The model must support composability in a useful way.
- The model must be able to represent relevant system behaviour while ignoring irrelevant activities.
- The model must maintain the causal and temporal relationships between events.
- It must be possible to represent a wide variety of properties under the model and to determine which properties are maintained under composition.
- The model must be able to represent nondeterministic behaviour.

The thesis of this work is that a grammatical approach can be used to maintain the overall structure of the event sequences, as well as allowing the causality between input and output events to be easily represented. The relationship between an overall operation and the sequence of sub-operations that implement

[5]Either too pessimistic or too optimistic.

it can be maintained within the grammatical structure. Given such a representation, a variety of desirable properties can be represented easily and their composability properties investigated and proven.

The first step in the development involves creating a grammatical structure capable of representing a variety of basic modular elements. Next, techniques for composing two or more of these modules are needed. Finally, a method for describing and composing grammatically defined abstract properties must be developed.

1.8 OVERVIEW

Chapter 2 reviews the relevant related research, including the areas of system modelling, trusted systems, and formal languages.

The next three chapters form the core of the work. Chapter 3 begins by describing how modular processing elements are represented. The notion of module grammars is introduced, followed by a number of example modules.

Chapter 4 continues by describing the technique for composing modules to form larger and more complex structures. A general, formal composition method is described. Various issues, including feedback, nondeterminism, and deadlock, are investigated and discussed through particular examples.

Chapter 5 begins the discussion of properties. A general technique for expressing properties through an extension of the grammatical representation is presented, followed by a proof of composability for a specific class of properties.

Chapter 6 considers the extension of the modelling framework to examples from the areas of safe systems, digital logic, and cryptographic protocols. Included is a discussion of confidentiality and integrity.

Finally, Chapter 7 summarizes the work and provides a number of conclusions. It also gives a number of directions for future work in the area.

In the presentation of the material in this thesis, a large number of special terms are introduced and defined. Many of these terms and their definitions are collected in Appendix A for convenient reference.

[This page should be blank.]

2 RELATED RESEARCH

2.1 INTRODUCTION

This chapter will introduce some of the related research in the areas of confidentiality, integrity, formal languages, and system specification. The emphasis will be placed on the research upon which this thesis was directly based.

2.2 CONFIDENTIALITY MODELS

Much of the initial work into formal modelling of computer systems and abstract properties was targeted towards a more specific problem—ensuring that systems were secure. In most of these cases security meant *confidentiality*—the property that information is not made available or disclosed to an unauthorized user, process or object. As the scope of the research grew, confidentiality became just one of several desired properties in an overall notion of computer security. However, many of the modelling techniques used originally for confidentiality apply to abstract properties in general. This section will review several of the existing formal approaches to confidentiality modelling, concentrating on the composability features of these models.

2.2.1 The Bell-LaPadula Model

One of the first significant works in the field of confidentiality modelling was produced by Bell and LaPadula during the middle 1970s.[6] Their BLP model is considered to be the earliest successful treatment of confidentiality, and was subsequently the basis of the U.S. Department of Defence criteria to be discussed in the next section.

---

[6] D. Elliott Bell and Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report TR-2547, MITRE Corporation, Bedford, MA, March 1973.

The BLP model is based upon requirements for access controls. All subjects (active entities) were permitted to read only objects (passive, data-storage entities) of equal or lower confidentiality level and were not permitted to perform any operations (like opening a file) that might allow them to write to or otherwise modify objects of lower confidentiality level. The two basic requirements were referred to as the "simple security property" and the "∗-property", respectively. In effect, data was prohibited from flowing from a high confidentiality level to a lower one. The resulting model is quite simple to use and allows the use of untrusted software, as long the externally imposed access control mechanisms work correctly.

Since the Bell-LaPadula model was introduced, several shortcomings have been noted in the literature. Systems invariably require trusted subjects, like memory managers, that must be permitted to violate the security properties. The model allows for such subjects, but provides no method for verifying their trustworthiness. Also, not all information can be easily described under the object representation. The simple notions of read and write operations do not adequately represent the often complex behaviour that real systems experience when, for example, executing a program or using a communication protocol. Finally, the BLP model does not support the composition of systems.

2.2.2 The Orange Book

In 1983, the National Computer Security Center of the United States produced the *Trusted Computer Security Evaluation Criteria*[7], also known as the TCSEC or "Orange Book". The Orange Book divides the fundamental security requirements into three groups, with nine underlying sub-requirements:

[7]National Computer Security Center, *Department of Defence Trusted Computer Security Evaluation Criteria*, DOD 5200.28-STD, December 1985.

Security Policy	Discretionary Access Control (DAC)
	Object Reuse
	Mandatory Access Control
	Labelling
Accountability	Identification and Authentication
	Audit
Assurance	Operational Assurance
	Life-Cycle Assurance
	Documentation

The Orange Book contains specifications for six classes of trusted systems: C1, C2, B1, B2, B3, and A1. Each class contains different combinations of specifications for the nine Security Policy, Accountability, and Assurance requirements. The classes are ordered from C1 through A1 such that each successive class contains all the security mechanisms of its predecessors.

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

2.2.3 Goguen and Meseguer's Noninterference

In 1982, Goguen and Meseguer introduced the notion of noninterference[8] as the basis for confidentiality. They proposed the following definition:

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

Goguen and Meseguer's noninterference model explicitly specifies a *capability table* that defines what users are permitted to do. They consider both the static case, in which users' capabilities are fixed, and the dynamic case, in which they may change as the system executes.

[8]Joseph A. Goguen and José Meseguer. Security Policies and Security Models. *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, April 1982, pp. 11–20.

A later paper[9] discussed a process for converting abstract, intuitive security policy statements into mathematical statements appropriate for input to a mechanical theorem-prover. This process, referred to as *unwinding*, was then used as the basis for a verification technique for security requirements based on noninterference assertions. The paper also considered the issue of *inference control*—ensuring that some highly classified information cannot be inferred by combining less classified data.

Their model is not completely general. It does not handle the behaviour of non-deterministic systems, and the composability properties are severely limited. In particular, the model requires that all process communicate through unbounded buffers, and that there be no merging of the outputs of processes. Some of these issues were later dealt with by McCullough with his notion of *generalized* noninterference[10].

Noninterference is a rigid prescription that often places unnecessary limitations on what users can do. For example, a cryptographic algorithm that accepts a high-level signal as input and generates a low-level encrypted version of it will fail, because regardless of the strength of the cryptography, the high-level input does interfere with the low-level output.

2.2.4 Sutherland's Deducibility

In 1986, David Sutherland took a different approach to solving the security modelling problem[11]. He made the claim that his model was "a generalization of the Goguen-Meseguer Model." The basis of the model is a standard state machine representation. Each distinct execution of the system being modelled is an element of the set of *possible worlds*. A piece of information about the system is represented by an *information function* whose domain is the set of possible

[9]Joseph A. Goguen and José Meseguer. Unwinding and Inference Control. *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, April 1984, pp. 75-86.
[10]Daryl McCullough. Noninterference and the Composability of Security Properties. *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988, pp. 177–186.
[11]David Sutherland. A Model of Information. *Proceedings of the 9th National Computer Security Conference,* 1986, pp. 175–183.

worlds. For example, one information function might represent all of the information that is available for observation by a particular user.

Sutherland then defines inference in terms of these concepts:[12]

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

In other words, information flows from f_1 to f_2 if knowing the value of f_1 rules out (or eliminates from consideration) even a single possible value of f_2.

Using these definitions, he formalized the notion of security. The binary relation $legal\_to\_get \subseteq \boldsymbol{I} \times \boldsymbol{I}$ (where $\boldsymbol{I}$ is the set of information functions) defines those entities that are entitled to obtain certain pieces of information, as required by the security policy. The security requirement was then stated as follows:[13]

> For all f_1 and f_2 in $\boldsymbol{I}$, if information flows from f_2 to f_1 then $legal\_to\_get(f_1, f_2)$.

Sutherland then instantiated the model using a state machine framework as its basis. One of the key information functions used is $view(l)$, that, given a possible world, returns the subsequence of the execution sequence consisting of those inputs[14] whose security levels are $\leq l$. Conversely, the $hidden\_from(l)$ function returns the subsequence of inputs whose levels are not $\leq l$. Security is then specified by making it illegal for information to flow from $hidden\_from(l)$ to $view(l)$ for any l. This essentially requires that every possible observation must be compatible with every conceivable secret. This prevents the observer from deducing the secret based on his or her observations.

---

[12]*Ibid.*, p. 176.
[13]*Ibid.*, p. 177.
[14]I.e., inputs to the state machine.

Deducibility is often thought to be excessively optimistic. If a high-level input is reproduced exactly as a low-level output, but with a, say, 1 in 10^9 chance that the output is not an accurate copy of the input, the system is deemed to be secure.

2.2.5 McCullough's Hook-Up Security and Restrictiveness

The first significant work in the area of security composition was done by Daryl McCullough in 1987.[15] McCullough's paper begins by discussing three popular models of multi-level security: Bell and LaPadula's access controls, Sutherland's deducibility model, and Goguen and Meseguer's non-interference requirement. In each case, he demonstrates that the models are inadequate as composable security properties.

McCullough's definition of *hook-up security* is based upon execution traces. Given a trace τ, modify it by adding or deleting some high-level inputs to result in a sequence σ, which is not necessarily a valid trace. It may be possible to construct a valid trace τ'' from σ by adding or deleting high-level outputs after the last input in, or immediately following, the modified input sequence. With this, hook-up security can then be defined as:

> A system is *hook-up secure* if, for every τ and for every σ that can be formed from it, at least one τ'' exists.

Intuitively, this requires that no high-level input can have an effect on low-level events and that the "fixing" of high-level outputs must wait until after any immediately following inputs.

The following theorem is then presented:

> If a system is hook-up secure, then it is deducibility secure, and it has the 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.

[15] Daryl McCullough. Specifications for Multi-Level Security and a Hook-Up Property. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987, pp. 161–166.

The validity of this result has been questioned in the literature[16]. In a later paper McCullough redefined his property as *restrictiveness*[17], and improved its abilities to handle nondeterministic systems.

2.2.6 Security and Machine Composition

In 1988, Johnson and Thayer presented a new security property[18] that is composable, but which is demonstrably weaker (less restrictive) than McCullough's hook-up security. Event traces are used as the basis for Johnson and Thayer's model. Informally, a *perturbation* of a trace is a sequence formed by inserting, modifying, or deleting high-level inputs in the trace. A *correction* of a sequence is a trace obtained by inserting, modifying, or deleting high-level non-inputs (i.e., outputs or internal events) in the sequence. A system is *correctable* if every perturbation has a correction that is a valid trace. In general, correctability is not preserved by composition.[19]

They then introduce a restricted form of correctability, denoted *n-forward correctability*, that is a composable property. The additional restrictions are as follows:

- It must be possible to correct every perturbation by modifying the sequence only after the perturbed part (if at all) to form a valid trace.
- If the perturbation happens to be immediately followed by n or fewer low-level inputs, it must be possible to delay any correction until after those inputs (and form a valid trace).

---

[16]E.S. Lee, P.I.P. Boulton, D.M. Lewis, M. Stumm, and B.W. Thomson. A Trusted Network Architecture. Technical Report, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, 1988.
[17]Daryl McCullough. Noninterference and the Composability of Security Properties. *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988, pp. 177–186.
[18]Dale M. Johnson and F. Javier Thayer. Security and the Composition of Machines. *Proceedings of the Security Foundations Workshop*, Franconia, NH, June 1988, pp. 72–89.
[19]*Ibid.*, p. 77.

2.2.7 Lin's Behavioural Security

In 1991, Ping Lin proposed a new security property, behavioural security, that is a form of noninterference[20]. A system algebra, based upon many-sorted algebras, was introduced to allow a representation of the input/output behaviour of systems and the effects of composition. The security property was then expressed within the system algebra to prevent the disclosure of both high-level inputs and outputs to low-level observers. The low-level behaviour of the system had to remain the same, regardless of whether high-level activity was present or not. It was proven that behavioural security was composable.

2.2.8 Nondeterminsitic Security Modelling

Researchers in the area of formal security models use the term "nondeterministic"[21] to refer to machines whose operation contains some element of unpredictability. In fact, there are a number of different types of nondeterminism that may occur. These were identified in the 1992 report by Lee et al[22].

Let T be the set of traces that characterize the behaviour of some system, and let $I(t)$ and $O(t)$ be the subsequences of input and output events in the trace $t \in T$, respectively. Lee et al present the following definitions:[23]

- A set of traces T is *strongly deterministic* iff, for any two traces t_1 and t_2, $I(t_1) = I(t_2)$ implies $t_1 = t_2$.

- A set of traces T is *input-output deterministic* iff, for any two traces t_1 and t_2, $I(t_1) = I(t_2)$ implies $O(t_1) = O(t_2)$.

---

[20]Ping Lin. The Composability of Behaviorally Secure Systems, Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, 1991.
[21]"Probabilistic" would be a more appropriate term.
[22]E.S. Lee, P.I.P. Boulton, B.W. Thomson, and R.E. Soper. Composable Trusted Systems. Technical Report 272, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, May 1992, pp. 168–169.
[23]*Ibid.*, p. 169.

- A prefix-closed set of traces T is *eventually deterministic* iff, for all pairs of traces $t_1 = \alpha\varepsilon$ and $t_2 = \alpha\gamma$, where α is some event sequence, and ε and γ are non-input events, implies $\gamma = \varepsilon$.

Sutherland's deducibility definition of security was the first that could be applied to nondeterministic systems. However, it makes the most optimistic assumptions about the statistical properties of the random behaviour of the system. It is a possibilistic model rather than a probabilistic one, since it does not consider the real likelihood of possible events.

The ability to represent nondeterministic systems is essential to any modern formal security definition. The difficulty in practice is that, to faithfully model nondeterministic behaviour, one requires a full characterization of every random process involved. The characterization might be statistical, probabilistic, possibilistic, or otherwise descriptive of a system's behaviour. Such information is rarely available. Therefore, a compromise must be struck between faithful representation and the tractability of the modelling exercise.

2.2.9 Parker's Foundations of Information Security

In 1991, Donn Parker introduced an expanded definition of information security[24]. He noted that the generally accepted security attributes, including confidentiality, integrity, and availability (CIA), are "incomplete … dangerously oversimplified, and inconsistent". He proposes to add two new attributes to the original CIA: authenticity and utility. *Authenticity* requires that every entity in the system, including users, data, programs, and the system itself, be authentically identified. *Utility* requires that information be in a form that is appropriate for the desired purpose. He states that the addition of these two attributes to the security notion is especially important for the business sector.

Whether or not one agrees with Parker's expanded security definition, his work does emphasize the fact that a modelling technique should be applicable to a wide variety of property objectives. Single-property models are of limited use

[24]Donn B. Parker. Restating the Foundation of Information Security. *Proceedings of the Second Annual North American System Security Symposium*, Toronto, Ontario, October 1991.

given the property interaction that is present in real systems and the different goals found in various system application domains.

2.3 INTEGRITY MODELS

The second area of computer security to receive attention was that of integrity. There has been considerable argument as to what "integrity" should mean in real computing systems, and what integrity controls are required. For the purpose of the discussion here, integrity involves issues of data quality, data consistency and the modification of data. This section will present two different approaches to dealing with the integrity problem.

2.3.1 Biba's Integrity Model

In his 1977 paper, K.J. Biba noted[25]:

> To date, security has been interpreted to address the authorized observation (dissemination) of classified information. We intend to extend the certified function of the utility to address the proper modification of classified information within computer systems.

Biba goes on to provide several different integrity policies, all based upon the same formalism as that used by Bell and LaPadula—namely, subjects and objects each with an appropriate integrity label. Several examples of Biba's integrity policies will be discussed in Section 6.4.

2.3.2 The Clark-Wilson Integrity Model

The Clark-Wilson "model"[26] is not a formal model. It is a set of high-level procedures and mechanisms intended to achieve a form of integrity control required by commercial data processing users. As a result, there is no mathematical basis that can be used to produce, for example, a formal process

---

[25] K.J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report TR-3153, MITRE Corporation, Bedford, MA, April 1977.

[26] D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1987, pp. 184–194. Also, D.D. Clark and D.R. Wilson. Evolution of a Model for Computer Integrity. *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, October 1988.

for composition. Nevertheless, the work done by Clark and Wilson has helped to improve the understanding of integrity issues by the security community.

2.4 FORMAL LANGUAGES

This section will attempt to briefly introduce some of the definitions and notation used to mathematically describe formal languages and grammars. This will form the basis for the material presented in the following sections. The definitions follow those used by Hopcroft and Ullman.[27]

A *grammar* G is denoted by the tuple $\langle V_N, V_T, P, S\rangle$. The symbols V_N, V_T, P, and S are, respectively, the *non-terminal symbols*, *terminal symbols*, *productions*, and *start symbol*. V_N, V_T, and P are finite sets. We assume that V_N and V_T contain no element in common; that is,[28]

$$V_N \cap V_T = \varnothing.$$

The union $V_N \cup V_T$ is often denoted by V. Let V^* denote the Kleene closure of V and $V^+ = V^* \setminus \{\varepsilon\}$. The set of productions P consists of expressions of the form $\alpha \to \beta$, where α is a string in V^+ and β is a string in V^*. Given a production $p \in P$ of the form $\alpha \to \beta$, we shall define $lhs(p) = \alpha$ and $rhs(p) = \beta$. Finally, S is always a symbol in V_N.

We now consider the language that a grammar $G = \langle V_N, V_T, P, S\rangle$ generates. We first need to define the relations $\underset{G}{\Rightarrow}$ and $\overset{*}{\underset{G}{\Rightarrow}}$ between strings in V^*. Specifically, if $\alpha \to \beta$ is a production of P and γ and δ are any strings in V^*, then[29] $\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta$. We say that the production $\alpha \to \beta$ is applied to the string $\gamma\alpha\delta$ to obtain $\gamma\beta\delta$. Thus $\underset{G}{\Rightarrow}$ relates two strings exactly when the second is obtained from the first by the applications of a single production.

---

[27]John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Series in Computer Science and Information Processing, Addison-Wesley, 1969.
[28] $\varnothing$ denotes the empty set.
[29]Say $\gamma\alpha\delta$ *directly derives* $\gamma\beta\delta$ in grammar G.

Suppose that $\alpha_1, \alpha_2, \ldots, \alpha_m$ are strings in V^*, and $\alpha_1 \underset{G}{\Rightarrow} \alpha_2, \alpha_2 \underset{G}{\Rightarrow} \alpha_3, \ldots, \alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$. Then we say[30] $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$. We say that for two strings α and β that $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ if we can obtain β from α by application of some number of productions in P. By convention, $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$ for each string α.

We define the *language generated by* G (denoted by $L(G)$) to be $\left\{ w \mid w \text{ is in } V_T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$. That is, a string is in $L(G)$ if and only if the string consists solely of terminals and can be derived from S. A string of terminals and non-terminals α is called a *sentential form* if $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$. Two grammars G^1 and G^2 are *equivalent* if $L(G^1) = L(G^2)$.

2.4.1 Turnbull's Parsing

In 1979, Christopher Turnbull proposed[31] a new hierarchy of languages, defined in terms related to programming languages and compilation. The classes of grammars were defined according to the ease by which sentences generated by the grammars could be parsed. Instead of basing the classes in terms of the form of rewriting rules, the new classes were defined in terms of constraints on the form and structure of canonical derivations.

To this end, Turnbull developed a new parser model called a *deterministic two stack machine* or D2SM, that is displayed in Figure 2.1.

---

[30] Say α_1 *derives* α_m in grammar G.

[31] Christopher J.M. Turnbull. Deterministic Left to Right Parsing. Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, 1974. Also Christopher J.M. Turnbull. Generalized Deterministic Left to Right Parsing. *Acta Informatica 12*, 1979, pp. 187–207.

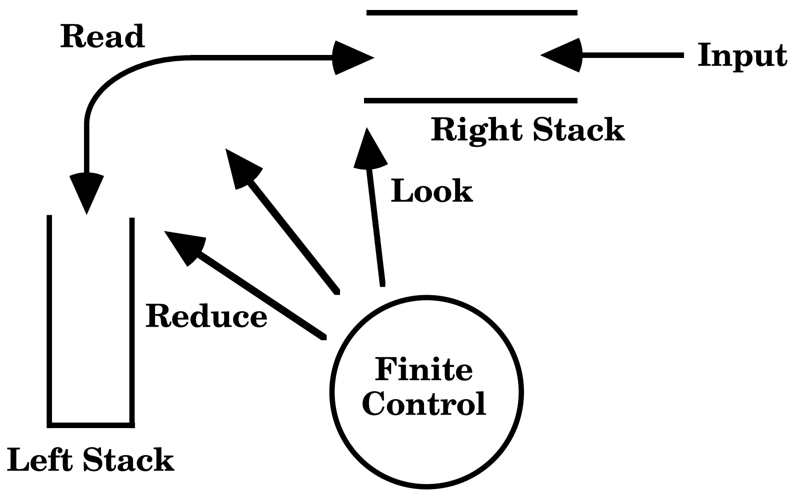

Figure 2.1: Two Stack Machine

The finite control is always in one of several possible states. Initially, the input is pushed into the Right Stack (which acts as an output restricted queue) and the control is in its initial state. The Left Stack is initially empty. At each step in its operations, the two stack machine may perform one of the following operations:

- *Read* — remove a symbol from the left end of the Right Stack, push that symbol into the Left Stack and change state.

- *Look ahead* — change state after looking at a bounded number of symbols at the left end of the Right Stack.

- *Reduce* — pop some symbols from the Left Stack, push some symbols into the left end of the Right Stack, and change state. The new symbols in the Right Stack have replaced the popped symbols and may be read or looked at in future steps.

- *Accept* — if the Left Stack is empty and the Right Stack contains the goal symbol, then accept the input.

- *Reject* — if none of the above moves can be made, the reject the input as being erroneous.

The D2SM model is an extension of the $LR(k)$ model[32] and has all of the standard properties of $LR(k)$ parsers. Such parsers:

1. halt on all inputs and accept exactly those strings that are generated by the grammar;
2. use a parsing algorithm that is deterministic and requires no backtracking;
3. perform a single left to right scan of the input; and,
4. detect errors as soon as possible in the parse.

Turnbull's approach to subdividing the class of type-0 grammars led to the development of the deterministic regular parsable (DRP) class. It was proven that DRP grammars were a proper superset of $LR(k)$ languages and suggested that DRP languages form a broad subset of type-0 languages. Turnbull proved that for any D2SM, a corresponding DRP grammar exists. He also demonstrated the converse. Turnbull then went on to describe a parser generator algorithm for DRP grammars, and showed that many of the techniques for handling $LR(k)$ grammars were directly applicable to DRP grammars.

2.5 SYSTEM SPECIFICATION

Having looked at the areas of confidentiality and integrity, we will now turn to the more general area of system specification. Here we are interested in abstract, mathematical specifications that describe the operating behaviour of systems. Such methods usually include a means for describing arbitrary interconnections of systems as well as some attempt at developing a notion of properties that can be expressed. This leads to the ability to provide proofs that the system representation fulfils the desired property.

---

[32] D.E. Knuth. On the Translation of Languages from Left to Right. *Information and Control 8*, 1965, pp. 607–639.

2.5.1 Control of Discrete Event Systems

During the late 1980s, Ramadge and Wonham developed a control theory for the logical aspects of Discrete Event Systems (DESs). This theory was subsequently extended (by them and others) to encompass a variety of control theoretic ideas such as controllability, observability, aggregation, and modular, decentralized and hierarchical control. An excellent review of this work is found in Ramadge and Wonham's 1989 survey paper.[33]

The theory includes both automata and formal language[34] models for DESs. The representations used are similar to those in the areas of confidentiality and integrity modelling discussed earlier. However, the kinds of properties being evaluated in the modelled systems are quite different. For example, one of the objectives in the control theory case is be to design a *supervisor* that selects control inputs in such a way that a given DES behaves in a certain desirable way. This is in contrast to the computer security domain, where systems are designed to withstand potentially malicious or unpredictable inputs from the surrounding environment.

2.5.2 Alpern and Schneider's Liveness

In 1985, Bowen Alpern and Fred Schneider proposed[35] a formal definition for liveness properties. Informally, the definition captured the notion that liveness properties stipulate that "something good" eventually happens during the execution of a system. This is contrasted with the definition of a safety property, that requires that "something bad" never happen.

Their work included a topological characterization of safety and liveness and a proof that every property can be expressed as the intersection of a safety property and a liveness property.

---

[33] Peter J.G. Ramadge and W. Murray Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

[34] Most of the theoretical results have been for the class of *regular languages*—those that have a finite state machine representation.

[35] Bowen Alpern and Fred B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4): 114–133, October 1985.

2.5.3 Abadi and Lamport's Composing Specifications

In their 1990 work, Abadi and Lamport culminated a series of efforts in the area of the abstract event system modelling of properties and the composability of systems[36]. They consider a specification of an interactive system to be an assertion that the system $\boldsymbol{\Pi}$ guarantees the desired property M only under the assumption that the system's environment satisfies some property E. Consider a system $\boldsymbol{\Pi}$ that is the composition of systems $\boldsymbol{\Pi}_1,\ldots,\boldsymbol{\Pi}_n$. It must then be proven that $\boldsymbol{\Pi}$ guarantees a property M under an environment assumption E, assuming that each $\boldsymbol{\Pi}_i$ satisfies a property M_i under environment assumption E_i. Abadi and Lamport first observe that:

1. It is expected that $\boldsymbol{\Pi}$ guarantee M only because of the properties guaranteed by its components. Therefore, it must be possible to infer that $\boldsymbol{\Pi}$ guarantees M from the assumption that each $\boldsymbol{\Pi}_i$ guarantees M_i.

2. The component $\boldsymbol{\Pi}_i$ guarantees M_i only under the assumption that its environment satisfies E_i; and $\boldsymbol{\Pi}_i$'s environment consists of $\boldsymbol{\Pi}$'s environment together with all the other components $\boldsymbol{\Pi}_j$. Therefore, it must be possible to infer E_i from the environment assumption E and the component guarantees M_j.

They then present the following (circular) *composition principle*:

> Let $\boldsymbol{\Pi}$ be the composition of $\boldsymbol{\Pi}_1,\ldots,\boldsymbol{\Pi}_n$, and let the following conditions hold.
>
> 1. $\boldsymbol{\Pi}$ guarantees M if each component $\boldsymbol{\Pi}_i$ guarantees M_i.
>
> 2. The environment assumption E_i of each component $\boldsymbol{\Pi}_i$ is satisfied if the environment of $\boldsymbol{\Pi}$ satisfies E and every $\boldsymbol{\Pi}_j$ satisfies M_j.
>
> 3. Every component $\boldsymbol{\Pi}_i$ guarantees M_i under environment assumption E_i.

---

[36]Martín Abadi and Leslie Lamport. Composing Specifications. Technical Report 66, Digital Systems Research Center, Palo Alto, California, October 1990.

Then $\boldsymbol{\Pi}$ guarantees M under environment assumption E.

They go on to demonstrate that this closure principle "is valid under suitably weak hypotheses, and that it provides a satisfactory rule for composing specifications."[37] They also claim that the above principle is extremely general in nature, since it does not assume any particular language or logic for writing specifications.

2.5.4 Petri Nets

Petri nets are a graphical and mathematical tool that are applicable to many different modelling areas. They are often used to study concurrent and asynchronous systems. By specifying appropriate state equations it is possible to mathematically model and study the behaviour of the modelled system.

A Petri net is specified by the following five components:

- P is a finite set of places,
- T is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $W : F \rightarrow \{1,2,3,\ldots\}$ is a weight function,
- $M_0 : P \rightarrow \{0,1,2,3,\ldots\}$ is the initial marking,
- $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.

The behaviour of many systems can be described in terms of system states. The marking of a Petri net corresponds to a particular state. The dynamic behaviour of the system is then specified with transition or firing rules that determine how the markings change. We can consider the markers as "tokens" that move around the Petri net according to the firing rules.

[37] *Ibid.*, p. 2.

One of the drawbacks of the Petri net model is that it does not allow one to easily determine whether an event sequence being considered could have been generated by a particular net.

There has been considerable research effort devoted to Petri nets and their applications. For an excellent review of this work, refer to Tadao Murata's 1989 work[38].

2.6 SUMMARY

This chapter began be reviewing research in the area of confidentiality models, particularly that directed at issues of composability. This provided a number examples of modelling techniques directed at a particular, important system property. A brief discussion of integrity properties was presented, followed by a review of formal grammars and languages. This included an important result relating to parsing context-sensitive languages. Finally, some important work in more general system specification and modelling was presented.

The models presented in this chapter all have shortcomings of one form or another. The Bell-LaPadula model was an important "first step" in the development of confidentiality, but is unable to deal with composition or the great complexity of real systems. Noninterference and deducibility both have limited use when nondeterministic systems are concerned. Later models have improved ability to handle composition and nondeterminism, but are difficult to use in practice because of tractability problems, or because the mathematical representations used are difficult to map to real system activity. Petri Nets allow the representation and analysis of systems for certain purposes, but do not provide a mechanism by which an event sequence can be identified as being valid or invalid. In addition, most of the models have been developed to handle specific properties, rather than a more general property notion.

As noted in Section 1.7, the modelling strategy proposed in this thesis will address all of these issues. It will handle both composition and nondeterminism

---

[38]Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

in a tractable manner. A general property concept will be supported. The overall result will be a modelling strategy that is better than any of those that existed previously.

Besides presenting particular modelling techniques, this chapter has also introduced some of the properties that we desire our systems to have. It is critical that, in addition to providing a better modelling framework, our model is able to support property goals that are of practical interest to those designing and verifying systems. Two such properties that have been introduced are confidentiality and integrity. Our model will be able to describe both of these properties as well as a number of others. This applicability will be fully described in Chapter 6.

[This page should be blank.]

3 MODULAR SPECIFICATION

3.1 INTRODUCTION

The first step in developing our modelling methodology is to define a technique for representing the basic "building blocks" of a system. These building blocks are conceptual objects that might map directly to "real" parts of a system (like hardware or software components) or might be related to abstract requirements placed on the system. As noted in the introductory chapter, we require a mathematically tractable technique for dealing with the complexity of an event system representation of components and systems.

We will ascribe the term *module* to the smallest, stand-alone representational element in our modelling methodology. Viewed from the external environment, a module has the ability to accept input events in several distinct streams, and to generate corresponding streams of output events. When modules are interconnected, the outputs of one module may become the input of another. Unattached streams of inputs come from the environment, while unattached streams of outputs are sent to the environment.

This chapter will begin by describing a grammatical approach that can be used to represent succinctly the behaviour of a module. Next, a number of simple single-state modules and their grammatical representations are presented. This will be followed by several more complicated multi-state examples. In Chapter 4, we will describe how modules can be interconnected to form more complicated specifications.

3.2 MODULAR REPRESENTATION

This section will introduce a specification for a module based on formal grammars. We will begin by informally describing some basic notions relating to the structure and operation of a module.

A module operates by accepting one or more *input events*, and possibly responding by generating one or more *output events*. Conceptually, input events to a module originate in the environment or from another module. Output events are generated by the module and sent somewhere else (either to another module or to the external environment). Each stream of input events will be called a *module input*, and each stream of output events will be called a *module output*. In modelling a particular system, the module inputs and outputs may not necessarily correspond to physical channels. The occurrence of certain input or output events might denote the completion of abstractly-defined operations or requirements.

Later in this section, we will develop the notion of a *module grammar*. Sentences in the language generated from the module grammar will be valid event sequences formed from a properly ordered succession of input and output events for the specified module.

We will use the term *complete transaction* to describe a complete set of input events and the corresponding, causally related output events at a particular module. A *partial transaction* is one in which output events are pending, or in which only a partial set of inputs has been provided to a module. Our module grammars will generate sentences (i.e., valid event sequences) that are made up of zero or more complete transactions.

It is assumed that the represented module is able to move through a set of possible *states*. The state concept in our modelling approach is quite different from that used in standard discrete event system modelling. We do not require a state transition after each event. In general, a module may stay in a single state for one or more complete transactions. The events that form a complete transaction are particular to each state that a module is in.

We will represent the behaviour of a module within a state with a *state grammar*. This sentences in the language generated by this grammar are each valid event sequences for the module while in that state. No interaction (i.e., intermixing of events) is permitted between the event sequences of different states. A module must complete all of its transactions for one state before it can proceed to another state.

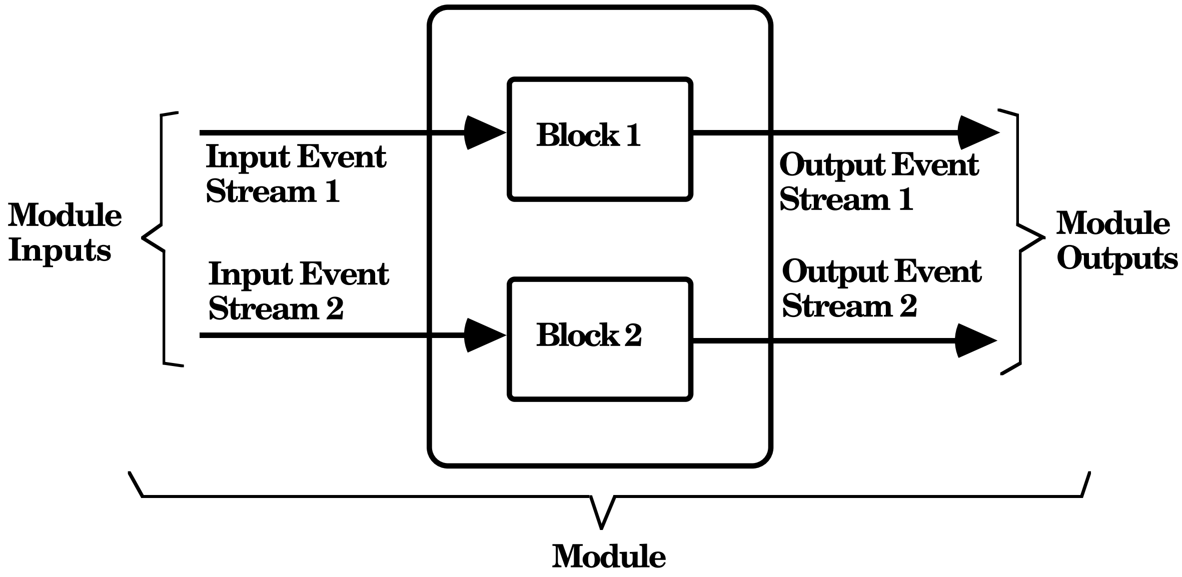

Figure 3.1: A Module with Two Blocks

In addition, a module may be composed of a number of causally independent substructures referred to as *blocks* (see Figure 3.1). A block can be thought of as a sub-module that has been composed with another sub-module, with no connections between them. Each block receives input events and generates output events independently of other blocks within the module. The events that make up a complete transaction are specific to each block. In this sense, the activities of one block have no effect whatsoever on the activities of another block within the module.[39] The definition of a block relates to the structure of a module and is therefore constant across the various states that a module might be in. However, the events that make up a complete transaction for a block are particular to each state that the module is in.

---

[39]Section 3.4.9 gives an example of the specification of a two block specification.

Although we consider the module to be the smallest, stand-alone element in our modelling methodology, it is necessary to maintain a module's block structure in order to allow composition with other modules. This will be further described in Section 4.2 when the composition procedure is introduced.

We will now proceed to formally define all of the above concepts. The presentation of the module specification will occur in two stages. First, the structure for a particular state will be introduced. Next, the single-state representation will be brought together under a global structure that describes the behaviour of the module as it moves between different states.

Let $m, n, b \in \{0,1,2,3,\ldots\}$ be the number of module inputs, module outputs, and blocks, respectively. We require that $b \leq m + n$, since each block must have at least one input or one output associated with it.[40]

For a module, define the terminal symbol α_i to correspond to a single input event on stream i and β_j to refer to a single output event on stream j. Each input and output stream is assigned a particular event symbol, corresponding to the subscript on the event symbol. We then define ${}^{I}E = \bigcup_{i \in [1,m]} \{\alpha_i\}$ and ${}^{O}E = \bigcup_{j \in [1,n]} \{\beta_j\}$ to be the set of input events and output events, respectively. Finally, we define $E = {}^{I}E \cup {}^{O}E = \bigcup_{i \in [1,m]} \{\alpha_i\} \cup \bigcup_{j \in [1,n]} \{\beta_j\}$ to be the complete set of terminal symbols corresponding to events that are relevant to the abstraction being built.

Let S be the set of states that the module can be in. Then the module specification for each state $s \in S$ has the following components:

- $prereq_s \subseteq E \times E$ the modular prerequisite relation used to identify temporal relationships between events in state s. The prerequisite relation is discussed further in Section 3.2.2.

[40] Because we are modelling externally visible behaviour, we do not allow blocks that have neither (externally visible) inputs nor outputs.

- $r_s \subseteq [0,\infty]$ an interval that defines the range of possible number of times complete transactions may occur in state s. If $i \in r_s$, then a sequence of i complete transactions is a valid event sequence for state s.

- $G_s = \langle V_s, E, P_s, A_s \rangle$ the grammar describing the possible event sequences in state $s \in S$. The sentences in the language $L(G_s)$ will be all possible sequences of events that are valid behaviours for the module while in state s.

The non-terminal set E corresponds to the set of possible events and is common to all state grammars. We will now define the remaining components of the grammar $G_s = \langle V_s, E, P_s, A_s \rangle$ for a particular state s.

For each input event terminal symbol α_i and each output event terminal symbol β_j, we define corresponding non-terminal symbols $Y_{i,s}$ and $Z_{j,s}$. While the event terminal symbols are common to all states, the corresponding event non-terminal symbols are state-specific, as denoted by the additional state subscript. The input and output non-terminals are state specific so that we can restrict the intermixing of events to within a single state.

We then define $\hat{V}_s = \bigcup_{i\in[1,m]} \{Y_{i,s}\} \cup \bigcup_{j\in[1,n]} \{Z_{j,s}\}$ to be the complete set of input and output non-terminal symbols for state s.

For each block $k \in b$ of the module, we define the block non-terminal symbol $B_{k,s}$. In the grammatical specification, a block non-terminal $B_{k,s}$ derives all valid event sequences for block k in state s. These valid event sequences are sequences of complete transactions. In the grammatical specification, the transaction non-terminal symbol $T_{k,s}$ derives a valid event sequence for a single complete transaction for block k in state s.

The non-terminal A_s is the start symbol for grammar G_s, and derives all possible event sequences for the module in state s. The complete non-terminal set for the grammar G_s is defined by $V_s = \{A_s\} \cup \bigcup_{k\in[1,b]} \{B_{k,s}, T_{k,s}\} \cup \hat{V}_s$.

The production set P_s will be presented in three distinct components: the basic production set BP_s, the delay production set DP_s, and the shuffle production set SP_s. It is possible to specify P_s directly, but having the set of productions broken into components will simplify the description of the composition algorithm in Section 4.2. Intuitively, having separate components also simplifies the analysis and verification of the more complex modules.

Once we have developed each of the three components, the complete set of productions for a particular module in state s is constructed by[41]

$$P_s = BP_s \cup DP_s \cup SP_s.$$

3.2.1 Basic Production Set

The *basic productions* (in the set BP_s) form a starting point in the grammatical development. They do not account for possible time-delay or reordering effects. The basic productions result in a language in which input event(s) are immediately followed by their corresponding output event(s).

As mentioned above, the non-terminal symbol $T_{k,s}$ will represent a single complete transaction for block k in state s. A particular block and state may have one or more possible transaction types, each corresponding to a production in BP_s with the non-terminal symbol $T_{k,s}$ on the left-hand side.

Having individual transactions clearly identified with a particular non-terminal (i.e., $T_{k,s}$) provides a convenient mechanism for examining the relationship between the inputs and the outputs in a particular (causally related) collection of events. This is probably the most important reason for using the grammatical approach to represent event sequences, and will be further explored when we consider properties in Chapter 5.

We will develop the set of basic productions BP_s in three parts, corresponding to the set of *common productions* BPA_s, the set of *block productions* BPB_s, and the

[41]Obviously, $BP_s \cap DP_s = \varnothing$, $DP_s \cap SP_s = \varnothing$, and $BP_s \cap SP_s = \varnothing$.

set of *transaction productions* BPT_s. The complete set of basic productions is then constructed by $BP_s = BPA_s \cup BPB_s \cup BPT_s$.

All modules include the following set of common productions:

$$BPA_s = \left\{A_s \to B_{1,s}B_{2,s}\ldots B_{b,s}\right\} \cup \bigcup_{i\in[1,m]}\left\{Y_{i,s} \to \alpha_i\right\} \cup \bigcup_{j\in[1,n]}\left\{Z_{j,s} \to \beta_j\right\}$$

In the set BPA_s, the symbol A_s is the goal symbol for state s. It derives all valid event sequences for that state. We mentioned earlier that each input and output event terminal symbol is associated with a particular state-specific input or output non-terminal symbol. This relationship is achieved through productions in BPA_s of the form $Y_{i,s} \to \alpha_i$ and $Z_{j,s} \to \beta_j$, respectively.

The block non-terminals $B_{k,s}$ are derived in terms of the transaction non-terminals $T_{k,s}$ in the block production set BPB_s. Only three symbol types may occur on the right-hand side of a $B_{k,s}$ production: one or more $T_{k,s}$ symbols, at most one $B_{k,s}$ symbol, or the empty string symbol ε . The structure of the set BPB_s depends on the set r_s. The following table provides some common cases:[42]

[42]The case where $r_s = [i,\infty]$ with $i \ge 2$ is not included in the table. Basic production sets can be easily constructed for this case, but additional non-terminal symbols are required.

r_s	BPB_s
$[0,0]$	$\bigcup_{k\in[1,b]}\{B_{k,s}\to\varepsilon\}$
$[0,1]$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to T_{k,s}\\ B_{k,s}\to\varepsilon\end{matrix}\right\}$
$[1,1]$	$\bigcup_{k\in[1,b]}\{B_{k,s}\to T_{k,s}\}$
$[1,2]$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to T_{k,s}\\ B_{k,s}\to T_{k,s}T_{k,s}\end{matrix}\right\}$
$[0,\infty]$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to T_{k,s}B_{k,s}\\ B_{k,s}\to\varepsilon\end{matrix}\right\}$
$[1,\infty]$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to T_{k,s}B_{k,s}\\ B_{k,s}\to T_{k,s}\end{matrix}\right\}$
$[i,j]$ $0\le i\le j<\infty$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to \overbrace{T_{k,s}T_{k,s}T_{k,s}\ldots T_{k,s}}^{i\text{ times}}\\ B_{k,s}\to \overbrace{T_{k,s}T_{k,s}T_{k,s}\ldots T_{k,s}T_{k,s}}^{i+1\text{ times}}\\ \vdots\\ B_{k,s}\to \overbrace{T_{k,s}T_{k,s}T_{k,s}\ldots T_{k,s}T_{k,s}T_{k,s}T_{k,s}T_{k,s}}^{j\text{ times}}\end{matrix}\right\}$
$[i,\infty]$ $0\le i<2$	$\bigcup_{k\in[1,b]}\left\{\begin{matrix}B_{k,s}\to T_{k,s}B_{k,s}\\ B_{k,s}\to \overbrace{T_{k,s}}^{i\text{ times}}\end{matrix}\right\}$

The transaction production set BPT_s defines the derivations of the transaction non-terminal symbols $T_{k,s}$, representing the possible complete transactions for block k of the module when in state s. The right hand side of each production in the set BPT_s is a sequence of symbols from the set $\hat{V}_s = \bigcup_{i\in[1,m]}\{Y_{i,s}\} \cup \bigcup_{j\in[1,n]}\{Z_{j,s}\}$, with the following restrictions:[43]

Requirement 3.1:

$$\forall s\in S, p\in BPT_s, \quad \exists y\in\left(\bigcup_{i\in[1,m]}\{Y_{i,s}\}\right)^*, z\in\left(\bigcup_{j\in[1,n]}\{Z_{j,s}\}\right)^* \quad \text{s.t.} \quad rhs(p)=yz$$

---

[43]The expression A^* indicates the Kleene closure of set A.

The symbols forming the right-hand side of any transaction production must be ordered so that all input non-terminal symbols appear before all output non-terminal symbols.

Requirement 3.2:

$$\forall s \in S,\ p \in BPT_s,\ v,x,z \in \hat{V}_s^*, w,y \in \hat{V}_s,$$
$$\left(rhs(p) = vwxyz\right) \Rightarrow w \neq y$$

Each of the non-terminal symbols from the set $\hat{V}_s = \bigcup_{i\in[1,m]}\{Y_{i,s}\} \cup \bigcup_{j\in[1,n]}\{Z_{j,s}\}$ may appear at most once in any single production. When we consider composition in Chapter 4, this requirement will greatly simplify our ability to merge module grammars.

As an example, all of the following are valid transaction productions according to the above requirements:

$$\begin{aligned}
T_{1,s_0} &\rightarrow Y_{1,s_0} Z_{1,s_0} Z_{2,s_0} \\
T_{4,s_1} &\rightarrow Y_{2,s_1} Y_{3,s_1} Y_{4,s_1} Z_{3,s_1} Z_{5,s_1} \\
T_{1,s_1} &\rightarrow Z_{2,s_1} Z_{4,s_1} \\
T_{2,s_3} &\rightarrow Y_{5,s_3}
\end{aligned}$$

The complete set of productions for the basic grammar is formed by $BP_s = BPA_s \cup BPB_s \cup BPT_s$.

3.2.2 Prerequisite Relation

The prerequisite relation $prereq_s$ is defined to indicate whether two events in a module are temporally related. For example, if $\langle \alpha, \beta \rangle \in prereq_s$, then event α must occur before event β can occur. We say that the occurrence of event α "is a prerequisite for the occurrence of" event β.

In the single module case, the prerequisite relation is analogous to a simple connectivity matrix. Therefore, we can consider a directed graph in which the vertices correspond to the terminal symbols, and a directed arc exists between vertex α and vertex β whenever $\langle \alpha, \beta \rangle \in prereq_s$.

The following requirements restrict the type of prerequisite relations that are valid under our specification technique.

Requirement 3.3:

$$\forall s \in S, \alpha \in E, \quad \langle \alpha, \alpha \rangle \notin prereq_s$$

No (externally visible) event is ever a prerequisite of itself.[44]

For convenience, we will sometimes use input or output non-terminals as arguments to the prerequisite relation. In this case, we define the value returned to be identical to the value that would be returned if the corresponding terminal symbols were used as arguments. In other words,

$$\begin{array}{ll} \forall s \in S, & \\ \quad \forall i_1, i_2 \in [1,m] & \left\langle \alpha_{i_1}, \alpha_{i_2} \right\rangle \in prereq_s \Leftrightarrow \left\langle Y_{i_1}, Y_{i_2} \right\rangle \in prereq_s \\ \forall i \in [1,m], j \in [1,n] & \left\langle \alpha_i, \beta_j \right\rangle \in prereq_s \Leftrightarrow \left\langle Y_i, Z_j \right\rangle \in prereq_s \\ \forall i \in [1,m], j \in [1,n] & \left\langle \beta_j, \alpha_i \right\rangle \in prereq_s \Leftrightarrow \left\langle Z_j, Y_i \right\rangle \in prereq_s \\ \quad \forall j_1, j_2 \in [1,n] & \left\langle \beta_{j_1}, \beta_{j_2} \right\rangle \in prereq_s \Leftrightarrow \left\langle Z_{j_1}, Z_{j_2} \right\rangle \in prereq_s \end{array}.$$

Requirement 3.4:

$$\begin{array}{l} \forall s \in S, \\ \quad \left(\beta \in {}^{O}E \wedge \alpha \in {}^{I}E\right) \Rightarrow \langle \beta, \alpha \rangle \notin prereq_s \end{array}$$

No (externally visible) output event can ever be a prerequisite for an (externally visible) input event.[45]

The transaction productions must be consistent with the prerequisite relation as defined above. If an input event non-terminal and an output event non-terminal are temporally related and appear in the right hand side of a transaction production, then $prereq_s$ must be defined appropriately.

---

[44] This is the case for all externally visible input and output events (i.e. all elements of the set E). When we consider feedback composition in Chapter 4, it will become possible for internal events to cause themselves.

[45] Once again, this requirement will not apply to internal events when composition is considered in Chapter 4.

Requirement 3.5:

$$\forall s \in S,\, w, y \in E,$$
$$\langle w, y \rangle \in prereq_s \Rightarrow \left(\exists p \in BPT_s,\, v, x, z \in \hat{V}_s^*, t, u \in \hat{V}_s \quad \text{s.t.} \quad rhs(p) = vtxuz \wedge t \underset{G_s}{\Rightarrow} w \wedge u \underset{G_s}{\Rightarrow} y \right)$$

If two events are temporally related, then there must exist a transaction production that enforces this relation. Events from different blocks cannot be temporally related.

We will now formalize a notion relating to the block structure. If we consider the graphical representation (mentioned at the beginning of Section 3.2.2) of temporal event ordering as defined by the $prereq_s$ relation, we can break the graph into weakly connected components. Each of these components corresponds to a block. In the specification of a modular grammar, blocks are defined by transaction productions that share common non-terminal symbols in their right-hand sides.

For example, a module with $b = 2$, transaction production set $BPT_s = \begin{Bmatrix} T_{1,s} \to Y_{1,s} Z_{1,s} \\ T_{2,s} \to Y_{2,s} Z_{2,s} \end{Bmatrix}$ and prerequisite relation with $\langle \alpha_1, \beta_1 \rangle \in prereq_s$ and $\langle \alpha_2, \beta_2 \rangle \in prereq_s$ has two separate blocks. This leads to the following general requirement:

Requirement 3.6:

$$\forall s \in S,\, y \in \hat{V},\, k_1, k_2 \in [1, b],\, p_1, p_2 \in BPT_s,\, v, w, x, z \in \hat{V}_s^*,$$
$$\left(lhs(p_1) = T_{k_1,s} \wedge lhs(p_2) = T_{k_2,s} \wedge rhs(p_1) = vyw \wedge rhs(p_2) = xyz \right) \Rightarrow k_1 = k_2$$

If two transaction productions have an input or an output non-terminal symbol in common in their right hand sides, then they must be part of the same block (i.e., have the same assigned block numbers).

3.2.3 Delay Production Set

To account for the queueing of input events, the basic productions must be augmented with one or more *delay productions* (contained in the set DP_s). In the various modules, delay productions are used to allow for situations in which an output event may be delayed until after several (later) input events have occurred. Such a delay might represent the propagation delay within a module or the amount of processing time required for a particular type of input event.

The set of delay productions is given by

$$DP_s = \bigcup_{\substack{i\in[1,m] \\ j\in[1,n]}} \left\{ Z_{j,s} Y_{i,s} \rightarrow Y_{i,s} Z_{j,s} \right\}.$$

An output may always be delayed past a later input while in a particular state.

3.2.4 Shuffle Production Set

Finally, there are the *shuffle productions* (the set SP_s). In more complicated modules and compositions, we may have multiple independent[46] input and output event sequences. Because the sequences are independent, the relative ordering of events from each sequence is not important. The shuffle rules allow these independent sequences to be freely intermixed. In effect, the shuffle productions are used to override artifacts of the particular ordering specified by the basic productions. Independent events must be ordered in a particular way in the transaction productions. The shuffle rules eliminate this original, fixed order where no ordering need be enforced.

The set of shuffle productions SP_s, is constructed through a two-stage process, as described in Algorithm 3.1. First shuffling within blocks is handled, followed by shuffling between blocks.

---

[46]By independent, we mean that there is no causal relationship between events in the two sequences *within the module*. For example, events representing the appearance of each of the operands to an adder circuit are independent in the sense understood here.

Algorithm 3.1: Single Module Shuffle Production Set Construction

$SP_s = \varnothing$

First add the productions within each block

$\forall p \in BPT_s, v,x,z \in \hat{V}_s^*, w,y \in \hat{V}_s$ s.t. $rhs(p) = vwxyz,$

$$\textbf{if } \left(w \notin \bigcup_{j\in[1,n]} \{Z_{j,s}\} \vee y \notin \bigcup_{i\in[1,m]} \{Y_{i,s}\} \right) \wedge \langle w,y \rangle \notin prereq_s \textbf{ then } SP_s = SP_s \cup \{wy \to yw\}$$

Now add the inter-block shuffle productions

$\forall k_1 \in [1,b-1], k_2 \in [k_1+1,b],\ p_1,p_2 \in BPT_s, u,w,x,z \in \hat{V}_s^*, v,y \in \hat{V}_s$

s.t. $lhs(p_1) = T_{k_1,s} \wedge lhs(p_2) = T_{k_2,s} \wedge rhs(p_1) = uvw \wedge rhs(p_2) = xyz,$

$$\textbf{if } \left(v \notin \bigcup_{j\in[1,n]} \{Z_{j,s}\} \vee y \notin \bigcup_{i\in[1,m]} \{Y_{i,s}\} \right) \textbf{ then } SP_s = SP_s \cup \{vy \to yv\}$$

The set of shuffle productions SP_s is based upon the set of transaction productions BPT_s and the prerequisite relation $prereq_s$. We must be careful when defining the set of shuffle productions. The percolation of events must occur in a single direction, so that circular derivation paths are not created. In addition, the grammar that results must be unambiguous, so that each valid sentence in the language has only a single derivation. Such constraints become important if one wishes to construct a parser that will recognize sentences generated by the module grammar.

The combined effect of the delay and shuffle productions is that output events can jump over both input events and output events, and input events can jump over other input events. Input events can jump over output events, but only when there is no possibility of violating temporal restrictions.

We can consider the basic grammars to be a normal form, with the addition of delay and shuffle productions being used to create variations on these forms. When we consider composition of grammars, we will create a composite normal form grammar from the elements of the composition, then add appropriate delay and shuffle production rules.

3.2.5 Multi-State Representation

The previous section developed the representation for a particular state $s \in S$. We will now go on to take the grammars for each of the individual states and connect them together to create a grammar $\boldsymbol{G}$ for the module as it moves from state to state.

There are a number of preliminary definitions required:

- $s_0 \in S$ the module's initial state.
- $N \subseteq S \times S$ the next state relation. If $\langle s_i, s_j \rangle \in N$ then the module can (validly) move from state s_i to state s_j.
- $\boldsymbol{G} = \langle \boldsymbol{V}, E, \boldsymbol{P}, \boldsymbol{A} \rangle$ the grammar describing the overall module. The derived components are defined as follows:
  - the non-terminal set $\boldsymbol{V} = \{\boldsymbol{A}\} \cup \bigcup_{s \in S} V_s$,
  - the production set $\boldsymbol{P} = \left\{\boldsymbol{A} \rightarrow A_{s_0}\right\} \cup \bigcup_{\langle s_i, s_j \rangle \in N} \left\{A_{s_i} \rightarrow A_{s_i} A_{s_j}\right\} \cup \bigcup_{s \in S} P_s$.

The set of terminal symbols for the global grammar is the same as that for each of the individual state grammars. The global grammar merely takes the goal symbols for each of the state grammars and concatenates them in a manner consistent with the next-state relation. The entire sets of non-terminals and productions from the individual state grammars[47] are included in the sets of non-terminals and productions for the overall grammar.

In this way, the overall grammar $\boldsymbol{G}$ generates a language whose sentences are valid event sequences as the module moves from state to state.

Given the above description of modular specification, it should be apparent that the minimum parameters for describing a module can be stated by the tuple

---

[47] I.e., the sets V_s and P_s, respectively.

$\langle m,n,b,S,N\rangle \cup \bigcup_{s\in S}\langle prereq_s, r_s, BPT_s\rangle$. Thus, by specifying the values for these parameters, we fully describe the module under the representation.

In the descriptions of the languages describing valid event (execution) sequences that follow, only those sequences that are made up of complete transactions will be generated by the grammars. In reality, the sets of valid event sequences are all *prefix closed*—any prefix of such a valid event sequence is also a valid event sequence. That is, if a particular event sequence is a valid one for a particular module, then all prefixes of that string (formed by removing one or more of the "tail" events) are also valid sequences. For simplicity, such incomplete sequences are not explicit members of the language representations.

3.3 DECIDABILITY ISSUES FOR MODULAR GRAMMARS

An important consideration in the development of a grammatical representation is the decidability of a number of important issues. It is known that some classes of languages or grammars have most relevant issues decidable, such as the nature of the language that results from language union and intersection. Another useful attribute is the ability to construct a deterministic parser to verify the validity of some event sequence under consideration. Turnbull showed that these issues are decidable for his DRP grammars. Practical and efficient parsers can be constructed to recognize sentences in DRP languages. In this section, we will consider the structure of the overall module grammar $\boldsymbol{G}$, introduced in Section 3.2.5.

The grammar, with all shuffle and delay productions removed and the addition of an end marker on the goal production, is clearly $LR(1)$. The productions are of two forms:

- Finite: Many of the productions in the grammar reduce to sentences that are either a single terminal or a finite string of terminals. The first terminal in each such sentence uniquely occupies this position.
- Right Recursive: Some of the productions take the form $S \to AS$. These are the only recursive productions in the grammar. Each of the symbols S in the

recursive productions admit sentential forms like $AAA\ldots AA$. Because of the form of the other productions, a lookahead of one symbol is always sufficient to identify the last A in one of these sentential forms.

These are consistent with $LR(1)$, because the lookahead that may be necessary to determine the required next move of the automaton in a parse is limited to one symbol. Because the terminals appear in a unique position, there will never be a need to look ahead more than one symbol.

The (context-sensitive) shuffle and delay productions have a common form of $ZY \rightarrow YZ$. They allow an event non-terminal symbol to interchange its place with a following event non-terminal symbol. The D2SM of Turnbull, described in Section 2.4.1, is ideally constructed to shift the input text for such cases from the right stack, into the left stack until the matching token is found, and then shift the text back to the right stack. Hence, a parser for our module grammar is a D2SM. Turnbull proves that a DRP grammar corresponds to each D2SM, so the connection between Turnbull's DRP grammars and our module grammars is easy and straightforward to make.

Finally, Turnbull showed that most questions that are decidable for $LR(k)$ are also decidable for DRP. Thus, it is always possible to construct a practical parser that will be able to verify whether an event sequence is valid for the system being modelled.

3.4 SINGLE-STATE MODULE EXAMPLES

This section will take the general representation framework described above and instantiate it for a number of particular single-state module examples. The examples are not exhaustive, but are presented to demonstrate the breadth of processing elements that can be represented under this framework.

3.4.1 Basic Module

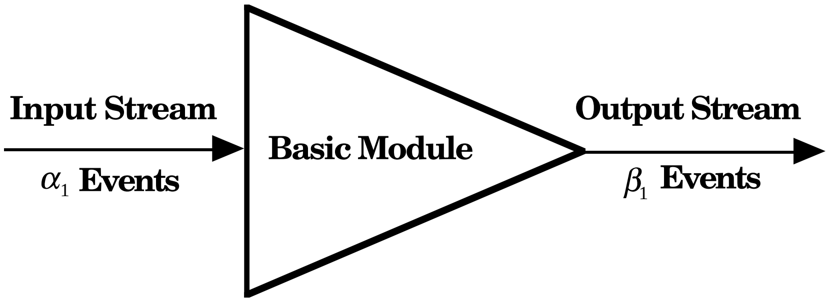

Figure 3.2: Basic Module

The Basic Module structure represents one of the simplest elements available in the abstract representational structure. A single input event, α_1, triggers its operation. After a possible processing delay, a corresponding output event, β_1, is generated. This delay may be a function of the specific input event provided as input to the module. The existence of such a delay will only be visible if it results in a reordering due to delay productions.[48] As a result, it is possible to queue input events in a Basic Module, while awaiting the occurrence of the corresponding output. Section 3.5.2 will provide an example demonstrating how such queuing can be limited if an input buffer size restriction exists.

Because of its general nature, a Basic Module may easily be used to represent a wide variety of real processing elements. For example, an input event might represent the triggering of the start of a mathematical calculation, and an output event indicates that the operation had completed. The following is the modular specification of the Basic Module:

$$m=1 \;\; n=1 \;\; b=1 \;\; S=\{s_0\} \;\; N=\varnothing$$

$$prereq_{s_0}=\{\langle\alpha_1,\beta_1\rangle\} \;\; r_{s_0}=[0,\infty] \;\; BPT_{s_0}=\{T_{1,s_0}\rightarrow Y_{1,s_0}Z_{1,s_0}\}$$

[48]This is an artifact of the relativistic timing nature of the event system representation, as noted in Section 1.4.

The derived values from the specification are:

$$E = \{\alpha_1, \beta_1\}$$

$$\boldsymbol{V} = \{\boldsymbol{A}\} \cup V_{s_0} \quad \boldsymbol{P} = \{\boldsymbol{A} \to A_{s_0}\} \cup P_{s_0}$$

$$V_{s_0} = \{A_{s_0}, B_{1,s_0}, T_{1,s_0}\} \cup \{Y_{1,s_0}, Z_{1,s_0}\} \quad BPA_{s_0} = \{A_{s_0} \to B_{1,s_0}\} \cup \begin{Bmatrix} Y_{1,s_0} \to \alpha_1 \\ Z_{1,s_0} \to \beta_1 \end{Bmatrix}$$

$$BPB_{s_0} = \begin{Bmatrix} B_{1,s_0} \to T_{1,s_0} B_{1,s_0} \\ B_{1,s_0} \to \varepsilon \end{Bmatrix} \quad DP_{s_0} = \{Z_{1,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{1,s_0}\} \quad SP_{s_0} = \varnothing$$

The module is defined to have a single input, a single output, and a single block. In addition, only a single state is defined—the next state relation is undefined, so the module stays in its initial state throughout its operation.

The basic productions generate sequences with zero or more complete transactions, in which input events (denoted by α_1) are immediately followed by their corresponding output event (denoted by β_1). The prerequisite relation reflects the dependence of β_1 events on α_1 events.

The following event sequences are examples of sentences in the language generated by the basic production set BP_{s_0}:[49]

$$\begin{gathered} \varepsilon \\ \alpha_1\beta_1 \\ \alpha_1\beta_1\alpha_1\beta_1\alpha_1\beta_1 \\ \alpha_1\beta_1\alpha_1\beta_1\alpha_1\beta_1\alpha_1\beta_1\alpha_1\beta_1 \end{gathered}$$

Next we consider the delay productions. Because of the possibility[50] that the output event corresponding to a particular input event may be delayed until after an arbitrary number of new input events have occurred, a new production must be introduced. In effect, the (context-sensitive) delay production set $DP_{s_0} = \{Z_{1,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{1,s_0}\}$ allows output events to "percolate" past the following input events. This makes sequences like the following possible:

---

[49]They are sentences generated by the grammar $G' = \langle V_{s_0}, E, BP_{s_0}, A_{s_0} \rangle$.

[50]If we wish to remove this possibility, we set $DP_{s_0} = \varnothing$.

$$\alpha_1\alpha_1\beta_1\beta_1\alpha_1\beta_1\alpha_1\alpha_1\alpha_1\beta_1\beta_1\beta_1$$
$$\alpha_1\alpha_1\alpha_1\alpha_1\beta_1\alpha_1\beta_1\beta_1\alpha_1\beta_1\alpha_1\beta_1\beta_1\beta_1.$$

We will take great advantage of this ability of context-sensitive grammars in the development of the grammatical representations of the various modules.

The grammar with delay productions DP_{s_0} guarantees sentences in which the number of output events is always equal to the number of input events. With this restriction on the event sequences, we have removed the possibility of *spontaneous events*. Such events are generated by the module itself, without any stimulus from the external environment (via the input channel). Since a single input event always occurs before each single output event, spontaneous output events are not possible in this particular module.

3.4.2 Fork Module

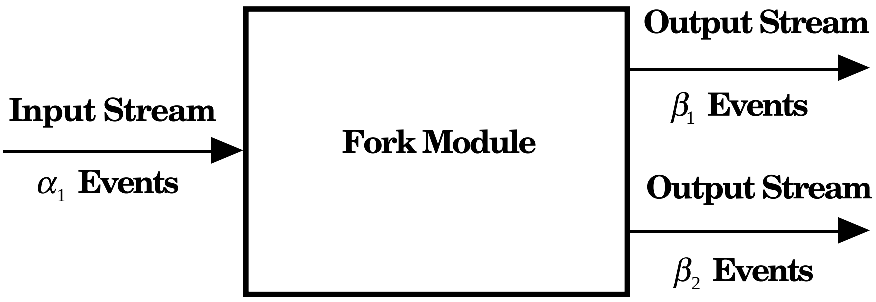

Figure 3.3: Fork Module

The Fork Module, upon receiving an input event, generates an output event on both of two separate streams. Either ordering of output events is allowed by the grammar. Like the Basic Module, input events may queue up before their corresponding output events are generated.

$$m = 1 \;\; n = 2 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \{\langle\alpha_1, \beta_1\rangle, \langle\alpha_1, \beta_2\rangle\} \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \{T_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} Z_{2,s_0}\}$$

The basic productions for the Fork Module generate a language whose sentences are series of zero or more complete transactions made up of a single input event followed by each of the two output events. In specification of the transaction production set BPT_{s_0} for the Fork Module, the β_1 output events are kept strictly before their corresponding β_2 output event[51].

The delay production set $DP_{s_0} = \begin{Bmatrix} Z_{1,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} \\ Z_{2,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{2,s_0} \end{Bmatrix}$ allows each type of output event to skip over later α_1 input events. The grammar does not permit output events to skip over other output events. Although the events are indistinguishable in this particular representation, preventing a reordering of output events is necessary in situations where the outputs are tagged to particular inputs, and a first-come first-served queueing method is required.

The shuffle production set $SP_{s_0} = \left\{ Z_{1,s_0} Z_{2,s_0} \to Z_{2,s_0} Z_{1,s_0} \right\}$ is defined to allow the ordering of the two types of output events to be switched. Such an allowance must be made since valid sentences like the following would otherwise not be allowed:

$$\begin{gathered} \alpha_1 \beta_2 \beta_1 \\ \alpha_1 \beta_1 \beta_2 \alpha_1 \beta_2 \beta_1 . \end{gathered}$$

The second of the two sentences demonstrates an important point. Just as the potential "processing delay" may be a function of the input event, so might the ordering of the two output events. The prerequisite relation allows the shuffle production, because $\langle \beta_1, \beta_2 \rangle \notin prereq_{s_0}$.

3.4.3 AND Module

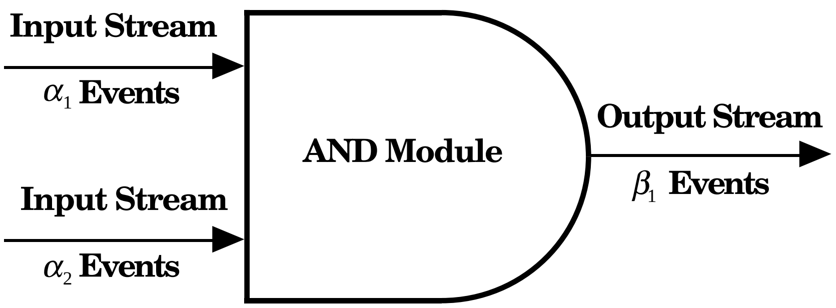

Figure 3.4: AND Module

---

[51] The choice of ordering is arbitrary, but consistent.

The AND Module awaits the occurrence, in either order, of an input event on each of its input lines. It then generates a corresponding output event. The specification is as follows:

$$m = 2 \;\; n = 1 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_1 \rangle\} \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \{T_{1,s_0} \to Y_{1,s_0} Y_{2,s_0} Z_{1,s_0}\}$$

As in the previous cases, the delay and shuffle production sets DP_{s_0} and SP_{s_0} allow the appropriate intermixing of input and output events.

3.4.4 Spontaneous Module

All the modules presented so far have required external input before any output is generated. This section will develop a simple module that produces spontaneous output, without any input.

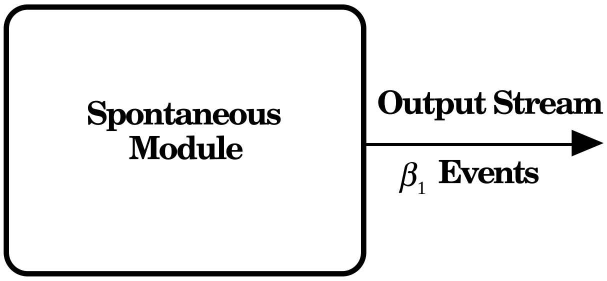

Figure 3.5: Spontaneous Module

The specification for this module is very simple:

$$m = 0 \;\; n = 1 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \varnothing \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \{T_{1,s_0} \to Z_{1,s_0}\}$$

In effect, the grammar for the Spontaneous Module merely generates sentences made up of strings of output events. This may seem trivial, but when composed with the Nondeterministic Fork module that will be described in Section 3.4.7, very powerful composite structures can be created.

3.4.5 Sink Module

The converse of the Spontaneous Module is the Sink Module. It merely consumes input events, producing no output events.

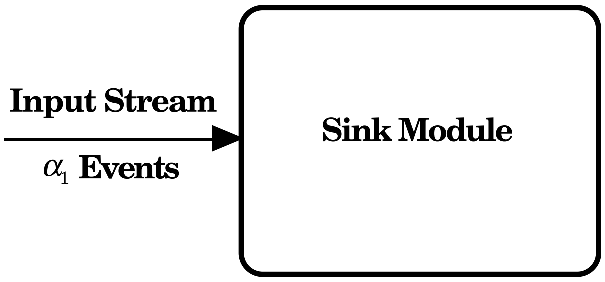

Figure 3.6: Sink Module

$$m = 1 \ \ n = 0 \ \ b = 1 \ \ S = \{s_0\} \ \ N = \varnothing$$

$$prereq_{s_0} = \varnothing \ \ r_{s_0} = [0, \infty] \ \ BPT_{s_0} = \{T_{1,s_0} \rightarrow Y_{1,s_0}\}$$

3.4.6 General Module

The five previous examples have been instances of a more general module class. This section will attempt to bring these examples under a general structure that has m inputs and n outputs. It will be referred to as the m,n-General Module (that we abbreviate as m,n-GM).

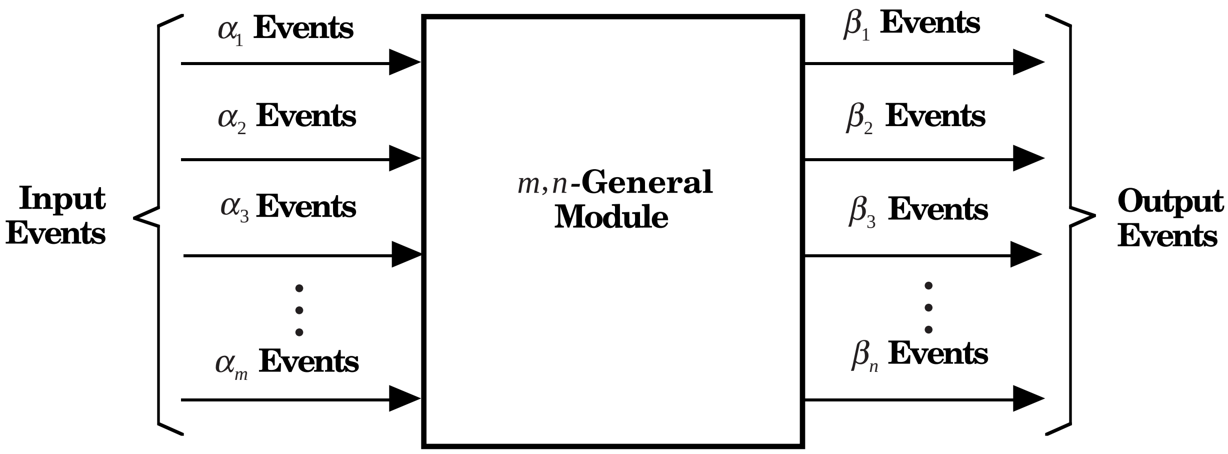

Figure 3.7: m,n-General Module

The module first awaits the arrival of each of the m input events, in any order. After all have arrived, it then generates the n output events, in an arbitrary order. The specification of this module is as follows:[52]

$$b=1 \;\; S=\{s_0\} \;\; N=\varnothing$$

$$prereq_{s_0} = \bigcup_{\substack{i\in[1,m]\\ j\in[1,n]}} \left\{\left\langle \alpha_i, \beta_j \right\rangle\right\} \;\; r_{s_0}=[0,\infty] \;\; BPT_{s_0} = \left\{T_{1,s_0} \to Y_{1,s_0} Y_{2,s_0} \ldots Y_{m,s_0} Z_{1,s_0} Z_{2,s_0} \ldots Z_{n,s_0}\right\}$$

The grammar assigns a single non-terminal symbol and a single terminal symbol to each input and output line of the module. The transaction production creates an ordering of both input and output events. The prerequisite relation is defined so that all inputs are prerequisites for all outputs.

The delay production set that results from the above specification is

$$DP_{s_0} = \bigcup_{\substack{i\in[1,m]\\ j\in[1,n]}} \left\{Z_{j,s_0} Y_{i,s_0} \to Y_{i,s_0} Z_{j,s_0}\right\}.$$

The delay productions allow any output event to be delayed past any input event, while the shuffle productions allow any ordering among both the input or the output events to be possible.

The resulting shuffle production set

$$SP_{s_0} = \bigcup_{\substack{i_1\in[1,m-1]\\ i_2\in[i_1+1,m]}} \left\{Y_{i_1} Y_{i_2} \to Y_{i_2} Y_{i_1}\right\} \cup \bigcup_{\substack{j_1\in[1,n-1]\\ j_2\in[j_1+1,n]}} \left\{Z_{j_1} Z_{j_2} \to Z_{j_2} Z_{j_1}\right\}$$

provides the minimum set of productions required to allow arbitrary ordering, without creating possible circular derivations. No shuffle productions in which an input event jumps over an output event are permitted, as required by the prerequisite relation.

[52]This is really a "meta-specification" since m and n are being treated as variables.

A wide variety of useful modular elements can be instantiated as particular cases of the General Module. The next group of modules to be presented fall outside those that can be represented by the m,n-GM structure.

3.4.7 Nondeterministic Fork

The Nondeterministic Fork (NDF) module, given a single input event, generates a corresponding output event on either of two output lines. The choice as to which output event is generated may be dependent on the input event, or may be arbitrary. The grammatical specification allows for both cases, by including two transaction productions in the set BPT_{s_0}.

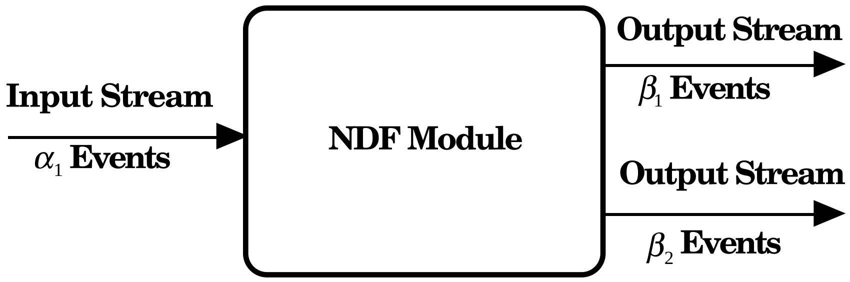

Figure 3.8: Nondeterministic Fork Module

$$m = 1 \;\; n = 2 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_1, \beta_2 \rangle\} \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \begin{Bmatrix} T_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} \\ T_{1,s_0} \to Y_{1,s_0} Z_{2,s_0} \end{Bmatrix}$$

The specification for this grammar is quite similar to the specification presented in Section 3.4.2 for the Fork Module. The difference is in the production set BPT_{s_0}.

3.4.8 OR Module

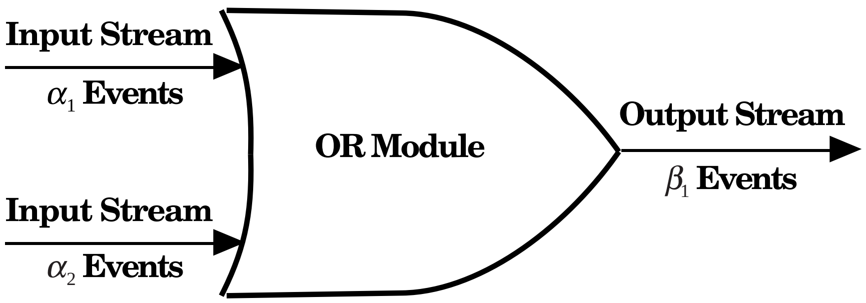

Figure 3.9: OR Module

The OR module awaits the occurrence of an input event on either of its input lines. It then generates a corresponding output event. The asynchronous nature of the specification makes it impossible for input events on each of the two lines to be accepted simultaneously. An output event is generated for *each* input event on either line. Thus, an α_2 event followed immediately by an α_1 event will generate two β_1 output events.

$$m = 2 \;\; n = 1 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_1 \rangle\} \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \begin{Bmatrix} T_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} \\ T_{1,s_0} \to Y_{2,s_0} Z_{1,s_0} \end{Bmatrix}$$

In the production set for the basic grammar, there are two transaction productions, representing the two possible types of complete transactions. They form a single block, so no inter-block shuffle productions are required. The causality relation is the same as that for the AND Module described previously, however the transaction productions change the way that this information is interpreted in constructing the shuffle production set.

3.4.9 Parallel Module

This section will present a module that is somewhat different from those presented so far. In this case, the module will have two blocks, thereby supporting two completely separate input and output streams. Given an input event on one of two input lines, it will produce an output event on the corresponding line. There is no interaction internally between the two sets of input/output lines. We shall see in Section 4.5.1 that the Parallel Module is identical in operation to the parallel composition of two independent Basic Modules.

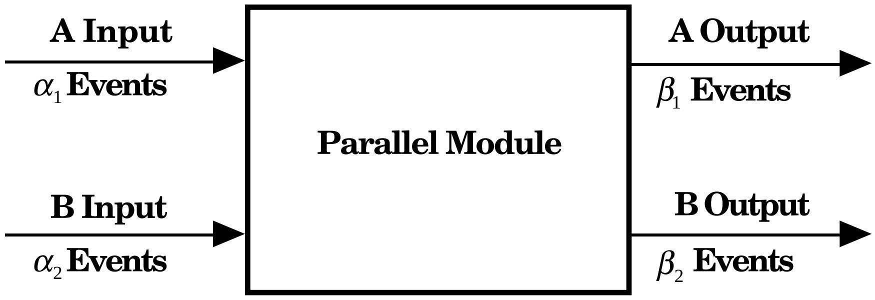

Figure 3.10: Parallel Module

The specification for the Parallel Module is as follows:

$$m = 2 \;\; n = 2 \;\; b = 2 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_2 \rangle\} \;\; r_{s_0} = [0, \infty] \;\; BPT_{s_0} = \begin{Bmatrix} T_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} \\ T_{2,s_0} \to Y_{2,s_0} Z_{2,s_0} \end{Bmatrix}$$

The delay production set

$$DP_{s_0} = \begin{Bmatrix} Z_{1,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{1,s_0} \\ Z_{1,s_0} Y_{2,s_0} \to Y_{2,s_0} Z_{1,s_0} \\ Z_{2,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{2,s_0} \\ Z_{2,s_0} Y_{2,s_0} \to Y_{2,s_0} Z_{2,s_0} \end{Bmatrix}$$

is typical of the other modules that have been described, except that we must allow shuffling of output events of the different blocks. Given the common production set

$$BPA_{s_0} = \{A_{s_0} \to B_{1,s_0} B_{2,s_0}\} \cup \begin{Bmatrix} Y_{1,s_0} \to \alpha_1 & Z_{1,s_0} \to \beta_1 \\ Y_{2,s_0} \to \alpha_2 & Z_{2,s_0} \to \beta_2 \end{Bmatrix},$$

the shuffle production set

$$SP_{s_0} = \begin{Bmatrix} Y_{1,s_0} Y_{2,s_0} \rightarrow Y_{2,s_0} Y_{1,s_0} \\ Y_{1,s_0} Z_{2,s_0} \rightarrow Z_{2,s_0} Y_{1,s_0} \\ Z_{1,s_0} Z_{2,s_0} \rightarrow Z_{2,s_0} Z_{1,s_0} \end{Bmatrix}$$

allows full reordering between the two separate streams of events. This is somewhat different from the situation in the Fork Module, for example, where shuffle productions within a block were used to override the arbitrary ordering in a particular transaction production. The extra shuffle productions are required here because the module is composed of two separate blocks.

Since the common production set keeps all block 1 transactions strictly before block 2 transactions, the shuffle productions allow full intermixing while guaranteeing unique derivations.

The shuffle production set shown above is one of two that are possible, depending on how the transaction productions are assigned their block numbers. The languages generated are equivalent.

3.4.10 Clocked Input Module

This section will model the clocked input module that appears in Figure 3.11.

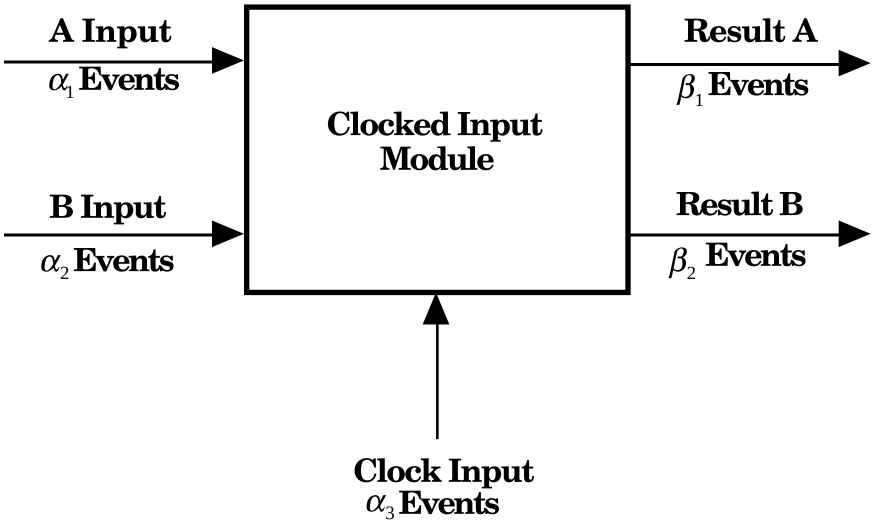

Figure 3.11: Clocked Input Module

In this particular situation, each of the two input events (perhaps corresponding to the arrival of the two operands in a calculation) must occur before the clock event, that might be equivalent to a "data ready" signal. Previous modules allowed input events to be presented in an arbitrary order, but here the module requires a particular ordering. The two output "result" events are allowed to occur in either order. Again, it is possible to queue several input-event triples, awaiting the corresponding output events. The specification for the module is as follows:

$$m = 3 \;\; n = 2 \;\; b = 1 \;\; S = \{s_0\} \;\; N = \varnothing$$

$$prereq_{s_0} = \bigcup_{\substack{i \in [1,3] \\ j \in [1,2]}} \{\langle \alpha_i, \beta_j \rangle\} \cup \{\langle \alpha_1, \alpha_3 \rangle, \langle \alpha_2, \alpha_3 \rangle\} \;\; r_{s_0} = [0, \infty]$$

$$BPT_{s_0} = \{T_{1,s_0} \to Y_{1,s_0} Y_{2,s_0} Y_{3,s_0} Z_{1,s_0} Z_{2,s_0}\}$$

The delay production set

$$DP_{s_0} = \bigcup_{\substack{i \in [1,3] \\ j \in [1,2]}} \{Z_j Y_i \to Y_i Z_j\}$$

allows each of the two output event types to skip over each of the three input event types. The shuffle production set

$$SP_{s_0} = \begin{Bmatrix} Y_{1,s_0} Y_{2,s_0} \to Y_{2,s_0} Y_{1,s_0} \\ Z_{1,s_0} Z_{2,s_0} \to Z_{2,s_0} Z_{1,s_0} \end{Bmatrix}$$

allows the output and input event ordering to be switched. The clock event, however, must occur after the two input events and before either output event, for a given single transaction. This is indicated by the extra two prerequisite relation entries, and reflected in the shuffle production set. This is the first example that has been presented where an input is a prerequisite of another input. An analogous technique could be used among output events to enforce a particular ordering.

This example demonstrates the flexibility of the prerequisite relation in allowing the modeller to specify any desired temporal relationship between events.

3.5 MULTI-STATE MODULE EXAMPLES

This section will instantiate the representation framework for two multiple-state examples.

3.5.1 Switch Module

Given a single input event, the Switch Module generates a corresponding output event on either of two output lines. The choice as to which output event is generated is dependent on whether an even or an odd number of toggle events have occurred.

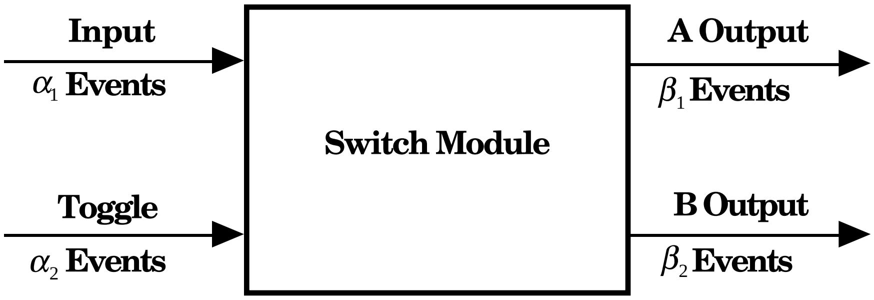

Figure 3.12: Switch Module

In this case, we need a total of four states: two states to represent the two operating modes, in which a α_1 event occurs causing either one of the output events, and two states to represent the toggle event transitions between the two modes.

$$m=2\ \ n=2\ \ b=1\ \ S=\{s_0,s_1,s_2,s_3\}\ \ N=\{\langle s_0,s_1\rangle,\langle s_1,s_2\rangle,\langle s_2,s_3\rangle,\langle s_3,s_0\rangle\}$$

$$prereq_{s_0}=\{\langle\alpha_1,\beta_1\rangle\}\ \ r_{s_0}=[0,\infty]\ \ BPT_{s_0}=\{T_{1,s_0}\rightarrow Y_{1,s_0}Z_{1,s_0}\}$$

$$prereq_{s_1}=\varnothing\ \ r_{s_1}=[1,1]\ \ BPT_{s_1}=\{T_{1,s_1}\rightarrow Y_{2,s_1}\}$$

$$prereq_{s_2}=\{\langle\alpha_1,\beta_2\rangle\}\ \ r_{s_2}=[0,\infty]\ \ BPT_{s_2}=\{T_{1,s_2}\rightarrow Y_{1,s_2}Z_{2,s_2}\}$$

$$prereq_{s_3}=\varnothing\ \ r_{s_3}=[1,1]\ \ BPT_{s_3}=\{T_{1,s_3}\rightarrow Y_{2,s_3}\}$$

The module begins in state s_0, in which it repeatedly accepts α_1 input events and generates β_1 output events. If a α_2 toggle input event is received, the module moves into state s_1, and then directly into state s_2. In state s_2, each α_1 input event causes a β_2 output event. If a α_2 toggle input event is received, the module moves into state s_3, and then directly into state s_0, where the cycle begins again.

3.5.2 Limited Buffer

All the previous examples have allowed the input events to a particular module to queue up indefinitely. This is not a realistic representation of many practical systems, since usually the number of possible pending operations is limited. In this section, we will consider a Basic Module that has been modified so that it has an input buffer of size $ibs<\infty$. This means that only ibs "unprocessed" input events are allowed to queue up at any given time, before the corresponding output events are generated.

The specification for the Limited Buffer Module is:

$$m=1\ \ n=1\ \ b=1\ \ S=\{s_0,s_1\}\ \ N=\{\langle s_0,s_1\rangle,\langle s_1,s_0\rangle\}$$

$$prereq_{s_0}=\{\langle\alpha_1,\beta_1\rangle\}\ \ r_{s_0}=[0,ibs]\ \ BPT_{s_0}=\{T_{1,s_0}\rightarrow Y_{1,s_0}Z_{1,s_0}\}$$

$$prereq_{s_1}=\{\langle\alpha_1,\beta_1\rangle\}\ \ r_{s_1}=[0,ibs]\ \ BPT_{s_1}=\{T_{1,s_1}\rightarrow Y_{1,s_1}Z_{1,s_1}\}$$

The derived values from the specification are:

$$E = \{\alpha_1, \beta_1\}$$

$$\boldsymbol{V} = \{\boldsymbol{A}\} \cup V_{s_0} \cup V_{s_1} \quad \boldsymbol{P} = \left\{ \begin{array}{l} \boldsymbol{A} \to A_{s_0} \\ A_{s_0} \to A_{s_0} A_{s_1} \\ A_{s_1} \to A_{s_1} A_{s_0} \end{array} \right\} \cup P_{s_0} \cup P_{s_1}$$

$$V_{s_0} = \{A_{s_0}, B_{1,s_0}, T_{1,s_0}\} \cup \{Y_{1,s_0}, Z_{1,s_0}\} \quad BPA_{s_0} = \{A_{s_0} \to B_{1,s_0}\} \cup \left\{ \begin{array}{l} Y_{1,s_0} \to \alpha_1 \\ Z_{1,s_0} \to \beta_1 \end{array} \right\}$$

$$BPB_{s_0} = \left\{ \begin{array}{l} B_{1,s_0} \to \varepsilon \\ B_{1,s_0} \to T_{1,s_0} \\ B_{1,s_0} \to T_{1,s_0} T_{1,s_0} \\ \quad \vdots \\ B_{1,s_0} \to \overbrace{T_{1,s_0} T_{1,s_0} \ldots T_{1,s_0}}^{ibs \text{ times}} \end{array} \right\} \quad DP_{s_0} = \{Z_{1,s_0} Y_{1,s_0} \to Y_{1,s_0} Z_{1,s_0}\} \quad SP_{s_0} = \varnothing$$

$$V_{s_1} = \{A_{s_1}, B_{1,s_1}, T_{1,s_1}\} \cup \{Y_{1,s_1}, Z_{1,s_1}\} \quad BPA_{s_1} = \{A_{s_1} \to B_{1,s_1}\} \cup \left\{ \begin{array}{l} Y_{1,s_1} \to \alpha_1 \\ Z_{1,s_1} \to \beta_1 \end{array} \right\}$$

$$BPB_{s_1} = \left\{ \begin{array}{l} B_{1,s_1} \to \varepsilon \\ B_{1,s_1} \to T_{1,s_1} \\ B_{1,s_1} \to T_{1,s_1} T_{1,s_1} \\ \quad \vdots \\ B_{1,s_1} \to \overbrace{T_{1,s_1} T_{1,s_1} \ldots T_{1,s_1}}^{ibs \text{ times}} \end{array} \right\} \quad DP_{s_1} = \{Z_{1,s_1} Y_{1,s_1} \to Y_{1,s_1} Z_{1,s_1}\} \quad SP_{s_0} = \varnothing$$

Here, we have defined a module with two separate states, each of which operates in an identical manner. The module has a single block. The block production sets BPB_s are defined so that up to *ibs* input-output event pairs may occur before the transition is made to the other state.

The delay productions allow outputs to be delayed, but only within the events of each state. Thus, an output event is never more than *ibs* steps away from the input event that caused it. It can be shown that as *ibs* grows large, the language generated by the limited buffer specification approaches that for the unlimited case described in Section 3.4.1.

3.6 SUMMARY

This chapter began by introducing a module representation based upon formal grammars and an event prerequisite relation. It then described a number of modules under this representation, ranging from the simple Basic Module to more complicated examples.

At the core of the representation are the module grammars. The overall behaviour of the module is represented by a grammar $\boldsymbol{G}$. Within each state s, a grammar G_s is defined to describe allowed event sequences within that state. The basic production set BP_s defines a canonical form for allowed event sequences in which output events occur immediately after input events, in a prescribed ordering. The delay productions (in DP_s) allow output events to be delayed past later input events. The shuffle productions (in SP_s) allow reordering of events where no temporal restriction results from the definition of the $prereq_s$ relation.

A good representational technique should allow the modelling of a wide range of practical situations. Through the general nature of our technique, it should be clear that we can easily model virtually any component that reacts to input events by generating output events. Most real systems can be represented in this way.

4 COMPOSITION OF MODULES

4.1 INTRODUCTION

As noted earlier, the purpose of this work is to produce a generalized framework for representing the behaviour of modular processing elements. Chapter 3 introduced a structure for describing the activities of individual modules. We will now consider how these individual elements can be interconnected or composed to form much larger and more complex structures such that the specification of the composition can be deduced from the specifications of the components.

We will begin by presenting a general framework for an arbitrary composition of the specifications of two separate modules. This will provide us with a set of transformation rules that will describe a mechanical procedure for composing arbitrary specifications of modules. The impact of composition on internal and external events will be described and the potential for deadlock considered. Finally, we will provide a number of composition examples, beginning with simple situations and proceeding to more complex ones. In each case, we will use the composition method to construct an appropriate composition grammar, based upon the grammars of the individual modules and the desired interconnection.

4.2 FORMALIZING COMPOSITION

This section will describe a formal method for composing the modular specifications described in Section 3.2. The first step in this process involves describing the elements of the composition: the individual module specifications plus the connections that are made between them. The second step is to construct the composite specification.

4.2.1 Specifying the Composition

Consider two modules:

$$\left\langle m^1, n^1, b^1, S^1, N^1 \right\rangle \cup \bigcup_{s \in S^2} \left\langle prereq_s^1, r_s^1, BPT_s^1 \right\rangle$$
$$\left\langle m^2, n^2, b^2, S^2, N^2 \right\rangle \cup \bigcup_{s \in S^2} \left\langle prereq_s^2, r_s^2, BPT_s^2 \right\rangle.$$

All of the other model elements are derived as described in Section 3.2, with the addition of superscripts to indicate module number.

The connections between the modules will be defined by a connection function $connected \subseteq \bigcup_{k \in [1,2]} {}^{O}E^k \times \bigcup_{k \in [1,2]} {}^{I}E^k$. We have that $\left\langle e_1, e_2 \right\rangle \in connected$ if (output) event e_1 has been made equivalent to (input) event e_2 as a result of a connection. Connections between modules associate two previously independent events as a single event. Therefore, a connection involves no storage, queueing or delay. The definition of the connection function does not rule out the possibility that a module's output is connected to one of its own inputs.

Each module input or output can participate in at most one connection. Therefore, we have the following requirement[53].

Requirement 4.1:

$$\begin{aligned} &\forall e_1, e_2 \in \bigcup_{k \in [1,2]} {}^{O}E^k, \quad e_3, e_4 \in \bigcup_{k \in [1,2]} {}^{I}E^k, \\ &\quad \left(\left\langle e_1, e_3 \right\rangle \in connected \wedge \left\langle e_1, e_4 \right\rangle \in connected \right) \Rightarrow e_3 = e_4 . \\ &\quad \left(\left\langle e_1, e_3 \right\rangle \in connected \wedge \left\langle e_2, e_3 \right\rangle \in connected \right) \Rightarrow e_1 = e_2 \end{aligned}$$

Those output lines that do not participate in a connection send their output events to the external environment. Likewise, those inputs that do not participate in a connection receive their input events from the external environment. If we wish to allow external access to connected events, we can do so by inserting Fork or OR Modules rather than simple connections.

---

[53]The first part of the requirement follows directly from the fact that $connected$ is a function.

Based upon the connections, we will construct the composite specification

$$\left\langle m^C, n^C, b^C, S^C, N^C \right\rangle \cup \bigcup_{s \in S^C} \left\langle prereq_s^C, r_s^C, BPT_s^C \right\rangle.$$

4.2.2 Constructing the Composite

The process of composing grammatically represented modules involves a number of steps. Informally, these are:

- Ensuring that the modules to be composed have independent sets of terminals and non-terminals. This may involve a trivial renaming of the elements of these two sets. In our case, we will use module number indices to ensure unique symbols. We must also remove those terminal and non-terminal symbols that correspond to the endpoints in a connection.

- Generating the hybrid prerequisite relation, based upon the prerequisite relations for the individual modules and the desired connections between modules.

- Generating the composite grammar by collecting all the transaction productions together, creating "hybrid" transaction productions for those that have a causal relationship as a result of a connection between modules. The composite block production set must respect the number of transactions specified by each of the individual block production sets.

- Adding delay and shuffle productions to allow intermixing of the event sequences of the modules, making sure that causal relationships are not violated. These productions must allow all valid sequences to be generated, without introducing loops into the derivation sequence.

We shall now take this informal process and derive a corresponding mathematically based method for composing groups of arbitrary modules. We begin by considering the specification of the state-independent components of the composite specification. The composite system has the total number of inputs and outputs of the two separate modules, reduced by the number of connections between them:

$$m^C = m^1 + m^2 - |connected|$$
$$n^C = n^1 + n^2 - |connected|$$

Each connection "consumes" one input and one output.

We can define the composite set of states as $S^C = \left(S^1 \times S^2\right) \cup s_0^C$. The extra initial state s_0^C may be necessary in certain cases involving feedback loops where we need to identify the state before any events have occurred.[54] If such an initial state is not required, then $S^C = \left(S^1 \times S^2\right)$ and the initial state for the composite is $\left\langle s_0^1, s_0^2 \right\rangle$. The composite next-state relation $N^C \subseteq S^C \times S^C$ can be defined by the following:

$$\begin{aligned}&\forall s_1, s_2 \in S^1,\ s_3, s_4 \in S^2,\\ &\left\langle \langle s_1, s_3 \rangle, \langle s_2, s_4 \rangle \right\rangle \in N^C \Leftrightarrow \begin{pmatrix} \left((s_1 = s_2) \wedge \left(\langle s_3, s_4 \rangle \in N^2\right)\right) \\ \vee\left((s_3 = s_4) \wedge \left(\langle s_1, s_2 \rangle \in N^1\right)\right) \\ \vee\left(\left(\langle s_1, s_2 \rangle \in N^1\right) \wedge \left(\langle s_3, s_4 \rangle \in N^2\right)\right) \end{pmatrix}.\end{aligned}$$

If the extra initial state is required,[2] then we must add the transition $\left\langle s_0^C, \left\langle s_0^1, s_0^2 \right\rangle \right\rangle$ to the next-state relation N^C.

Either one or both of the modules in a composition may undergo a (valid) state transition as part of a single valid composite system state transition.

The composite event set

$$E^C = \bigcup_{k \in [1,2]} E^k \quad \setminus \quad \bigcup_{\substack{k_1, k_2 \in [1,2] \\ i \in [1, m^{k_2}] \\ j \in [1, n^{k_1}] \\ \left\langle \beta_j^{k_1}, \alpha_i^{k_2} \right\rangle \in connected}} \left\{ \alpha_i^{k_2}, \beta_j^{k_1} \right\}$$

is defined so that it includes all events from the individual modules, with those events involved in connections removed. For convenience, we use the names of

[54]This will be explained further when we construct the composite, later in this section.

the event terminal and non-terminal symbols as they were in the individual modules. After the composition has been completed, the symbols can easily be renumbered and reindexed to fit into the standard structure described in Section 3.2.

We now consider the construction of the composite prerequisite relation $prereq_s^C \subseteq \bigcup_{k\in[1,2]} E^k \times \bigcup_{k\in[1,2]} E^k$. To begin, we define the relation P by

$$\begin{aligned}&\forall\langle s_1,s_2\rangle \in S^C, \alpha,\beta \in E^1 \cup E^2,\\ &\quad \langle\alpha,\beta\rangle \in P \Leftrightarrow \langle\alpha,\beta\rangle \in prereq_{s_1}^1 \vee \langle\alpha,\beta\rangle \in prereq_{s_2}^2 \vee \langle\alpha,\beta\rangle \in connected\cdot\end{aligned}$$

We can then define the recursive support relation $prereq*_s^C$ to be the transitive closure of P:

$$\begin{aligned}&\forall\langle s_1,s_2\rangle \in S^C, \alpha,\beta \in E^1 \cup E^2,\\ &\quad \langle\alpha,\beta\rangle \in prereq*_{\langle s_1,s_2\rangle}^C\\ &\qquad \Leftrightarrow \begin{pmatrix}\langle\alpha,\beta\rangle \in P\\ \vee\exists\gamma \in E^1 \cup E^2, \quad \text{s.t.} \quad \langle\alpha,\gamma\rangle \in P \wedge \langle\gamma,\beta\rangle \in prereq*_{\langle s_1,s_2\rangle}^C\end{pmatrix}\\ &\qquad \Leftrightarrow \begin{pmatrix}\langle\alpha,\beta\rangle \in prereq_{s_1}^1 \vee \langle\alpha,\beta\rangle \in prereq_{s_2}^2 \vee \langle\alpha,\beta\rangle \in connected\\ \vee \begin{pmatrix}\exists\gamma \in E^1 \cup E^2, \quad \text{s.t.}\\ \begin{pmatrix}\langle\alpha,\gamma\rangle \in prereq_{s_1}^1 \vee \langle\alpha,\gamma\rangle \in prereq_{s_2}^2\\ \vee\langle\alpha,\gamma\rangle \in connected\end{pmatrix} \wedge \langle\gamma,\beta\rangle \in prereq*_{\langle s_1,s_2\rangle}^C\end{pmatrix}\end{pmatrix}\end{aligned}$$

In other words, event α is a prerequisite of event β in the composite system if

- event α is a prerequisite of event β in any of the individual modules, or
- event α is connected to event β as defined by the *connected* function, or
- there exists an intermediate event γ such that
  - α is a prerequisite of γ in an individual module, or
  - α is connected to γ

  and γ is a (previously defined) prerequisite of β in the composite system.

This definition is analogous to a recursive definition of connectivity. Now, if $\exists\alpha$ s.t. $\langle\alpha,\alpha\rangle \in prereq*^{C}_{\langle s_0^1, s_0^2\rangle}$, indicating that there is a feedback loop, then we require the special initial state s_0^C.

Since $E^C \subseteq E^1 \cup E^2$, once we have the definition for $prereq*^C_s$, we must define

$$prereq^C_s = \left\{\langle\alpha,\beta\rangle \,\middle|\, \langle\alpha,\beta\rangle \in prereq*^C_s \wedge \alpha,\beta \in E^C\right\}$$

We now define the composite grammars $G^C_s = \left\langle V^C_s, E^C, P^C_s, A^C_s \right\rangle$ for each state $s \in S^C$.

We cannot determine the number of blocks in the composite system until after we have calculated the composite transaction sets.

Algorithm 4.1 will iteratively construct the sets $BPT^C_s, s \in S^C$ based upon the individual transaction production sets $BPT^k_s, k \in [1,2], s \in S^k$ and the *connected* function. The algorithm is broken into two parts: the unification procedure for a single connection in Algorithm 4.1a, and the mainline code that loops through the various states and connections, calling the unification procedure as required, in Algorithm 4.1b. An extensively annotated version of this algorithm appears in Appendix B.

The purpose of the algorithm is to create the composite transaction production set $BPT^C_{\langle s_1, s_2\rangle}$, based on the transaction production sets for the individual modules, and the *connected* function. For each connection (defined by the corresponding output-input event pair), the unification procedure is called. The unification procedure first isolates those productions that contain one or both of the non-terminals corresponding to the endpoints of the connection being considered. It then creates the unified transaction productions by concatenating the right-hand sides of each pair of transaction productions, removing the endpoint non-terminal symbols in the process. This process is repeated for each pair of productions involving the connection endpoint non-terminal symbols.

The algorithm calls upon three support functions that are defined as follows:

$substring(p, s)$

A Boolean function that returns true if there exist (possibly empty) strings y and z such that $ypz = s$.

$delstring(p, s)$

A function that returns the string formed when the first occurrence of p is removed from s. If $\neg substring(p, s)$ then $delstring(p, s) = s$.

$remdups(s)$

A function that returns the string formed when all single character duplicates are removed from the string s. The operation is performed with the following code:

```
function remdups(s : string) : string
    tempstring := ""
    for i: 1 .. length(s)
        if ¬substring(s(i), tempstring) then tempstring := tempstring + s(i)
    end for
    result tempstring
end function
```

In this code, $s(i)$ returns the ith character of string s, $length(s)$ returns the length of string s in characters, and the + operator is used for string concatenation.

In the following algorithm, the “strings” that are being manipulated are sequences of non-terminal symbols.

Algorithm 4.1a: Unification Procedure

procedure $Unify(z, y)$

$outProds, inProds, bothProds := \varnothing$

$\forall x \in BPT^C_{\langle s_1, s_2 \rangle}$,

if $substring(z, rhs(x)) \wedge substring(y, rhs(x))$ **then**

$bothProds := bothProds \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow rhs(x) \right\}$

elsif $substring(z, rhs(x))$ **then** $outProds := outProds \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow rhs(x) \right\}$

elsif $substring(y, rhs(x))$ **then** $inProds := inProds \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow rhs(x) \right\}$

if $bothProds = \varnothing \wedge \left((outProds \neq \varnothing \wedge inProds = \varnothing) \vee (outProds = \varnothing \wedge inProds \neq \varnothing) \right)$

then $badstate := badstate \cup \langle s_1, s_2 \rangle$, **exit procedure**

$\forall p_{out} \in outProds, p_{in} \in inProds$,

$$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow remdups \begin{pmatrix} delstring(z, rhs(p_{out})) \\ +delstring(y, rhs(p_{in})) \end{pmatrix} \right\} \setminus p_{in}$$

$\forall p_{out} \in outProds, p_{both} \in bothProds$,

$$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow remdups \begin{pmatrix} delstring(z, rhs(p_{out})) \\ +delstring(y, delstring(z, rhs(p_{both}))) \end{pmatrix} \right\}$$

$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \setminus outProds$

$\forall p_{both} \in bothProds$,

$$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \rightarrow remdups\left(delstring\left(y, delstring\left(rhs(p_{both}), z \right) \right) \right) \right\} \setminus p_{both}$$

end $Unify$

Algorithm 4.1b: Transaction Production Unification (Mainline Code)

$badstate := \varnothing$
$\forall \langle s_1, s_2 \rangle \in S^C$

$BPT^C_{\langle s_1, s_2 \rangle} := \varnothing$

$\forall k \in [1,2], x \in BPT^k_{s_k},$
$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \to rhs(x) \right\}$

$\forall k_1, k_2 \in [1,2], j \in [1, n^{k_1}], i \in [1, m^{k_2}],$
if $\langle \beta^{k_1}_j, \alpha^{k_2}_i \rangle \in connected$ **then** $Unify(Z^{k_1}_{j,s_1}, Y^{k_2}_{i,s_2})$

$\forall x \in BPT^C_{\langle s_1, s_2 \rangle},$
$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \to sortseq(rhs(x)) \right\} \setminus x$

The above algorithm does not assign the appropriate block numbers to the transactions in the composite transaction production set $BPT^C_{\langle s_1, s_2 \rangle}$. However, calculating b^C and assigning the appropriate block numbers can easily be made using Requirement 3.6, after Algorithm 4.1 has completed.

The algorithm works by first copying all of the transaction productions for the individual modules to the composite transaction production set, $BPT^C_{\langle s_1, s_2 \rangle}$, setting their left-hand sides to $T^C_{1,\langle s_1, s_2 \rangle}$ in the process.

For each connection (defined by the corresponding output-input event pair), the unification procedure is called. The procedure first isolates those productions that contain one or both of the non-terminals corresponding to the endpoints of the connection being considered. It then creates the unified transaction productions by concatenating the right-hand sides of each pair of transaction productions containing a non-terminal symbol corresponding to an endpoint.

After all the unifications have been performed, the algorithm sorts non-terminals in the right-hand side of each of the composite transactions by module and line number, with all input non-terminals occurring before all output non-terminals. For example, $sortseq(Z_2^3 Z_1^4 Y_1^2 Y_3^1 Y_2^1 Z_2^3 Z_1^4 Z_1^3 Z_2^2) = Y_2^1 Y_3^1 Y_1^2 Z_2^2 Z_1^3 Z_2^3 Z_1^4$. The algorithm does not depend on the prerequisite relation. An example of the operation of the algorithm for a specific example is found in Section 4.6.2.

Now, if we require the special initial state s_0^C, then we must make some adjustments to the transaction production set $BPT_{s_0^C}^C$. Consider a transaction production $p \in BPT_{s_0^C}^C$. We must delete this production if $rhs(p)$ is a string of output non-terminal symbols such that

$$\exists z \in \left(\bigcup_{\substack{k \in [1,2] \\ j \in [1,n^k]}} Z_{j,s_0^C}^C \right)^* \text{ s.t. } rhs(p) = z$$

and there exists another production $p' \in BPT_{s_0^C}^C$ such that

$$\exists y \in \left(\bigcup_{\substack{k \in [1,2] \\ i \in [1,m^k]}} Y_{i,s_0^C}^C \right)^+ \wedge \ rhs(p') = yz.$$

An example of the need for this procedure is found in Section 4.7.1. The extra state allows us to differentiate between the case where a feedback loop has never been active and where it is continuously active.

We define the set of input and output non-terminal symbols for the composite specification as follows:

$$\hat{V}_{\langle s_1,s_2 \rangle}^C = \bigcup_{\substack{k_1 \in [1,2] \\ i \in [1,m^{k_1}] \\ \neg\exists k_2 \in [1,2], j \in [1,n^{k_2}] \\ \langle \beta_j^{k_2}, \alpha_i^{k_1} \rangle \in connected}} \left\{ Y_{i,s_{k_1}}^{k_1} \right\} \cup \bigcup_{\substack{k_1 \in [1,2] \\ j \in [1,n^{k_1}] \\ \neg\exists k_2 \in [1,2], j \in [1,n^{k_2}] \\ \langle \beta_j^{k_1}, \alpha_i^{k_2} \rangle \in connected}} \left\{ Z_{j,s_{k_1}}^{k_1} \right\}.$$

The non-terminal symbols corresponding to the endpoints of a connection are omitted from $\hat{V}_{\langle s_1,s_2 \rangle}^C$. The set of common productions for the composite specification is defined as:

$$BPA^C_{\langle s_1,s_2\rangle} = \left\{ A_{\langle s_1,s_2\rangle} \to B^C_{1,\langle s_1,s_2\rangle} B^C_{2,\langle s_1,s_2\rangle} \ldots B^C_{b^C,\langle s_1,s_2\rangle} \right\}$$
$$\cup \bigcup_{k\in[1,2]} \left(\bigcup_{\substack{i\in[1,m^k] \\ \neg\exists e, \langle e,\alpha_i^k\rangle \in connected}} \left\{ Y^k_{i,s_k} \to \alpha_i^k \right\} \cup \bigcup_{\substack{j\in[1,n^k] \\ \neg\exists e, \langle \beta_j^k, e\rangle \in connected}} \left\{ Z^k_{j,s_k} \to \beta_j^k \right\} \right).$$

The next task is to define the sets BPB^C_s for arbitrary $s \in S^C$. This involves finding the "common ground" between each of the individual block production sets. For example, if one module specifies that each transaction may occur two, three, or four times and the other module specifies that each transaction may occur three or more times, then the composite will only allow three or four transactions, wherever a connection is made between the two blocks. This can be achieved with the following:

$$r^C_{\langle s_1,s_2\rangle} = r^1_{s_1} \cap r^2_{s_2}$$

If the result of this assignment is that $r^C_{\langle s_1,s_2\rangle} = \varnothing$ then the two modules being composed are inherently incompatible and cannot be connected. If the composition requires the extra state s^C_0, then we set $r_{s^C_0} = [1,1]$.

As before, we set $BP^C_s = BPA^C_s \cup BPB^C_s \cup BPT^C_s$ to obtain the compete set of basic productions for the composite system.

The set of delay productions is defined as:

$$DP^C_{\langle s_1,s_2\rangle} = \bigcup_{\substack{k_1,k_2\in[1,2] \\ i\in[1,m^{k_2}] \\ j\in[1,n^{k_1}] \\ \left(\neg\exists e_1\in {}^I E^C, \langle \beta_j^{k_1}, e_1\rangle \in connected\right) \wedge \left(\neg\exists e_2\in {}^O E^C, \langle e_2, \alpha_i^{k_2}\rangle \in connected\right)}} \left\{ Z^{k_1}_{j,s_{k_1}} Y^{k_2}_{i,s_{k_2}} \to Y^{k_2}_{i,s_{k_2}} Z^{k_1}_{j,s_{k_1}} \right\}$$

In other words, we allow any output of the composite system to the environment to be delayed.

The set of shuffle productions SP^C_S is constructed through a two-stage process, analogous to that described for single modules in Section 3.2.3.

Algorithm 4.2: Composite Shuffle Production Set Construction

$SP_s^C := \varnothing$

First add the productions within each block

$$\forall u \in BPT_s^C,\ v,x,z \in \left(\bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\} \right)^*,\ w,y \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\}$$

s.t. $rhs(u) = vwxyz,$

if $\left(w \notin \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\} \vee y \notin \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \right) \wedge \langle w,y \rangle \notin prereq_s^C$ **then**

$SP_S^C := SP_S^C \cup \{wy \to yw\}$

Now add the inter-block shuffle productions

$\forall k_1 \in \left[1, b^C - 1\right], k_2 \in [k_1 + 1, b^C],\ p_1, p_2 \in BPT_s^C,$

$$u,w,x,z \in \left(\bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\} \right)^*, v,y \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\},$$

s.t. $lhs(p_1) = T_{k_1,s}^C \wedge lhs(p_2) = T_{k_2,s}^C \wedge rhs(p_1) = uvw \wedge rhs(p_2) = xyz,$

if $\left(v \notin \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{Z_{j,s}^k\right\} \vee y \notin \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{Y_{i,s}^k\right\} \right)$ **then** $SP_s^C := SP_s^C \cup \{vy \to yv\}$

The particular ordering of inputs and outputs that is generated by the algorithm to create the transaction productions constrains the execution sequences so that events in a causally related group occur in order by module and line number. The shuffle productions generated above allow the possibility of other orderings, without violating causality requirements.

As in Section 3.2, we set $P_s^C = BP_s^C \cup DP_s^C \cup SP_s^C$.

Finally, the components of the overall module grammar $\boldsymbol{G}^C = \left\langle \boldsymbol{V}^C, E^C, \boldsymbol{P}^C, \boldsymbol{A}^C \right\rangle$ are:

- the non-terminal set $\boldsymbol{V}^C = \left\{\boldsymbol{A}^C\right\} \cup \bigcup_{s \in S} V_s^C$,
- the production set

$$\boldsymbol{P}^C = \left\{\boldsymbol{A}^C \to A^C_{\langle s_0^1, s_0^2 \rangle}\right\} \cup \bigcup_{\substack{\langle s_i, s_j \rangle \in N^C \\ s_i, s_j \notin badstate}} \left\{A^C_{s_i} \to A^C_{s_i} A^C_{s_j}\right\} \cup \bigcup_{\substack{\langle s_i, s_j \rangle \in N^C \\ \langle s_j, s_k \rangle \in N^C \\ s_j \in badstate}} \left\{A^C_{s_i} \to A^C_{s_i} A^C_{s_k}\right\} \cup \bigcup_{s \in S^C} P_s^C .$$

In the construction of the composite production set $\boldsymbol{P}^C$, the *badstate* set created by Algorithm 4.1 is used to determine which states to skip over.

The composition algorithm as defined above is clearly symmetric and associative, since the combination of the grammatical elements is based on the set union operator and the fact that $A \cup B = B \cup A$ and $A \cup (B \cup C) = (A \cup B) \cup C$. Thus, when composing more than two modules, the particular choice of composition ordering is irrelevant. We can compose the modules in a single step, or choose any two, compose them, then compose the result with a third, etc. The result, in either case, will be the same.

One of the most important aspects of the above specification is that an arbitrary composition always results in a specification that has all the properties (as defined by the requirements of Section 3.2) of a single "stand-alone" module specification.

Thus, after we have performed a composition[55] of two or more modules, the resulting structure is indistinguishable from a single atomic module, both in its operation and in its grammatical representation. When we consider properties in Chapter 5 this will allow us to treat "atomic" modules and composites in the same manner.

4.3 INTERNAL AND EXTERNAL EVENTS

In general, we are interested in modelling the manner in which the *system* under consideration interacts with its surrounding *environment*. For example, a computer workstation might be the system and its environment made up of the

---

[55]Including appropriate symbol renaming.

user typing at the keyboard, any networks to which it is connected, the external supply of electricity, the temperature of the room in which it is located, etc.

In the event-system approach of this work, a clear distinction must be made between internal system events and external environment events. This distinction ties in directly with the "black box" nature of the modular processing elements and the way in which these elements are composed hierarchically to form larger structures. Each of the individual basic module types reacts to events in the environment in a certain way. Any internal events and actions are hidden from the external environment. The grammatical structure makes this distinction by having all visible events as elements of the set of terminal symbols, E.

When we compose two or more modules, the environment that each module sees is changed by the interconnections made. In effect, we identify a particular output event of one module with an input event of another module. Each connection converts two previously visible environment events into a single hidden internal system event. The mechanism for achieving this conversion is the introduction of the "hybrid" transaction productions.

After modules have been composed, the composite structure is itself rendered a "black box". The particular internal operation becomes irrelevant, and the composite grammar represents the externally visible behaviour only. Likewise, the causality between internal events is removed from the causality relation, so that only externally visible events are considered.

4.4 DEADLOCK

Deadlock is an area that has been heavily researched by those in the operating system community. A set of modules will be in a deadlock state when every module in the set is waiting for an event that can only be caused by another module in the set[56].

---

[56]James L. Peterson and Abraham Silberschatz. *Operating System Concepts*. Second Edition. Addison-Wesley Publishing Company, 1985, pp. 271-306.

Both the transaction count parameter r_s and the *badstate* set are employed to identify and rule out situations where deadlock is possible as a result of interconnection incompatibilities. Therefore, we conjecture that the possibility of deadlock has been eliminated by our approach. It may also be the case that some deadlock-free systems are not allowed by the constraints that we place on composition. However, a comprehensive proof of the necessary and sufficient conditions for deadlock-freedom is beyond the scope of this work.

4.5 COMPLEXITY ISSUES RELATED TO COMPOSITION

The method used in our composition algorithm creates a new grammatical specification for each composite state. Consider the composition of p modules. The worst-case number of states occurs when we compose completely independent modules, with no connections between them. In such a case, every possible state of one module is compatible with every possible state of every other module. If each individual module specification has n states, then the composite specification will have a total of n^p states, and correspondingly, n^p state grammars. This type of exponential "state explosion" is a problem with all state-based representations.

There are a number of considerations that help to alleviate this problem. First, the worst case above describes components that are operating independently. In such cases, it may be more reasonable to map the problem into one in which the components can be modelled and studied separately. Second, whenever a connection is made between modules, the number of states in the resulting composite system is reduced. In most practical cases, there will be connections between modules.

Finally, as noted by Ramadge and Wonham,[57] there are a number of techniques available that may allow a mapping of a specification with many states into one with fewer states. If this is done prior to composition, a considerable reduction in the number of states will be possible for the composite system.

---

[57] Peter J.G. Ramadge and W. Murray Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

Another important complexity issue relates to the size (i.e., length) of the specification that results from a composition. A single module with m inputs, n outputs and state set S will have specification size $O(|S|mn)$, due to the size of the delay production sets DP_s.[58] If we compose two modules, the resulting specification size will be $O(|S^1||S^2|(m^1+m^2)(n^1+n^2))$. As with the growth in numbers of states, the worst case results when two modules are composed without connections between them.

4.6 SINGLE-STATE COMPOSITION EXAMPLES

This section will provide a number of examples where the composition method is applied to derive the composite grammar, given the grammars of two or more separate modules, and a particular interconnection. To begin, several single-state modules will be composed. This will be followed by some cases of composition of multi-state modules.

4.6.1 Simple Parallel Composition

In this section, two Basic Modules will be composed in a parallel fashion to create the composite structure shown Figure 4.1.

[58]This is most obvious if we consider the General Module present in Section 3.4.6.

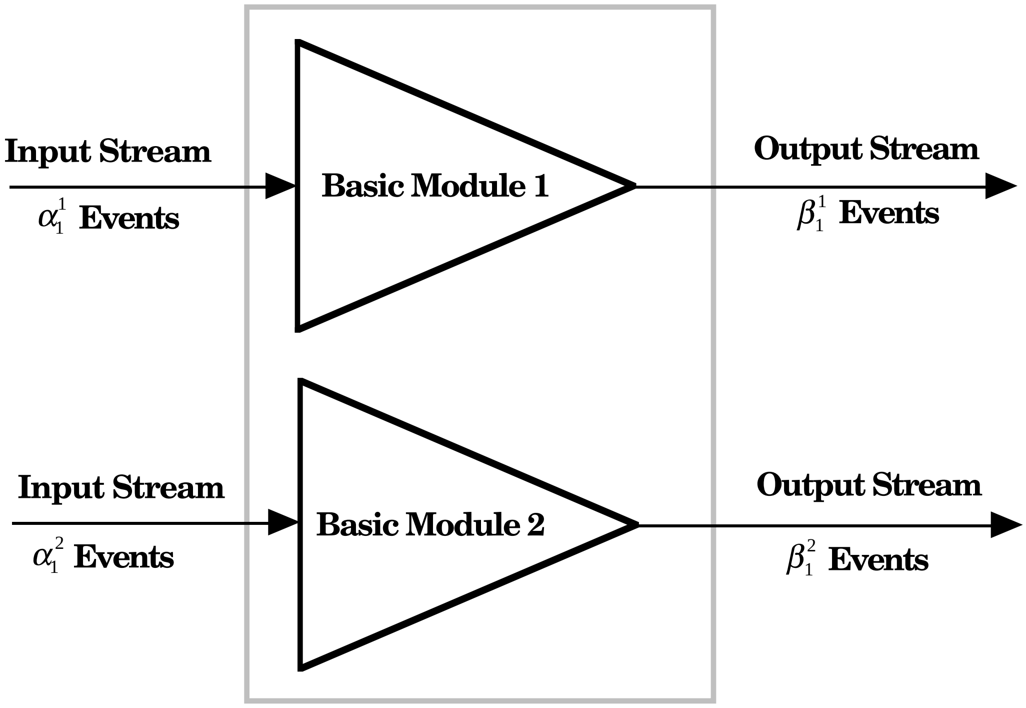

Figure 4.1: Parallel Composition of Basic Modules

In the figure for this and the other compositions, the grey box is used to separate the composite (i.e., the new internal) structure from the external environment.

The specifications of the two Basic Modules are the same as the specification from Section 3.4.1, with the addition of module number indices on all symbols:

$$\forall k \in [1,2],$$

$$m^k = 1 \;\; n^k = 1 \;\; b^k = 1 \;\; S^k = \left\{ s_0^k \right\} \;\; N^k = \varnothing$$

$$prereq_{s_0^k}^k = \left\{ \left\langle \alpha_1^k, \beta_1^k \right\rangle \right\} \;\; r_{s_0}^k = [0, \infty] \;\; BPT_{s_0^k}^k = \left\{ T_{1,s_0^k}^k \to Y_{1,s_0^k}^k Z_{1,s_0^k}^k \right\}$$

$$connected = \varnothing$$

The resulting composite system specification is

$$m^C = 2 \;\; n^C = 2 \;\; b^C = 2 \;\; S^C = \left\{ \left\langle s_0^1, s_0^2 \right\rangle \right\} \;\; N^C = \varnothing$$

$$E^C = \left\{ \alpha_1^1, \alpha_1^2, \beta_1^1, \beta_1^2 \right\}$$

$$prereq^C_{\langle s_0^1,s_0^2\rangle} = \left\{\left\langle \alpha_1^1,\beta_1^1\right\rangle,\left\langle \alpha_1^2,\beta_1^2\right\rangle\right\} \; r^C_{s_0} = [0,\infty] \; BPT^C_{\langle s_0^1,s_0^2\rangle} = \begin{Bmatrix} T^C_{1,\langle s_0^1,s_0^2\rangle} \to Y^1_{1,s_0^1} Z^1_{1,s_0^1} \\ T^C_{1,\langle s_0^1,s_0^2\rangle} \to Y^2_{1,s_0^2} Z^2_{1,s_0^2} \end{Bmatrix}$$

$$DP^C_{\langle s_0^1,s_0^2\rangle} = \begin{Bmatrix} Z^1_{1,s_0^1} Y^1_{1,s_0^1} \to Y^1_{1,s_0^1} Z^1_{1,s_0^1} \\ Z^1_{1,s_0^1} Y^2_{1,s_0^2} \to Y^2_{1,s_0^2} Z^1_{1,s_0^1} \\ Z^2_{1,s_0^2} Y^1_{1,s_0^1} \to Y^1_{1,s_0^1} Z^2_{1,s_0^2} \\ Z^2_{1,s_0^2} Y^2_{1,s_0^2} \to Y^2_{1,s_0^2} Z^2_{1,s_0^2} \end{Bmatrix} \; SP^C_{\langle s_0^1,s_0^2\rangle} = \begin{Bmatrix} Y^1_{1,s_0^1} Y^2_{1,s_0^2} \to Y^2_{1,s_0^2} Y^1_{1,s_0^1} \\ Y^1_{1,s_0^1} Z^2_{1,s_0^2} \to Z^2_{1,s_0^2} Y^1_{1,s_0^1} \\ Z^1_{1,s_0^1} Z^2_{1,s_0^2} \to Z^2_{1,s_0^2} Z^1_{1,s_0^1} \end{Bmatrix}$$

Since the two modules are effectively independent in this configuration, the parallel composition requires the generation of a language whose sentences allow the event sequences of the two modules to be freely mixed. The basic production set for the composite system generates sequences of complete transactions for either of the two modules. All of the transactions from the first module appear before those of the second module in the basic grammar, due to the definition of the common production set:

$$BPA^C_{\langle s_1,s_2\rangle} = \left\{A_s \to B^C_{1,s} B^C_{2,s}\right\} \cup \bigcup_{k\in[1,2]} \begin{Bmatrix} Y^k_{1,s_k} \to \alpha_i^k \\ Z^k_{1,s_k} \to \beta_1^k \end{Bmatrix}.$$

The shuffle productions then allow the non-terminals from transactions of the first module to "percolate" past those of the second module to their desired final location. The composite delay production set allows either of the outputs to be delayed past any new inputs.

The composite structure that results is pictured in Figure 4.2 (with internal structures in grey):

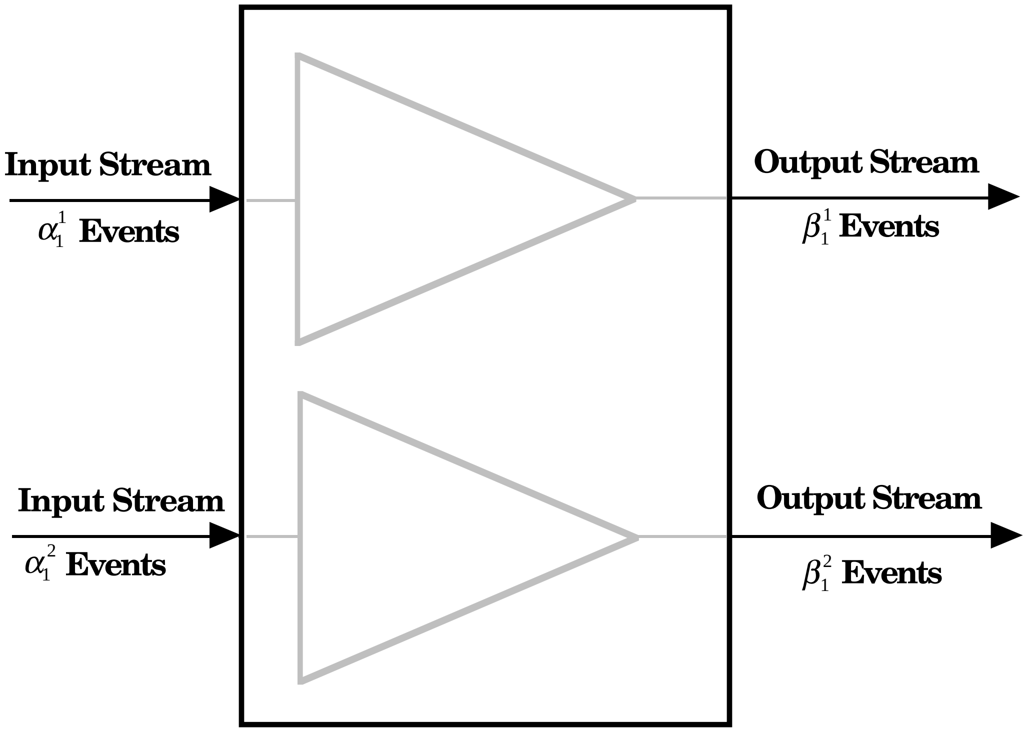

Figure 4.2: Parallel Composition with Internal Structures Removed

As far as the external environment is concerned, the elements in grey above are not relevant. Only external behaviour visible at the inputs and outputs is observable. The external behaviour is identical to that of the Parallel Module presented in Section 3.4.9.

4.6.2 Simple Serial Composition

In this section, two Basic Modules will be connected in a serial fashion to create a composite structure that is shown in Figure 4.3.

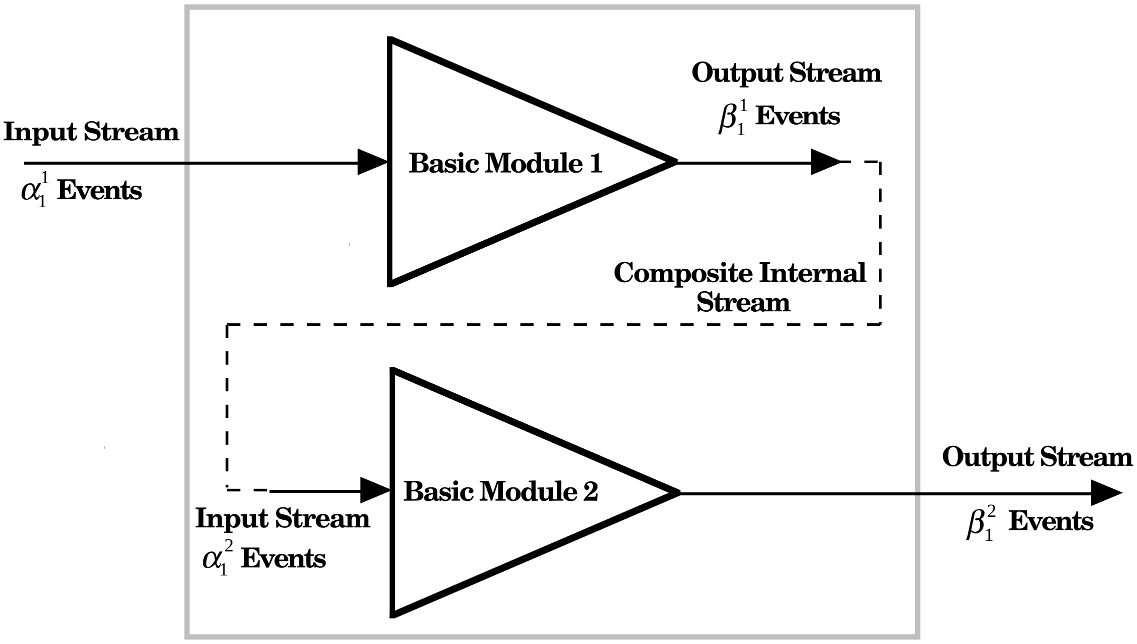

Figure 4.3: Serial Composition of Basic Modules

The specifications of the individual modules are the same as in the case described in Section 4.6.1:

$$\forall k \in [1,2],$$

$$m^k = 1 \;\; n^k = 1 \;\; b^k = 1 \;\; S^k = \left\{ s_0^k \right\} \;\; N^k = \varnothing$$

$$prereq_{s_0^k}^k = \left\{ \left\langle \alpha_1^k, \beta_1^k \right\rangle \right\} \;\; r_{s_0}^k = [0, \infty] \;\; BPT_{s_0^k}^k = \left\{ T_{1,s_0^k}^k \rightarrow Y_{1,s_0^k}^k Z_{1,s_0^k}^k \right\}$$

However, we now have the interconnection:

$$connected = \left\{ \left\langle \beta_1^1, \alpha_1^2 \right\rangle \right\}$$

The resulting composite system specification is quite different from the previous case. In this case, we will describe how Algorithm 4.1 creates the composite transaction production set.

Consider the two transaction production sets $BPT_{s_0}^1 = \left\{ T_{1,s_0^1}^1 \rightarrow Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 \right\}$ and $BPT_{s_0}^1 = \left\{ T_{1,s_0^1}^1 \rightarrow Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 \right\}$ and the desired connection $connected = \left\{ \left\langle \beta_1^1, \alpha_1^2 \right\rangle \right\}$. This would result in the procedure call $Unify\left(Z^1_{1,s^1_0}, Y^2_{1,s^2_0}\right)$. In the procedure, we would first set $outprods = \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^1_{1,s^1_0} Z^1_{1,s^1_0}\right\}$, $inprods = \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^2_{1,s^2_0} Z^2_{1,s^2_0}\right\}$ and $bothprods = \varnothing$.

Next, we perform the concatenation:

$$\begin{aligned}
BPT^C_{\langle s^1_0,s^2_0\rangle} &= \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to remdups\left(\begin{array}{l} delstring\left(Z^1_{1,s^1_0}, rhs\left(T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^1_{1,s^1_0} Z^1_{1,s^1_0}\right)\right) \\ +delstring\left(Y^2_{1,s^2_0}, rhs\left(T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^2_{1,s^2_0} Z^2_{1,s^2_0}\right)\right)\end{array}\right)\right\} \\
&= \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to remdups\left(delstring\left(Z^1_{1,s^1_0}, Y^1_{1,s^1_0} Z^1_{1,s^1_0}\right) + delstring\left(Y^2_{1,s^2_0}, Y^2_{1,s^2_0} Z^2_{1,s^2_0}\right)\right)\right\} \\
&= \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to remdups\left(Y^1_{1,s^1_0} + Z^2_{1,s^2_0}\right)\right\} \\
&= \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^1_{1,s^1_0} Z^2_{1,s^2_0}\right\}
\end{aligned}$$

The composite specification that results is:

$$m^C = 1 \;\; n^C = 1 \;\; b^C = 1 \;\; S^C = \left\{\left\langle s^1_0, s^2_0\right\rangle\right\} \;\; N^C = \varnothing$$

$$E^C = \left\{\alpha^1_1, \beta^2_1\right\}$$

$$prereq^C_{\langle s^1_0,s^2_0\rangle} = \left\{\left\langle \alpha^1_1, \beta^2_1\right\rangle\right\} \;\; r^C_{s_0} = [0,\infty]$$

$$BPT^C_{\langle s^1_0,s^2_0\rangle} = \left\{T^C_{1,\langle s^1_0,s^2_0\rangle} \to Y^1_{1,s^1_0} Z^2_{1,s^2_0}\right\}$$

$$DP^C_{\langle s^1_0,s^2_0\rangle} = \left\{Z^2_{1,s^2_0} Y^1_{1,s^1_0} \to Y^1_{1,s^1_0} Z^2_{1,s^2_0}\right\} \;\; SP^C_{\langle s^1_0,s^2_0\rangle} = \varnothing$$

The composite grammar has removed all reference to the terminal and non-terminal symbols corresponding to the connection endpoint events β^1_1 and α^2_1. That connection requires a close relationship between the two modules. The number of output events β^1_1 from the first module must be equal to the number of α^2_1 input events to the second module. In addition, the input to module 1 must always occur before the corresponding output of module 2. These requirements are satisfied by the composite transaction $T^C_{1,\langle s^1_0,s^2_0\rangle} \rightarrow Y^1_{1,s^1_0} Z^2_{1,s^2_0}$ and the composite prerequisite relation $prereq^C_{\langle s^1_0,s^2_0\rangle} = \left\{\left\langle \alpha^1_1, \beta^2_1 \right\rangle\right\}$.

Compare the situation here with that of the previous section. When we composed the two modules in parallel, there was no connection between them. Hence, there was no causal relationship between events occurring in either of the modules. The composite grammar reflected this by having separate composite transactions corresponding to each module. By connecting the two modules in serial fashion as we have done in this section, we have introduced a causal relationship between them. Thus, we merge the two modules' transactions into a single composite transaction.

The resulting composite system is pictured in Figure 4.4.

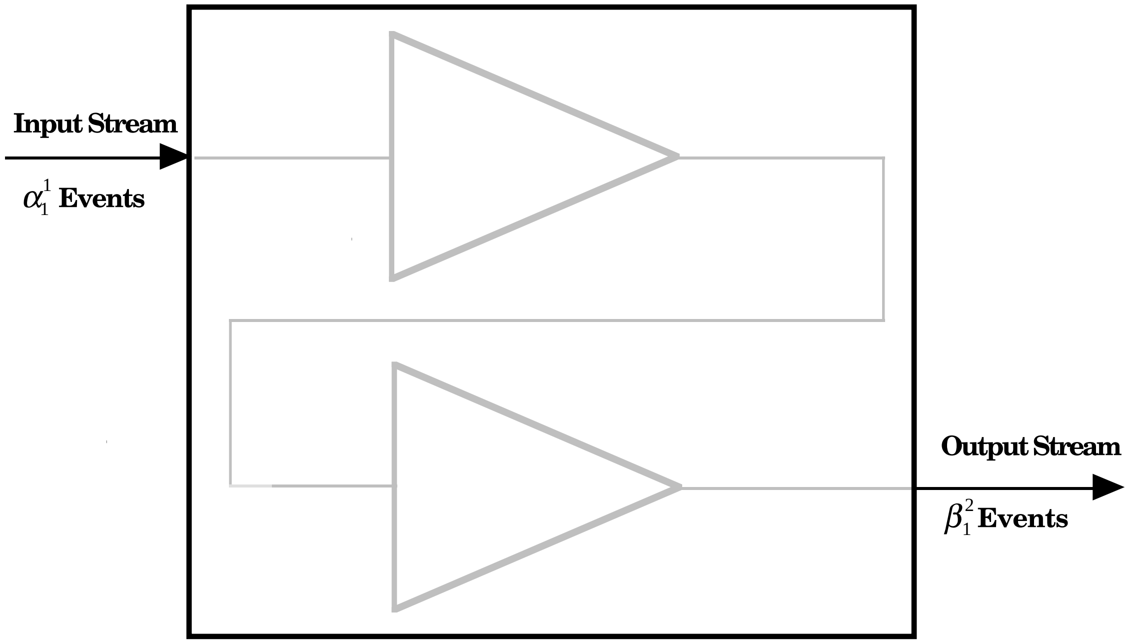

Figure 4.4: Serial Composition of Basic Modules with Internals Hidden

As far as the external environment is concerned, the composite structure behaves exactly as if it were a single Basic Module, even though it is really composed of two Basic Modules in series. This is also reflected in the composite specification: with symbol renaming it is equivalent to the Basic Module grammar of Section 3.4.1.

4.6.3 Three Module Composition

This section will consider a more complicated example involving the composition of three separate modules that appears in Figure 4.5. This example serves to demonstrate the importance of the prerequisite relation in controlling the shuffling of independent and dependent input and output streams.

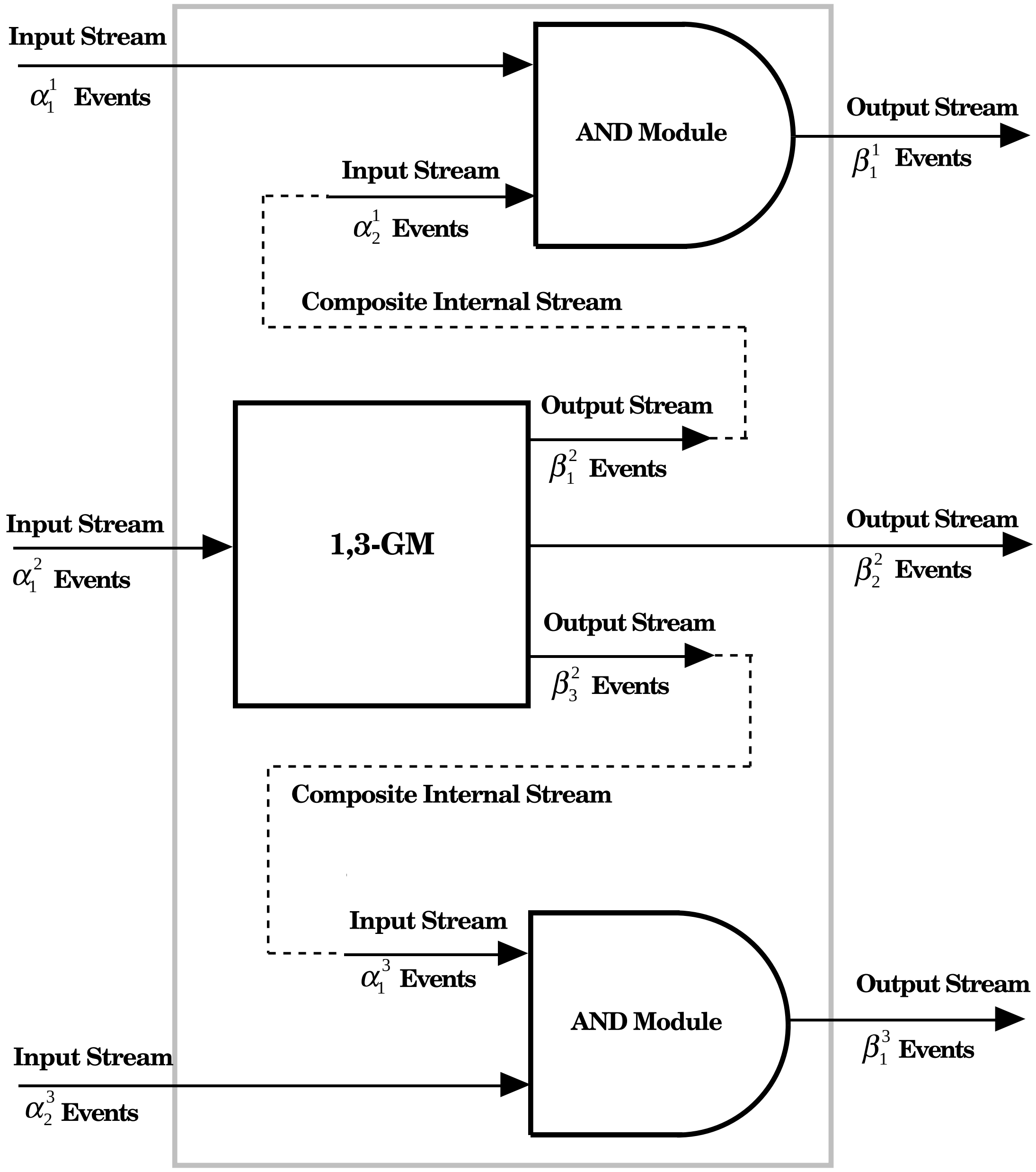

Figure 4.5: Composition of 1,3-GM with Two AND Modules

The two AND Modules are specified by

$$\forall k \in \{1,3\},$$

$$m^k = 2 \;\; n^k = 1 \;\; b^k = 1 \;\; S^k = \left\{s_0^k\right\} \;\; N^k = \varnothing$$

$$prereq_{s_0^k}^k = \left\{\langle \alpha_1^k, \beta_1^k \rangle, \langle \alpha_2^k, \beta_1^k \rangle\right\} \;\; r_{s_0^k}^k = [0,\infty] \;\; BPT_{s_0^k}^k = \left\{T_{1,s_0^k}^k \to Y_{1,s_0^k}^k Y_{2,s_0^k}^k Z_{1,s_0^k}^k\right\}$$

the 1,3-GM by

$$m^2 = 1 \;\; n^2 = 3 \;\; b^2 = 1 \;\; S^2 = \left\{s_0^2\right\} \;\; N^2 = \varnothing$$

$$prereq_{s_0^2}^2 = \bigcup_{j \in [1,3]} \left\{\langle \alpha_1^2, \beta_j^2 \rangle\right\} \;\; r_{s_0^2}^2 = [0,\infty] \;\; BPT_{s_0^2}^2 = \left\{T_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{1,s_0^2}^2 Z_{2,s_0^2}^2 Z_{3,s_0^2}^2\right\}$$

and the interconnections by

$$connected = \left\{\langle \beta_1^2, \alpha_2^1 \rangle, \langle \beta_3^2, \alpha_1^3 \rangle\right\}.$$

The resulting composite system is specified by

$$m^C = 3 \;\; n^C = 3 \;\; b^C = 1 \;\; S^C = \left\{\langle s_0^1, s_0^2, s_0^3 \rangle\right\} \;\; N^C = \varnothing$$

$$E^C = \left\{\alpha_1^1, \alpha_1^2, \alpha_2^3, \beta_1^1, \beta_2^2, \beta_1^3\right\}$$

$$prereq_{\langle s_0^1, s_0^2, s_0^3 \rangle}^C = \left\{\langle \alpha_1^1, \beta_1^1 \rangle, \langle \alpha_1^2, \beta_1^1 \rangle, \langle \alpha_1^2, \beta_2^2 \rangle, \langle \alpha_1^2, \beta_1^3 \rangle, \langle \alpha_2^3, \beta_1^3 \rangle\right\}$$

$$r_{\langle s_0^1, s_0^2, s_0^3 \rangle}^C = [0,\infty]$$

$$BPT_{\langle s_0^1, s_0^2, s_0^3 \rangle}^C = \left\{T_{1,\langle s_0^1, s_0^2, s_0^3 \rangle}^C \to Y_{1,s_0^1}^1 Y_{1,s_0^2}^2 Y_{2,s_0^3}^3 Z_{1,s_0^1}^1 Z_{2,s_0^2}^2 Z_{1,s_0^3}^3\right\}$$

$$DP_{\langle s_0^1, s_0^2, s_0^3 \rangle}^C = \left\{\begin{array}{lll} Z_{1,s_0^1}^1 Y_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 & Z_{2,s_0^2}^2 Y_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{2,s_0^2}^2 & Z_{1,s_0^3}^3 Y_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{1,s_0^3}^3 \\ Z_{1,s_0^1}^1 Y_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{1,s_0^1}^1 & Z_{2,s_0^2}^2 Y_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{2,s_0^2}^2 & Z_{1,s_0^3}^3 Y_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{1,s_0^3}^3 \\ Z_{1,s_0^1}^1 Y_{2,s_0^3}^3 \to Y_{2,s_0^3}^3 Z_{1,s_0^1}^1 & Z_{2,s_0^2}^2 Y_{2,s_0^3}^3 \to Y_{2,s_0^3}^3 Z_{2,s_0^2}^2 & Z_{1,s_0^3}^3 Y_{2,s_0^3}^3 \to Y_{2,s_0^3}^3 Z_{1,s_0^3}^3 \end{array}\right\}$$

$$SP^C_{\langle s_0^1, s_0^2, s_0^3 \rangle} = \left\{ \begin{array}{ll} Y^1_{1,s_0^1} Y^2_{1,s_0^2} \rightarrow Y^2_{1,s_0^2} Y^1_{1,s_0^1} & Y^2_{1,s_0^2} Y^3_{2,s_0^3} \rightarrow Y^3_{2,s_0^3} Y^2_{1,s_0^2} \\ Y^1_{1,s_0^1} Y^3_{2,s_0^3} \rightarrow Y^3_{2,s_0^3} Y^1_{1,s_0^1} & Y^3_{2,s_0^3} Z^1_{1,s_0^1} \rightarrow Z^1_{1,s_0^1} Y^3_{2,s_0^3} \\ Y^1_{1,s_0^1} Z^2_{2,s_0^2} \rightarrow Z^2_{2,s_0^2} Y^1_{1,s_0^1} & Y^3_{2,s_0^3} Z^2_{2,s_0^2} \rightarrow Z^2_{2,s_0^2} Y^3_{2,s_0^3} \\ Y^1_{1,s_0^1} Z^3_{1,s_0^3} \rightarrow Z^3_{1,s_0^3} Y^1_{1,s_0^1} & \end{array} \right\}$$

In this example, there is a much more complex temporal relationship between the events in the three modules. The set of shuffle productions, derived from the prerequisite relation, reflects this.

The increased notational complexity of this example clearly indicates the need for an automated tool in dealing with large compositions.

4.6.4 Nondeterministic Example

This section will consider composition involving a Nondeterministic Fork Module and a Sink Module. The particular structure that results provides an interesting example of how the nondeterministic modules available can be combined to create interesting composite behaviours. The interconnection is shown in Figure 4.6.

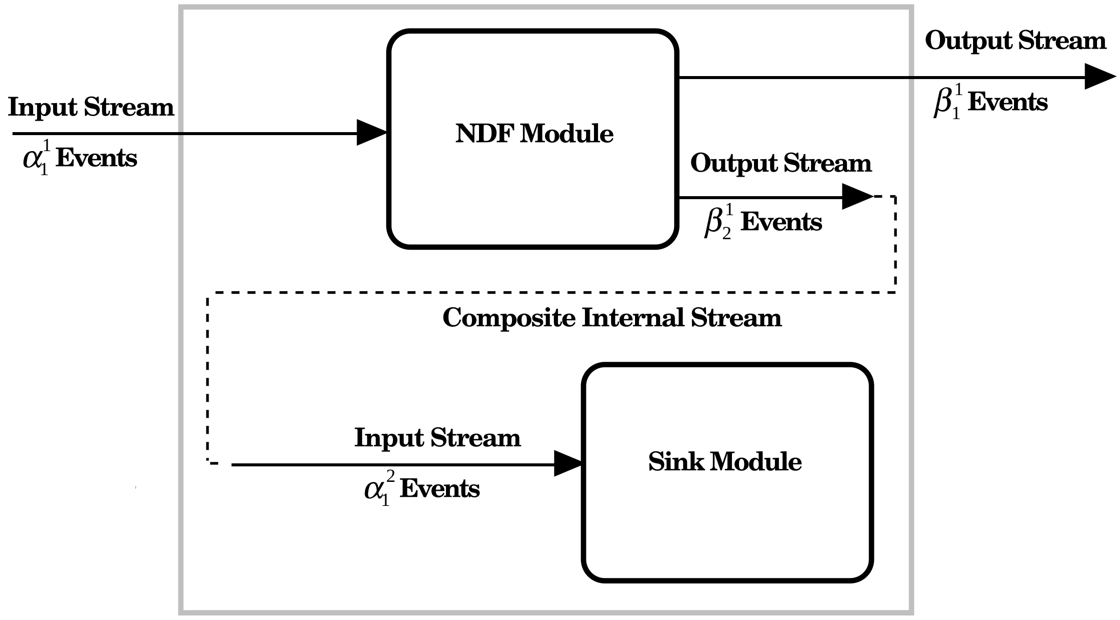

Figure 4.6: NDF Module Composed with Sink Module

The specification of the components and interconnection is:

$$m^1 = 1 \;\; n^1 = 2 \;\; b^1 = 1 \;\; S^1 = \left\{ s_0^1 \right\} \;\; N^1 = \varnothing$$

$$prereq_{s_0^1}^1 = \left\{ \left\langle \alpha_1^1, \beta_1^1 \right\rangle, \left\langle \alpha_1^1, \beta_2^1 \right\rangle \right\} \;\; r_{s_0}^k = [0, \infty] \;\; BPT_{s_0^1}^1 = \begin{Bmatrix} T_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 \\ T_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{2,s_0^1}^1 \end{Bmatrix}$$

$$m^2 = 1 \;\; n^2 = 0 \;\; b^2 = 1 \;\; S^2 = \left\{ s_0^2 \right\} \;\; I^2 = s_0^2 \;\; N^2 = \varnothing$$

$$prereq_{s_0^2}^2 = \varnothing \;\; BPB_{s_0^2}^2 = \begin{Bmatrix} B_{1,s_0^2}^2 \to T_{1,s_0^2}^2 B_{1,s_0^2}^2 \\ B_{1,s_0^2}^2 \to \varepsilon \end{Bmatrix} \;\; BPT_{s_0^2}^2 = \left\{ T_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 \right\}$$

$$connected = \left\{ \left\langle \beta_2^1, \alpha_1^2 \right\rangle \right\}$$

and the resulting composite specification is

$$m^C = 1 \;\; n^C = 1 \;\; b^C = 1 \;\; S^C = \left\{ \left\langle s_0^1, s_0^2 \right\rangle \right\} \;\; N^C = \varnothing$$

$$prereq_{\left\langle s_0^1, s_0^2 \right\rangle}^C = \left\{ \left\langle \alpha_1^1, \beta_1^1 \right\rangle \right\} \;\; r_{s_0}^C = [0, \infty] \;\; BPT_{\left\langle s_0^1, s_0^2 \right\rangle}^C = \begin{Bmatrix} T_{1,\left\langle s_0^1, s_0^2 \right\rangle}^C \to Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 \\ T_{1,\left\langle s_0^1, s_0^2 \right\rangle}^C \to Y_{1,s_0^1}^1 \end{Bmatrix}$$

The operation of the composite system creates a mechanism in which, given an input event, an output event may or may not be generated. The prerequisite relation $\left\langle \alpha_1^1, \beta_1^1 \right\rangle \in prereq_{\left\langle s_0^1, s_0^2 \right\rangle}^C$ for the composite system guarantees that an α_1^1 event always occurs before its corresponding β_1^1 event, if one is generated. If no β_1^1 event is generated, then the prerequisite relation has no impact.

4.6.5 Another Nondeterministic Example

This section will consider a more complicated system involving the serial composition of an OR Module and a NDF Module. This will demonstrate the complexity involved in developing the composite production set from those of the individual modules.

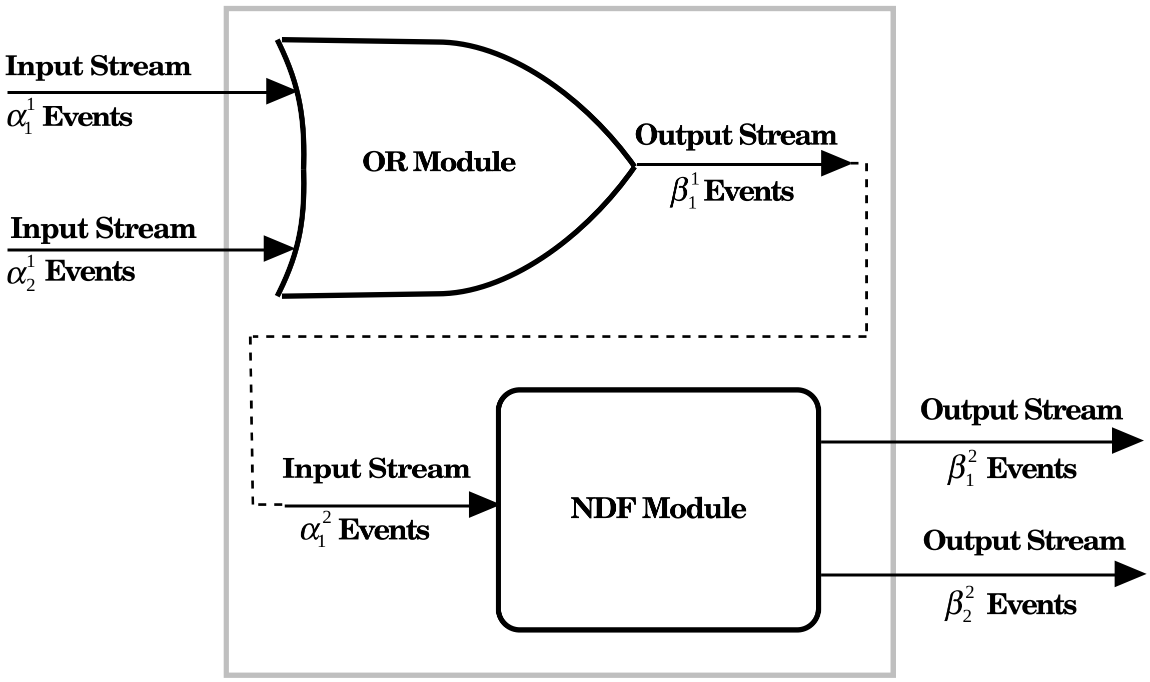

Figure 4.7: OR Module Composed with NDF Module

The specification of the components and interconnection is:

$$m^1 = 2 \;\; n^1 = 1 \;\; b^1 = 1 \;\; S^1 = \left\{ s_0^1 \right\} \;\; N^1 = \varnothing$$

$$prereq_{s_0^1}^1 = \left\{ \left\langle \alpha_1^1, \beta_1^1 \right\rangle, \left\langle \alpha_2^1, \beta_1^1 \right\rangle \right\} \;\; r_{s_0}^1 = [0, \infty] \;\; BPT_{s_0^1}^1 = \begin{Bmatrix} T_{1,s_0^1}^1 \to Y_{1,s_0^1}^1 Z_{1,s_0^1}^1 \\ T_{1,s_0^1}^1 \to Y_{2,s_0^1}^1 Z_{1,s_0^1}^1 \end{Bmatrix}$$

$$m^2 = 1 \;\; n^2 = 2 \;\; b^2 = 1 \;\; S^2 = \left\{ s_0^2 \right\} \;\; N^2 = \varnothing$$

$$prereq_{s_0^2}^2 = \left\{ \left\langle \alpha_1^2, \beta_1^2 \right\rangle, \left\langle \alpha_1^2, \beta_2^2 \right\rangle \right\} \;\; r_{s_0}^2 = [0, \infty] \;\; BPT_{s_0^2}^2 = \begin{Bmatrix} T_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{1,s_0^2}^2 \\ T_{1,s_0^2}^2 \to Y_{1,s_0^2}^2 Z_{2,s_0^2}^2 \end{Bmatrix}$$

$$connected = \left\{ \left\langle \beta_1^1, \alpha_1^2 \right\rangle \right\}$$

and the resulting composite specification is

$$m^C = 2 \;\; n^C = 2 \;\; b^C = 1 \;\; S^C = \left\{ \left\langle s_0^1, s_0^2 \right\rangle \right\} \;\; N^C = \varnothing$$

$$prereq^C_{\langle s_0^1, s_0^2 \rangle} = \begin{Bmatrix} \langle \alpha_1^1, \beta_1^2 \rangle, \langle \alpha_2^1, \beta_1^2 \rangle \\ \langle \alpha_1^1, \beta_2^2 \rangle, \langle \alpha_2^1, \beta_2^2 \rangle \end{Bmatrix} \; r^C_{s_0} = [0, \infty] \;\; BPT^C_{\langle s_0^1, s_0^2 \rangle} = \begin{Bmatrix} T^C_{1,\langle s_0^1, s_0^2 \rangle} \rightarrow Y^1_{1,s_0^1} Z^2_{1,s_0^1} \\ T^C_{1,\langle s_0^1, s_0^2 \rangle} \rightarrow Y^1_{1,s_0^1} Z^2_{2,s_0^1} \\ T^C_{1,\langle s_0^1, s_0^2 \rangle} \rightarrow Y^1_{2,s_0^1} Z^2_{1,s_0^1} \\ T^C_{1,\langle s_0^1, s_0^2 \rangle} \rightarrow Y^1_{2,s_0^1} Z^2_{2,s_0^1} \end{Bmatrix}$$

Each of the individual modules in the composition, when considered separately, has two possible transactions. When these are composed in a serial fashion, four "hybrid" transactions are created to account for the four possible overall behaviours of the system.

No shuffle productions are possible due to the restrictions imposed by the prerequisite relation.

4.7 MULTIPLE-STATE COMPOSITION EXAMPLES

4.7.1 Feedback Example

This example will create a triggered oscillating system that is shown in Figure 4.8 below.

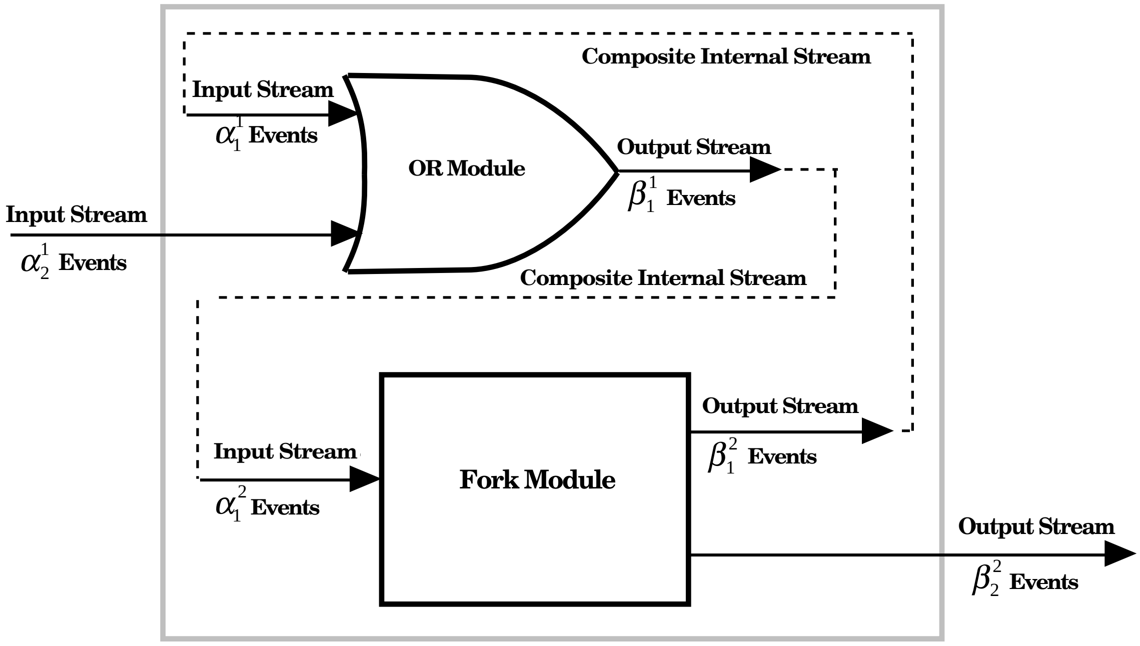

Figure 4.8: OR Module with Fork Module Feedback

As a result of the feedback loop, this is one of the situations mentioned in Section 3.2 requiring that we explicitly distinguish between the initial state of the two modules (i.e., before any inputs have occurred and the feedback loop is empty) and the "operating" state after an input has been received and a continuous stream of events travel around the feedback loop.

The component and connection specification of the above system is as follows:

$$m^1 = 2 \;\; n^1 = 1 \;\; b^1 = 1 \;\; S^1 = \left\{s_0^1\right\} \;\; N^1 = \varnothing$$

$$prereq^1_{s_0^1} = \left\{\left\langle \alpha_1^1, \beta_1^1 \right\rangle, \left\langle \alpha_2^1, \beta_1^1 \right\rangle\right\} \;\; r^1_{s_0} = [0, \infty] \;\; BPT^1_{s_0^1} = \begin{Bmatrix} T^1_{1,s_0^1} \rightarrow Y^1_{1,s_0^1} Z^1_{1,s_0^1} \\ T^1_{1,s_0^1} \rightarrow Y^1_{2,s_0^1} Z^1_{1,s_0^1} \end{Bmatrix}$$

$$m^2 = 1 \;\; n^2 = 2 \;\; b^2 = 1 \;\; S^2 = \left\{s_0^2\right\} \;\; N^2 = \varnothing$$

$$prereq^2_{s_0^2} = \left\{\left\langle \alpha_1^2, \beta_1^2 \right\rangle, \left\langle \alpha_1^2, \beta_2^2 \right\rangle\right\} \;\; r^2_{s_0} = [0, \infty] \;\; BPT^2_{s_0^2} = \left\{T^2_{1,s_0^2} \rightarrow Y^2_{1,s_0^2} Z^2_{1,s_0^2} Z^2_{2,s_0^2}\right\}$$

$$connected = \left\{\left\langle \beta_1^1, \alpha_1^2 \right\rangle, \left\langle \beta_1^2, \alpha_1^1 \right\rangle\right\}$$

and the composite specification is

$$m^C = 1 \;\; n^C = 1 \;\; b^C = 1 \;\; S^C = \left\{s_0^C, \left\langle s_0^1, s_0^2 \right\rangle\right\} \;\; N^C = \left\{\left\langle s_0^C, \left\langle s_0^1, s_0^2 \right\rangle\right\rangle\right\}$$

$$E^C = \left\{\alpha_1^1, \beta_2^2\right\}$$

$$prereq^C_{s_0^C} = \left\{\left\langle \alpha_1^1, \beta_2^2 \right\rangle\right\} \;\; r^C_{s_0^C} = [1,1]$$

$$BPT^C_{s_0^C} = \left\{T^C_{1,s_0^C} \rightarrow Y^1_{1,s_0^1} Z^2_{2,s_0^2}\right\}$$

$$DP^C_{s_0^C} = \left\{Z^2_{2,s_0^2} Y^1_{1,s_0^1} \rightarrow Y^1_{1,s_0^1} Z^2_{2,s_0^2}\right\} \;\; SP^C_{s_0^C} = \varnothing$$

$$prereq^C_{\left\langle s_0^1, s_0^2 \right\rangle} = \left\{\left\langle \alpha_1^1, \beta_2^2 \right\rangle\right\} \;\; r^C_{\left\langle s_0^1, s_0^2 \right\rangle} = [0, \infty]$$

$$BPT^C_{\left\langle s_0^1, s_0^2 \right\rangle} = \begin{Bmatrix} T^C_{1,\left\langle s_0^1, s_0^2 \right\rangle} \rightarrow Y^1_{1,s_0^1} Z^2_{2,s_0^2} \\ T^C_{1,\left\langle s_0^1, s_0^2 \right\rangle} \rightarrow Z^2_{2,s_0^2} \end{Bmatrix}$$

$$DP^{C}_{\langle s_0^1, s_0^2 \rangle} = \left\{ Z^2_{2,s_0^2} Y^1_{1,s_0^1} \rightarrow Y^1_{1,s_0^1} Z^2_{2,s_0^2} \right\} \; SP^{C}_{\langle s_0^1, s_0^2 \rangle} = \varnothing$$

This composition is different from any of those presented so far. The difference lies in the existence of the feedback loop. Before any input events have occurred, the system is idle and there are no events on the loop. After one input, events travel around the loop and the system then continuously generates output events. Thus, the system requires the addition of the extra state s_0^C to describe the initial situation before the input has arrived. Once a single transaction has occurred, the system moves into the feedback state.

The construction of the transaction production set $BPT^{C}_{s_0^C}$ follows the guidelines described in Section 4.2.2 and as a result $BPT^{C}_{s_0^C} = BPT^{C}_{\langle s_0^1, s_0^2 \rangle} \setminus \left\{ T^{C}_{1,\langle s_0^1, s_0^2 \rangle} \rightarrow Z^2_{2,s_0^2} \right\}$.

4.8 SUMMARY

This chapter began by introducing a formalized mechanism for composing two or more systems specified using the technique from Section 3.2. The specification of the composition was of the same form as a single-module specification. Next, internal and external events and deadlock were discussed. Finally, a number of composition examples were presented.

5 PROPERTIES

5.1 INTRODUCTION

Chapter 3 presented a method by which the behaviour of abstract processing modules could be easily described. A method for interconnecting simple structures to form more complex ones was then introduced and formalized in Chapter 4.

This chapter represents the next major section of the work. We will begin by developing formal notions of properties within the language-based framework. A generalized notion of properties will be presented and its composability investigated.

5.2 EVENT LABELS

In all of the previous sections, it has been assumed that all of the events associated with a particular module input or output were homogeneous. This section will describe an extension that will allow events of a certain class to occur on a particular input or output. Each event will have the same base symbol, with a variable attached label. This will greatly increase the flexibility of the approach by allowing different types of events to appear at a given input or be produced at an output.

Consider a set of event labels L. This section will extend the specification structure presented in Section 3.2 to include the labelling information. To avoid repetition, only the aspects of the modular representation that require changes will be presented. In general, the extension will involve adding label information to both terminal and non-terminal event input and output symbols.

We use the event terminal symbol ${}^{l}\alpha_i$ to refer to the (input) event with label l on input stream i and the event terminal symbol ${}^{l}\beta_j$ to refer to the (output) event with label l on output stream j of a particular module. We will define ${}^{I}E = \bigcup_{\substack{l\in L\\ i\in[1,m]}}\left\{{}^{l}\alpha_i\right\}$ and ${}^{O}E = \bigcup_{\substack{l\in L\\ j\in[1,n]}}\left\{{}^{l}\beta_j\right\}$ to be the set of module input events and output events, respectively. The set $E = \bigcup_{l\in L}\left(\bigcup_{i\in[1,m]}\left\{{}^{l}\alpha_i\right\} \cup \bigcup_{j\in[1,n]}\left\{{}^{l}\beta_j\right\}\right)$ is the set of events that are relevant to the abstraction being built. Each input and output is assigned a group of $|L|$ events so that $|E| = |L|(m+n)$.

Define $\hat{V}_s = \bigcup_{l\in L}\left(\bigcup_{i\in[1,m]}\left\{{}^{l}Y_{i,s}\right\} \cup \bigcup_{j\in[1,n]}\left\{{}^{l}Z_{j,s}\right\}\right)$ to be the set of input and output non-terminal events for state s.

There is some variability between modules in the basic productions set. However, all have the following core sets of productions:

$$BPA_s = \left\{A_s \rightarrow B_{1,s}B_{2,s}\ldots B_{b,s}\right\} \cup \bigcup_{l\in L}\left(\bigcup_{i\in[1,m]}\left\{{}^{l}Y_{i,s} \rightarrow {}^{l}\alpha_i\right\} \cup \bigcup_{j\in[1,n]}\left\{{}^{l}Z_{j,s} \rightarrow {}^{l}\beta_j\right\}\right)$$

The of transaction productions BPT_s derives the non-terminals $T_{k,s}$ and defines the possible complete transactions for block k of the module when in state s.

The requirements placed on the set BPT_s are an extension of those presented in Section 3.2:

Requirement 5.1:

$$\forall s \in S, p \in BPT_s, \quad \exists y \in \left(\bigcup_{\substack{l\in L\\ i\in[1,m]}}\left\{{}^{l}Y_{i,s}\right\}\right)^{*}, z \in \left(\bigcup_{\substack{l\in L\\ j\in[1,n]}}\left\{{}^{l}Z_{j,s}\right\}\right)^{*} \quad \text{s.t.} \quad rhs(p) = yz$$

The symbols forming the right-hand side of any transaction production must be ordered so that all input non-terminals appear before all output non-terminals. With the addition of the labels, this is the same as Requirement 3.1.

Requirement 5.2:

$$\forall s \in S,\ p \in BPT_s,\ v,x,z \in \hat{V}_s^*, w,y \in \hat{V}_s,$$
$$\left(rhs(p) = vwxyz\right) \Rightarrow w \neq y$$

Each of the input and output non-terminal symbols from the set $\hat{V}_s = \bigcup_{l \in L}\left(\bigcup_{i \in [1,m]}\left\{ {}^{l}Y_{i,s} \right\} \cup \bigcup_{j \in [1,n]}\left\{ {}^{l}Z_{j,s} \right\} \right)$ may appear at most once in any single production.

Requirement 5.3:

$$\forall s \in S,\ y \in \hat{V},\ k_1,k_2 \in [1,b],\ p_1,p_2 \in BPT_s,\ v,w,x,z \in \hat{V}_s^*,$$
$$\left(lhs(p_1) = T_{k_1,s} \wedge lhs(p_2) = T_{k_2,s} \wedge rhs(p_1) = vyw \wedge rhs(p_2) = xyz\right) \Rightarrow k_1 = k_2$$

If two transactions share an input or an output non-terminal symbol, then they must be part of the same block (i.e., have the same assigned block numbers).

The set of delay productions is given by

$$DP_s = \bigcup_{\substack{l_1,l_2 \in L \\ i \in [1,m] \\ j \in [1,n]}} \left\{ {}^{l_1}Z_{j,s}\,{}^{l_2}Y_{i,s} \rightarrow {}^{l_2}Y_{i,s}\,{}^{l_1}Z_{j,s} \right\}.$$

An output may always be delayed past a later input from the same state.

Requirement 5.4:

$$\forall s \in S, \alpha \in E, \quad \langle \alpha,\alpha \rangle \notin prereq_s$$

No (externally visible) event is ever a prerequisite of itself.

Requirement 5.5:

$$\left(\beta \in {}^{O}E \quad \text{and} \quad \alpha \in {}^{I}E\right) \Rightarrow \langle \beta,\alpha \rangle \notin prereq_s$$

No (externally visible) output event can ever be a prerequisite for an (externally visible) input event.

The transaction productions must be consistent with the prerequisite relation as defined above. If a temporally related input event non-terminal and output event non-terminal appear in the right hand side of a transaction production, then *prereq* must be defined appropriately.

Requirement 5.6:

$$\forall s \in S,\, w,y \in E,$$
$$\langle w,y\rangle \in prereq_s \Rightarrow \left(\exists p \in BPT_s,\, v,x,z \in \hat{V}_s^*,\, t,u \in \hat{V}_s \quad \text{s.t.} \quad rhs(p) = vtxuz \wedge t \underset{G_s}{\Rightarrow} w \wedge u \underset{G_s}{\Rightarrow} y \right)$$

If two events are temporally related, then there must exist a transaction production that enforces this relation. Events from different blocks cannot be temporally related.

The set of shuffle productions, SP_s, is constructed through a two-stage process as described in Algorithm 5.1. Except for the addition of labels to the non-terminals symbols, this algorithm is identical to Algorithm 3.1.

Algorithm 5.1: Composite Shuffle Production Construction with Labels

$$SP_s := \varnothing$$

First add the productions within each block

$$\forall p \in BPT_s,\; v,x,z \in \hat{V}_s^*,\, w,y \in \hat{V}_s \quad \text{s.t.} \quad rhs(p) = vwxyz,$$
$$\textbf{if } \left(w \notin \bigcup_{\substack{l \in L \\ j \in [1,n]}} \left\{ {}^{l}Z_{j,s} \right\} \vee y \notin \bigcup_{\substack{l \in L \\ i \in [1,m]}} \left\{ {}^{l}Y_{i,s} \right\} \right) \wedge \langle w,y\rangle \notin prereq_s \textbf{ then } SP_s := SP_s \cup \{ wy \to yw \}$$

Now add the inter-block shuffle productions

$$\forall k_1 \in [1,b-1], k_2 \in [k_1+1,b],\; p_1,p_2 \in BPT_s,\, u,w,x,z \in \hat{V}_s^*,\, v,y \in \hat{V}_s$$
$$\text{s.t.} \quad lhs(p_1) = T_{k_1,s} \wedge lhs(p_2) = T_{k_2,s} \wedge rhs(p_1) = uvw \wedge rhs(p_2) = xyz,$$
$$\textbf{if } \left(v \notin \bigcup_{\substack{l \in L \\ j \in [1,n]}} \left\{ {}^{l}Z_{j,s} \right\} \vee y \notin \bigcup_{\substack{l \in L \\ i \in [1,m]}} \left\{ {}^{l}Y_{i,s} \right\} \right) \textbf{ then } SP_s := SP_s \cup \{ wy \to yw \}$$

Given the above description of modular specification, it should be apparent that the minimum parameters for describing a module with event labels can be stated by the tuple $\langle m,n,b,S,N,L\rangle \cup \bigcup_{s\in S}\langle prereq_s, r_s, BPT_s\rangle$. Thus, by specifying the values for these parameters, we fully describe the module under the representation.

Composition involving labelled events is done in a manner analogous to that described in Section 4.2.2. The only difference is that for each connection, there are now $|L|$ unification operations, rather than just a single one.

5.3 MODULAR TRANSACTION PROPERTIES

In the previous chapters, we have modelled the activities of a module or a group of modules as a sequence of events. The externally visible structure of the event sequences was contained in the language described by the module's grammar. Each possible event sequence (or execution sequence) is called a *behaviour*. Each behaviour is a sentence in the language formed from the corresponding module grammar.

In the most general sense, a *property* is a (desired) quality (of a system) that is relevant to the modelling exercise. In our case, a property places constraints on the way that a module operates. The property may only allow certain types of behaviours to occur. Using our representation, we shall concentrate on the notion of a property as a restriction on the types of transactions that a module can undergo.

Given an arbitrary modular grammar, a property will restrict the possible combinations of labelled non-terminal symbols that can occur on the right-hand side of any transaction production for the module. The restrictions will be expressed as relations whose domain is the set of labels. In turn, this will limit the possible label combinations that may be attached to events in valid sentences of the module language.

In general, we have a transaction production with a number of input and output non-terminal symbols forming its right-hand side. A property will constrain the relationship between the labels attached to the various input symbols in a transaction production, and those attached to the output symbols.

Before we consider actual properties, we must discuss some notion of causality. Consider the transaction production

$$T_{1,s} \to {}^{l_1}Y_{1,s}\,{}^{l_2}Y_{2,s}\,{}^{l_3}Y_{3,s}\,{}^{l_4}Y_{4,s}\,{}^{l_1'}Z_{1,s}\,{}^{l_2'}Z_{2,s}\,{}^{l_3'}Z_{3,s}\,{}^{l_4'}Z_{4,s}.$$

Our notion of properties will constrain the relationship between causally related input and output events, as defined by the *prereq* relation. In the above example, if we know that $\forall l_1, l_2 \in L, \left\langle {}^{l_1}\alpha_1, {}^{l_2}\beta_1 \right\rangle \in prereq_s \wedge \left\langle {}^{l_1}\alpha_2, {}^{l_2}\beta_1 \right\rangle \in prereq_s$, then the label l_1' on non-terminal ${}^{l_1'}Z_{1,s}$ would be constrained by the labels l_1 and l_2 on ${}^{l_1}Y_1$ and ${}^{l_2}Y_2$, respectively. In general, there will be a relationship between an input non-terminal and all output non-terminals that follow it in the sequence in the right-hand side of each transaction production. This relationship will be mirrored in the definition of the prerequisite relation for events with different labels.

The relation ρ is *transitive* iff $x\rho y$ and $y\rho z$ imply $x\rho z$. A *modular transaction property* is expressed as a transitive relation $\rho \subseteq L \times L$ between the labels associated with the group of input events and the labels associated with the group of output events that make up a particular complete transaction. As will be demonstrated in the next chapter, many practical properties can be expressed in this form.

Here is the mathematical specification of a property, based upon transitive relation ρ, for a single module:[59]

$$\forall s \in S, l_1, l_2 \in L, w \in \bigcup_{i\in[1,m]} \left\{ {}^{l_1}\alpha_i \right\}, y \in \bigcup_{j\in[1,n]} \left\{ {}^{l_2}\beta_j \right\}, \quad \langle w, y \rangle \in prereq_s \Rightarrow l_1 \rho l_2$$

This specification implies that if an input event α is a prerequisite for an output event β, then the labels l_1 and l_2 of events α and β must be such that $l_1 \rho l_2$. If the above holds for a modular specification, then we can conclude that the module in question satisfies the desired property.

[59]The notation $a\rho b$ is equivalent to $\langle a, b \rangle \in \rho$.

Some interesting modular transaction properties include:

- confidentiality (dominates operator on confidentiality labels)
- integrity (dominates operator on confidentiality labels)
- time (output events occur at a later time than input events)
- probability (probability of an output event occurring is less than or equal to the probability of the corresponding input event(s) occurring)

Some of these properties will be described more fully in Chapter 6.

By Requirement 5.6, the prerequisite relation constrains the transaction production set BPT_s in that there must exist production(s) that enforce the desired temporal relationships.

This notion of properties can then be easily applied to the grammar with labels described in Section 5.2. The only extension required is found in the statement of the transaction productions. The following table provides examples of unconstrained and constrained transaction production sets and causality relations for an m,n-General Module:

Unconstrained	**Modular Transaction Property**
$\bigcup_{\substack{l_1,l_2,\ldots,l_m \in L \\ l'_1,l'_2,\ldots,l'_n \in L}} \left\{ T_{1,s} \to {}^{l_1}Y_{1,s}\,{}^{l_2}Y_{2,s} \ldots {}^{l_m}Y_{m,s}\,{}^{l'_1}Z_{1,s}\,{}^{l'_2}Z_{2,s} \ldots {}^{l'_n}Z_{n,s} \right\}$	$\bigcup_{\substack{l_1,l_2,\ldots,l_m \in L \\ l'_1,l'_2,\ldots,l'_n \in L \\ \forall i \in [1,m], j \in [1,n], l_i \rho l'_j}} \left\{ T_{1,s} \to {}^{l_1}Y_{1,s}\,{}^{l_2}Y_{2,s} \ldots {}^{l_m}Y_{m,s}\,{}^{l'_1}Z_{1,s}\,{}^{l'_2}Z_{2,s} \ldots {}^{l'_n}Z_{n,s} \right\}$
$\forall l_1, l_2 \in L, i \in [1,m], j \in [1,n],$ $\left\langle {}^{l_1}\alpha_i, {}^{l_2}\beta_j \right\rangle \in prereq_s$	$\forall l_1, l_2 \in L, i \in [1,m], j \in [1,n],$ $l_1 \rho l_2 \Leftrightarrow \left\langle {}^{l_1}\alpha_i, {}^{l_2}\beta_j \right\rangle \in prereq_s$

The addition of the property constrains the possible label combinations of non-terminal symbols that are permitted in both the transaction productions and the prerequisite relation. The complete module specification with event labels and the property relation is specified by the tuple $\langle m,n,b,S,N,L,\rho \rangle \cup \bigcup_{s \in S} \langle prereq_s, r_s, BPT_s \rangle$.

5.4 COMPOSING WITH PROPERTIES

We will now consider the more general case where two arbitrary modules, described by modular grammars are composed. Given that each of the modules satisfies the desired property individually, we need to prove that the composite also satisfies the property.

To begin, the generalized composition formalism from the Section 4.2 will be adjusted to include the event labels and property relation.

Consider two modules:

$$\begin{gathered}\left\langle m^1, n^1, b^1, S^1, N^1, L^1, \rho^1 \right\rangle \cup \bigcup_{s \in S^2} \left\langle prereq_s^1, r_s^1, BPT_s^1 \right\rangle \\ \left\langle m^2, n^2, b^2, S^2, N^2, L^2, \rho^2 \right\rangle \cup \bigcup_{s \in S^2} \left\langle prereq_s^2, r_s^2, BPT_s^2 \right\rangle .\end{gathered}$$

For convenience, we assume that $L^1 = L^2 = L$. Also, as noted in Section 1.3, we are only considering the case where $\rho^1 = \rho^2 = \rho$. All of the other model elements are derived as described in Sections 4.2 and 5.2.

Based upon the connections, we will construct the composite specification

$$\left\langle m^C, n^C, b^C, S^C, N^C, L^C, \rho^C \right\rangle \cup \bigcup_{s \in S^C} \left\langle prereq_s^C, r_s^C, BPT_s^C \right\rangle .$$

where $L^C = L$ and $\rho^C = \rho$.

In our situation, we would expect that a connection is made between events with the same label, and that if it is made for a single label, then it is also made for every other label. Thus, we have the following requirement:

Requirement 5.7:

$$\begin{aligned}&\forall k_1, k_2 \in [1,2],\, l_1, l_2 \in L,\, j \in n^{k_1},\, i \in m^{k_2}, \\ &\quad \left\langle {}^{l_1}\beta_j^{k_1}, {}^{l_2}\alpha_i^{k_2} \right\rangle \in connected \Rightarrow \left(l_1 = l_2 \wedge \left(\forall l_3 \in L, \left\langle {}^{l_3}\beta_j^{k_1}, {}^{l_3}\alpha_i^{k_2} \right\rangle \in connected \right) \right) .\end{aligned}$$

No event can be relabelled as it passes through a connection, and if a connection is made for events with one particular label, then the connection must be made for events with all other labels.

The composition of the two modules with a property is constructed in exactly the same manner as the labelled event composition described in Section 5.2.

Lemma 5.1. The composition of two modules that obey a modular transaction property also obeys the modular transaction property.

Proof. In this case, we have modules[60]:

$$\langle m^1, n^1, b^1, S^1, N^1, L, \rho \rangle \cup \bigcup_{s \in S^1} \langle prereq_s^1, r_s^1, BPT_s^1 \rangle$$
$$\langle m^2, n^2, b^2, S^2, N^2, L, \rho \rangle \cup \bigcup_{s \in S^2} \langle prereq_s^2, r_s^2, BPT_s^2 \rangle:$$

that each obey the transaction property, and must show that any composition of the two

$$\langle m^C, n^C, b^C, S^C, I^C, N^C, L, \rho^C \rangle \cup \bigcup_{s \in S^C} \langle prereq_s^C, r_s^C, BPT_s^C \rangle$$

also has the property (i.e., $\rho^C = \rho$). In other words, we know

$$\forall k \in [1,2], s \in S^k, l_1, l_2 \in L, w \in \bigcup_{i \in [1, m^k]} \{{}^{l_1}\alpha_i^k\}, y \in \bigcup_{j \in [1, n^k]} \{{}^{l_2}\beta_j^k\}, \quad \langle w, y \rangle \in prereq_s^k \Rightarrow l_1 \rho l_2$$

and wish to prove that[61]

$$\forall s \in S^C, l_1, l_2 \in L, w \in \bigcup_{\substack{k \in [1,2] \\ i \in [1, m^C]}} \{{}^{l_1}\alpha_i^k\}, y \in \bigcup_{\substack{k \in [1,2] \\ j \in [1, n^C]}} \{{}^{l_2}\beta_j^k\}, \quad \langle w, y \rangle \in prereq_s^C \Rightarrow l_1 \rho l_2.$$

We shall prove the lemma for one arbitrary state $s \in S^C$. If it is true for that arbitrary state, then it is true for all other states of the composite system.

---

[60]We assume that $\forall \langle s_1, s_2 \rangle \in S^C, r_{s_1}^1 \cap r_{s_2}^2 \neq \varnothing$.

[61]Remember that, after the composition, the domain of the $prereq^C$ relation is $E^C \times E^C$.

As a general observation, notice that shuffle and delay productions have no impact on a module's satisfaction of a particular property. Only the prerequisite relations $prereq^1$ and $prereq^2$ and the way in which they are combined to produce the composite prerequisite relation $prereq_s^C$ affect the property.

We will proceed by induction on the number of connections between the modules. For the base case, consider two modules with no connections between them, $connected = \varnothing$. Under this condition, the composite prerequisite relation reduces to

$$\begin{aligned}&\forall \langle s_i, s_j \rangle \in S^C, \alpha, \beta \in E^1 \cup E^2,\\ &\quad \langle \alpha, \beta \rangle \in prereq *^C_{\langle s_i, s_j \rangle} \Leftrightarrow \langle \alpha, \beta \rangle \in prereq^1_{s_i} \vee \langle \alpha, \beta \rangle \in prereq^2_{s_j}\end{aligned}$$

$$\forall s \in S^C,\ prereq_s^C = \left\{ \langle \alpha, \beta \rangle \,\middle|\, \langle \alpha, \beta \rangle \in prereq *^C_s \wedge \alpha, \beta \in E^C \right\}.$$

Since the modules have the property individually, this implies that

$$\forall s \in S^C, l_1, l_2 \in L, w \in \bigcup_{\substack{k \in [1,2] \\ i \in [1, m^C]}} \left\{ {}^{l_1}\alpha_i^k \right\}, y \in \bigcup_{\substack{k \in [1,2] \\ j \in [1, n^C]}} \left\{ {}^{l_2}\beta_j^k \right\}, \quad \langle w, y \rangle \in prereq_s^C \Rightarrow l_1 \rho l_2$$

as required.

For the inductive case, assume that we have k connections (i.e. $|connected| = k$), and that the property holds for this situation. Now consider the addition of a new connection, $\langle \delta, \varphi \rangle$, such that $connected' = connected \cup \{ \langle \delta, \varphi \rangle \}$. We have that $prereq_s^C \subseteq prereq_s^{C'}$. The new connection affects the satisfaction of the property only where we have $\langle \alpha, \beta \rangle \in prereq_s^{C'} \setminus prereq_s^C$ for α an input and β an output. This situation is pictured in Figure 5.1.

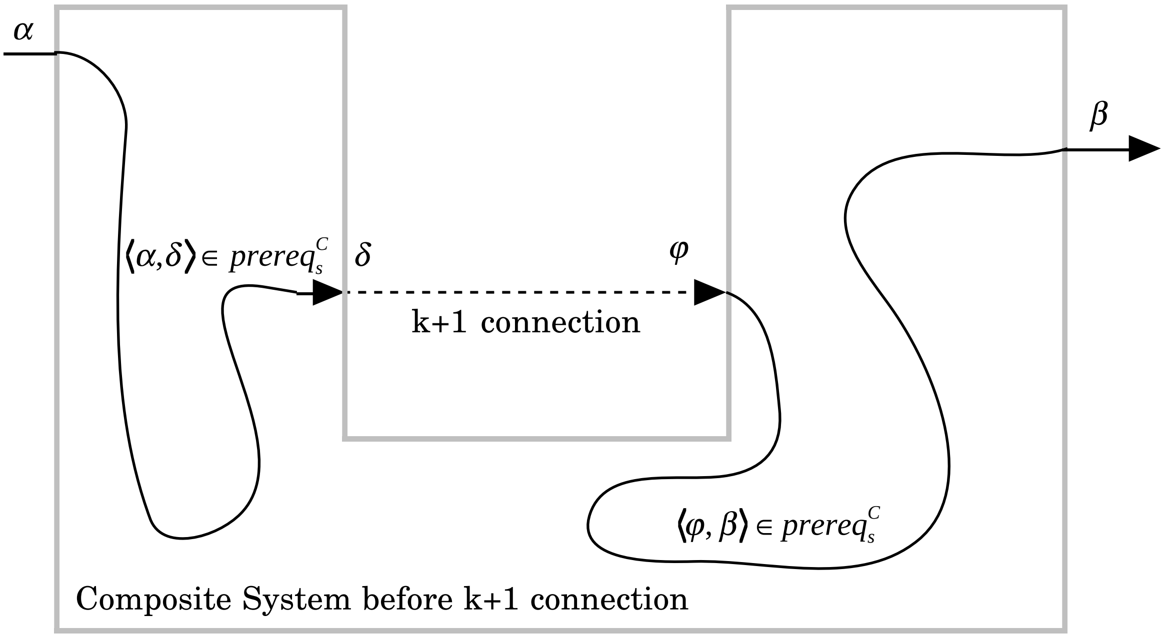

Figure 5.1: Addition of $k+1$ Connection

Now, by the inductive assumption, we know that

$$\forall l_1,l_2 \in L, \alpha \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}}\left\{{}^{l_1}\alpha_i^k\right\}, \delta \in \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}}\left\{{}^{l_2}\beta_j^k\right\}, \quad \langle\alpha,\delta\rangle \in prereq_s^C \Rightarrow l_1\rho l_2$$
$$\forall l_3,l_4 \in L, \varphi \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}}\left\{{}^{l_3}\alpha_i^k\right\}, \beta \in \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}}\left\{{}^{l_4}\beta_j^k\right\}, \quad \langle\varphi,\beta\rangle \in prereq_s^C \Rightarrow l_3\rho l_4 .$$

By Requirement 5.7, we know that

$$\forall l_2,l_3 \in L, \delta \in \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}}\left\{{}^{l_2}\alpha_j^k\right\}, \varphi \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}}\left\{{}^{l_3}\beta_i^k\right\}, \quad \langle\delta,\varphi\rangle \in connected \Rightarrow l_2 = l_3 .$$

Combining these, we have that

$$\begin{gathered}\forall l_1,l_2,l_3,l_4 \in L, \alpha \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}}\left\{{}^{l_1}\alpha_i^k\right\}, \delta \in \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}}\left\{{}^{l_2}\beta_j^k\right\}, \varphi \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}}\left\{{}^{l_3}\alpha_i^k\right\}, \beta \in \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}}\left\{{}^{l_4}\beta_j^k\right\}, \\ \langle\alpha,\delta\rangle \in prereq_s^C \wedge \langle\delta,\varphi\rangle \in connected \wedge \langle\varphi,\beta\rangle \in prereq_s^C \Rightarrow l_1\rho l_2 \wedge l_2 = l_3 \wedge l_3\rho l_4\end{gathered} .$$

Thus, since $\forall l_1, l_2, l_3, l_4 \in L, (l_1 \rho l_2 \wedge l_2 = l_3 \wedge l_3 \rho l_4) \Rightarrow (l_1 \rho l_2 \wedge l_2 \rho l_4) \Rightarrow l_1 \rho l_4$, for all paths through the $k+1$ connection, the property is maintained.

Since the choice of the state in which the lemma has been proven to be composable was arbitrary, any state that is a member of S^C could have been chosen, and all states that are a member of S^C are composable.■

Theorem 5.1. All modular transaction properties are composable.

Proof. Given a group of k modules to be composed, we can apply Lemma 5.1 $k-1$ times, beginning with a pair of modules, composing them, then successively adding a single module to the composite to build up the overall composition.■

5.5 SUMMARY

This chapter began by describing an extension to the formalism, based upon the notion of event labels. It then went on to introduce a definition of properties using these labels. Appropriate extensions to the composition algorithm were presented. Finally, a proof of the composability of the class of transitive label-based properties was provided.

6 APPLICATION EXAMPLES

6.1 INTRODUCTION

A safe system is one that has a set of desired properties. These properties generally ensure that the system behaves in a manner consistent with our own expectation, often as expressed by stated policy. This chapter will demonstrate the value and applicability of the modelling strategy developed by applying it to a number of areas of safe systems, as well as some other areas, like digital logic and protocol specification. The examples provided come from a range of previous and current work in the area.

We shall present the examples in this chapter in two groups. First, we will describe an example that uses modular transaction properties to describe the actual property goals for the system. This will allow the use of the composition results from Chapter 5, thereby illustrating the applicability and utility of our composition theory. We will describe a confidentiality property that is based upon Sutherland's deducibility, as described in Section 2.2.4.

Next, we will present several examples where the target properties do not fall directly into the class of modular transaction properties. In some cases the properties can be translated into equivalent MTPs, so that our composition results can be applied. In other cases, the examples serve as pointers to potential future extensions to the modelling technique. The examples are not presented to identify the limitations of our theory, but rather to demonstrate the flexibility of our modelling methodology, by making extensive use of the grammatical specification technique, including labelled events, to capture the desired behaviour.

6.2 PREREQUISITE CONFIDENTIALITY

The presentation of the example of this section serves two purposes. First, it provides an example of a practical property that falls into the class of modular transaction properties and to which Theorem 5.1 can be applied. Second, it introduces a new confidentiality definition that provides a number of significant improvements over previous definitions.

One of the most-studied properties in the area of formal modelling of systems is confidentiality. Originally termed (computer) security, confidentiality has now been classified as one of several ideas involved in an overall notion of computer security.[62] Confidentiality ensures that information is accessed for observation only by those duly authorized to have such access. Attempts at unauthorized access must be detected and prevented. This simple definition has led to a variety of mathematical formulations of confidentiality, all with varying abilities to capture the behaviour of real systems. Many intuitively reasonable confidentiality properties are not preserved by implementation or by composition.[63]

In this section, we will develop a notion of confidentiality that is based upon Sutherland's deducibility, as described in Section 2.2.4. This statement of confidentiality will then be mapped into an appropriate modular transaction property.

Our definition of confidentiality, called *prerequisite confidentiality*, is defined as follows:

---

[62] Donn B. Parker. Restating the Foundation of Information Security. In *Proceedings of the Second Annual North American System Security Symposium*, Toronto, Ontario, October 1991. Also Heather M. Hinton. An Environment-Based Approach to the Composition of Safe Systems, Masters thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, January 1993.
[63] E.S. Lee, P.I.P. Boulton, B.W. Thomson, and R.E. Soper. Composable Trusted Systems. Technical Report 272, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, May 1992, Final Report of Contract W2213-9-6351/01-SZ, pp. 210–214.

> Given an execution sequence with events of varying confidentiality levels, knowledge of low level events in the sequence provide no information about when or if higher level events occur.

We can easily express this confidentiality notion within the module framework, by requiring that high-level, prerequisite input events to a module never result in lower level output events being generated. If an input event α is a prerequisite for output event β, then our knowledge that β has occurred allows us to deduce that event α has (previously) occurred. Therefore, the confidentiality label attached to β must dominate the label of event α, in order to prevent our knowledge of β's occurrence giving us information about higher-level events.

Consider a set of partially ordered confidentiality labels L_C with the dominates relation $\rhd_C$. Then a module specification $\langle m,n,b,S,N,L\rangle \cup \bigcup_{s\in S} \langle prereq_s, r_s, BPT_s\rangle$ has the prerequisite confidentiality property if and only if

$$\forall s \in S, l_1, l_2 \in L, w \in \bigcup_{i\in[1,m]} \left\{ {}^{l_1}\alpha_i \right\}, y \in \bigcup_{j\in[1,n]} \left\{ {}^{l_2}\beta_j \right\}, \quad \langle w,y\rangle \in prereq_s \Rightarrow l_2 \rhd_C l_1$$

This requires that the confidentiality level of the output events dominate the confidentiality level of any prerequisite input events.

Clearly, since $\rhd_C$ is a partial order, it is a transitive relation. Therefore, we can conclude that Theorem 5.1 holds, and any composition of modules with the prerequisite confidentiality property will result in a composite that also has the prerequisite confidentiality property.

As an example, Figure 6.1 shows the AND Module from Section 3.4.3, with the prerequisite confidentiality property imposed.

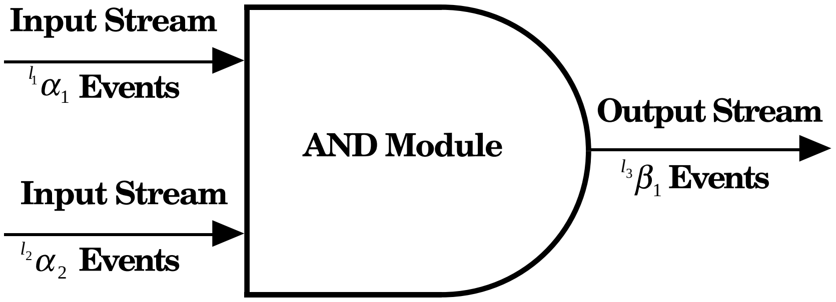

Figure 6.1: AND Module with Prerequisite Property

Define the event label set $L_C = \{TS, S, U\}$ to correspond to confidentiality levels "Top Secret", "Secret", and "Unclassified". Clearly, $TS \triangleright_C S \triangleright_C U$ and the transitive relation ρ expresses this. The specification is as follows:

$$m=2 \;\; n=1 \;\; b=1 \;\; S=\{s_0\} \;\; N=\varnothing \;\; L_C=\{TS,S,U\} \;\; \rho=\left\{\begin{matrix}\langle TS,TS\rangle,\langle TS,S\rangle,\langle TS,U\rangle \\ \langle S,S\rangle,\langle S,U\rangle,\langle U,U\rangle\end{matrix}\right\}$$

$$prereq_{s_0} = \bigcup_{\substack{l_1,l_2\in L_C \\ i\in[1,2] \\ l_2 \triangleright_C l_1}} \left\{\left\langle {}^{l_1}\alpha_i, {}^{l_2}\beta_1\right\rangle\right\} \;\; r_{s_0} = [0,\infty] \;\; BPT_{s_0} = \bigcup_{\substack{l_1,l_2,l_3\in L_C \\ l_3 \triangleright_C l_1 \\ l_3 \triangleright_C l_2}} \left\{T_{1,s_0} \rightarrow {}^{l_1}Y_{1,s_0}\, {}^{l_2}Y_{2,s_0}\, {}^{l_3}Z_{1,s_0}\right\}$$

The prerequisite relation is defined so that the labels attached to output events always dominate those of the two input events that were prerequisites. The transaction production is defined in an analogous manner.

Prerequisite confidentiality, as defined in this section, provides a simple, intuitive notion of confidentiality. Because it is an instance of a modular transaction property, it is composable. Our notion of confidentiality improves upon that proposed by Sutherland by more realistically describing nondeterministic behaviour. If it is possible for an input event to be a prerequisite for an output event, then prerequisite confidentiality requires that the desired event label relationship hold. Thus, confidentiality violations, no matter how unlikely, are made visible by the model.

6.3 SUBJECTS AND OBJECTS

Before we consider the various property applications, it is important to recognize that a different metaphor for representing systems exists in much of the previous literature. Therefore, if we wish to describe some of the previous models and properties within our model, then we must be able to translate a subject and object-based representation into one based on our modules. This section will instantiate a subject and object representation using the module-based approach developed in this thesis.

Under many modelling frameworks, there are three types of entities that play a role in the behaviour of a system. A *subject* is a person, or agent acting on his or her behalf, such as a process. Subjects are active entities that perform actions to accomplish various tasks, thus providing inputs to the system. The *system* is a special subject that controls when and how other subjects access the objects that are available. It accepts the inputs from other subjects and directs or processes them in a certain manner. An *object* is a passive entity that is maintained by the system. Examples of objects include files, video screens, and keyboards.

Under various safety properties, both subjects and objects are given a label containing information about confidentiality level, integrity level, etc. A *safety property* is a constraint on the relationship between the assigned labels of interacting subjects and objects. If a system is proven to guarantee that the constraint always holds, then we say that the system has the desired property.

We will consider three types of interactions between subjects and objects. A subject *observes* an object when it performs a read-only operation in which information is transferred from the object to the subject. An example of an observation would be reading a file. Notice that the subject must send a request to an object to initiate the observation.

The second operation involves *alteration* of the object by the subject. This is similar to writing data to a file.

The third type of access involves *invocation* of one subject by another. This is analogous to execution of one process by another. Under most modelling frameworks, these are the only operations represented.

In most secure computing systems, the hardware and software subsystem responsible for enforcing the desired security policy is known as the *Trusted*

Computing Base. The TCB includes the implementation of a *reference monitor*.[64] The reference monitor mediates all accesses to objects by subjects.

It is useful to consider how the subject and object model can be represented under the event trace framework. At first glance, it would appear that there is no notion of subject and object, only events and how they are interrelated. Under this framework, labels are attached to events, and properties expressed as constraints on the labels of interacting events.

It is implicitly assumed that observe, alter, and invoke events are caused by subjects. A subject's label can then be attached to all events generated by that subject. Objects can be represented as modules that accept observation and alteration events and (if applicable) output an event representing the production of any returned data. The composition of all of the modules present in a particular situation forms the system. The label attached to any events generated by an object is that of the object itself. As far as the modelling exercise is concerned, only subjects can spontaneously generate events—objects can only generate events in response to other events. The behaviour of the subjects falls outside the modelled system.

[64] J.P. Anderson. *Computer Security Technology Planning Study*, ESD–TR–73–51, vol. I, AD–758 206, ESD/AFSC, Hanscom AFB, Bedford, Mass., October 1972.

The following structure can be used to depict an object using modular elements:

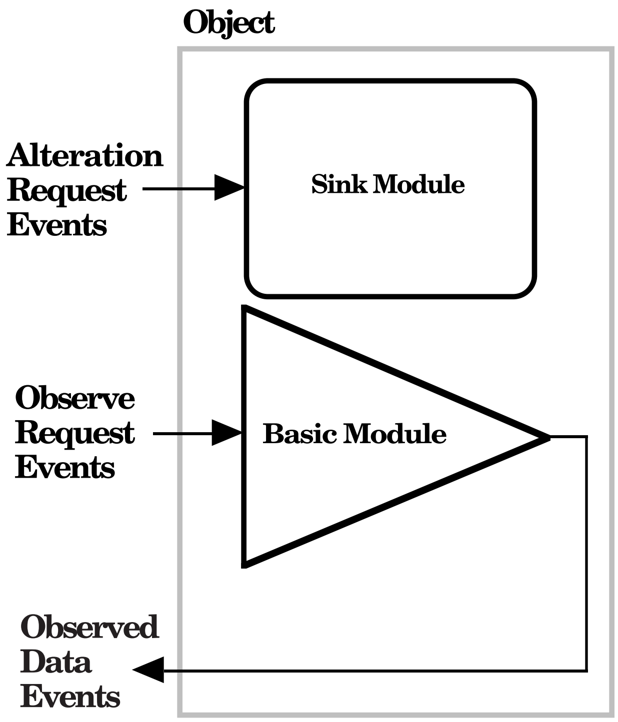

Figure 6.2: Object Modular Representation

In the above representation, the object structure can only respond to events received.

A practical system would have many different subjects and objects, so some kind of interconnection network would have to be modelled to allow all possible exchanges of events.

In a practical system, it might be desirable to filter the stream of events directed to an object module by interposing mediator modules. The mediators might only allow events with certain kinds of labels to pass through to the object. Other events would simply be ignored, and not passed to the object. An extension of this would allow modelling of systems where the labels placed on events are dynamic and depend on previous behaviour.

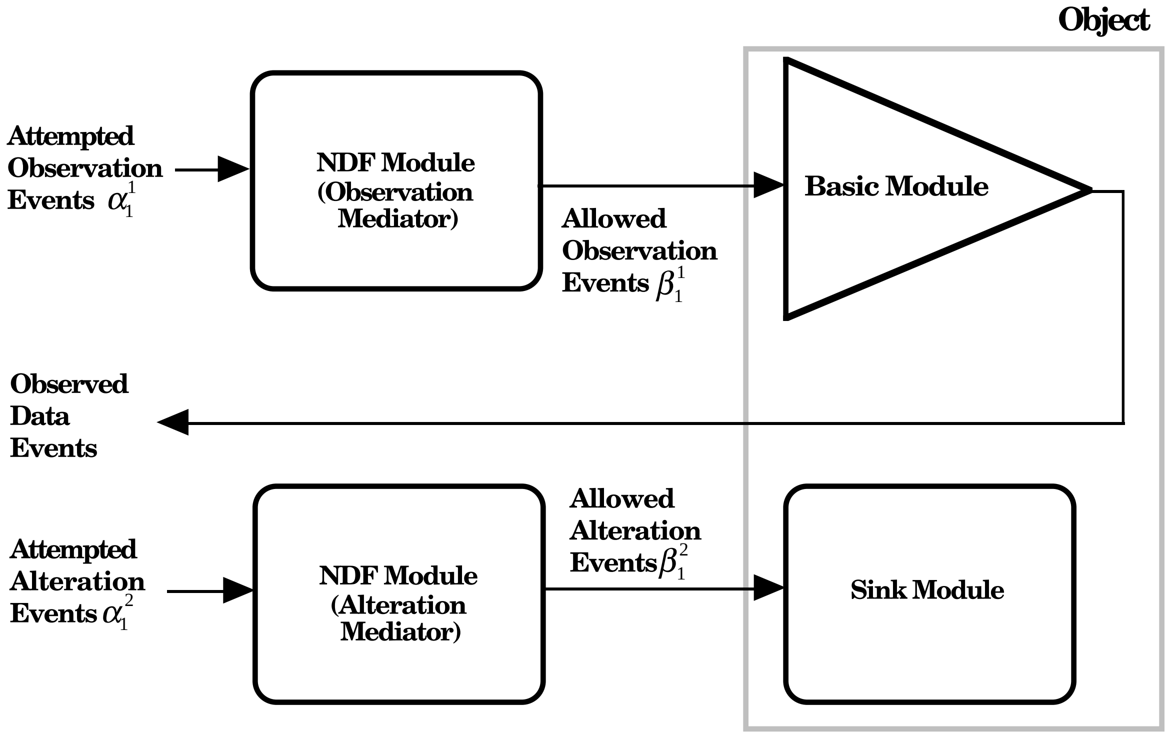

Figure 6.3: Object Observation and Alteration Interface

The mediator modules shown in Figure 6.3 actually form part of the reference monitor described above. If the reference monitor is trusted, then access to a subject by an object will be granted by the TCB if and only if that subject should have that access. The decision as to whether or not access is permitted is based upon the relative security levels of the subject and object. Such relationships can be usually be described by an equivalent modular transaction property on the TCB.

The important conclusion that can be drawn from this section is that the subject and object model can be easily converted to the modular formulation developed in this thesis. As a result, what has been done is directly applicable to much of the previous work done with subjects and objects. The following sections will help to illustrate this point by taking some previous subject and object safety property examples and instantiating them in the language-based modular framework.

6.4 BELL-LAPADULA CONFIDENTIALITY

This section will consider an important previous modelling approach to confidentiality in the context of our modular representation. Because the definition of confidentiality in this approach is based upon subjects accessing objects through a Trusted Computing Base (TCB), the target property cannot be directly expressed as a modular transaction property. However, since the basis for granting access to subjects is based upon a lattice of confidentiality levels and an appropriate dominates operator, it is possible to translate this modelling approach into one that is based upon modular transaction properties. The result would be a property that similar to the prerequisite confidentiality example described in Section 6.2.

The particular definition of confidentiality to be used in this section is one of the original explications of computer security.[65] It has formed the basis for a wide variety of security policies, including the U.S. Department of Defence Trusted Computer Security Evaluation Criteria. Although it is somewhat simpler than many of the more recent confidentiality definitions, it has gained widespread acceptance because it can be easily implemented and related to most practical computer systems. This section will serve to illustrate the application of the property of confidentiality to the language-based modular representation of this thesis.

There are two types of access control normally associated with most security definitions. Discretionary access control or DAC is control of access to objects by subjects. This works much like the UNIX permission scheme, where the owner of a particular file (object) can specify access to that file by other users on the system. Mandatory access control or MAC restricts access based on the sensitivity of the information contained in the object, and the authorization of subjects wishing to access that object. It is the latter form of access control that we wish to investigate in the following sections.

---

[65] D. Elliott Bell and Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report, MITRE Corporation, Bedford, MA, March 1973.

The Bell-LaPadula definition works by indexing each subject and each object with a confidentiality level. For example, we might have a set of levels[66] L_C. These levels might allow us to classify information and users as "Confidential, Financial", or "Top Secret, Research". This classification, in turn, allows us to restrict access to information based upon both the information's confidentiality level and the allowed access privileges or clearance of a particular user or viewer. For example, a user with "Top Secret, All" clearance should be permitted to observe any data, while a user with "Confidential, Research" clearance should only have access to Confidential or Unclassified data in the Research domain. Users with low clearances should never be permitted to access, either directly or indirectly, information at higher levels. We will ascribe the term *confidentiality-safe* to systems that have the confidentiality property.

The binary dominates relation $\triangleright_C$ induces the required transitive relation over L_C. This provides a mathematical mechanism for comparing confidentiality levels of different objects or users. The lattice-based structure for confidentiality labels was first introduced by Denning in 1976[67].

If we consider a set of subjects S, a set of objects O, and a function $cl: S \cup O \to L_C$ that returns the confidentiality label of a particular subject or object, then the BLP confidentiality property is represented formally by the following two axioms:

$$\forall s \in S, o \in O, \quad s \text{ can observe } o \Rightarrow cl(s) \triangleright_C cl(o)$$

$$\forall s \in S, o \in O, \quad s \text{ can alter } o \Rightarrow cl(o) \triangleright_C cl(s)$$

These two axioms were referred to as the *simple security property* and the *$*$-security property*, respectively. The policy that they enforce is often referred to as "read down, write up". Intuitively, a user with a high confidentiality level is able to observe (down) those objects with lower confidentiality levels. A user with a low confidentiality level can only write (up) to those objects with a higher

---

[66]The choice and number of labels that are possible has no impact in the development, as was shown by the general case proofs in Chapter 5.

[67]D.E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*. 19(5):236–242, May 1976.

confidentiality level. The net effect is that information is only allowed to travel from low level entities to high level entities, but never vice versa.

Within the event system framework of this thesis, we can represent this policy by defining appropriate grammars for the observation and alteration mediator modules defined above. Consider an object $o \in O$. Then the following would be the specification of the observation mediator:

$$m^1 = 1 \;\; n^1 = 1 \;\; b^1 = 1 \;\; S^1 = \{s_0^1\} \;\; N^1 = \varnothing$$

$$\forall l \in L_C, \langle {}^l\alpha_1^1, {}^l\beta_1^1 \rangle \in prereq_{s_0^1}^1 \Leftrightarrow l \rhd_C cl(o) \;\; r_{s_0^1}^1 = [0, \infty]$$

$$BPT_{s_0^1}^1 = \bigcup_{\substack{l \in L_C \\ l \rhd_C cl(o)}} \left\{ T_{1,s_0^1}^1 \to {}^lY_{1,s_0^1}^1 \, {}^lZ_{1,s_0^1}^1 \right\} \cup \bigcup_{\substack{l \in L_C \\ \neg(l \rhd_C cl(o))}} \left\{ T_{1,s_0^1}^1 \to {}^lY_{1,s_0^1}^1 \right\}$$

If an input event with a confidentiality label that dominates $cl(o)$ occurs, then the appropriate output event is generated by the mediator. If the input event label does not dominate $cl(o)$, then no output event is generated.

The following would be the specification of the alteration mediator:

$$m^2 = 1 \;\; n^2 = 1 \;\; b^2 = 1 \;\; S^2 = \{s_0^2\} \;\; N^2 = \varnothing$$

$$\forall l \in L_C, \langle {}^l\alpha_1^2, {}^l\beta_1^2 \rangle \in prereq_{s_0^1}^2 \Leftrightarrow cl(o) \rhd_C l \;\; r_{s_0^2}^2 = [0, \infty]$$

$$BPT_{s_0^2}^2 = \bigcup_{\substack{l \in L_C \\ cl(o) \rhd_C l}} \left\{ T_{1,s_0^2}^2 \to {}^lY_{1,s_0^2}^2 \, {}^lZ_{1,s_0^2}^2 \right\} \cup \bigcup_{\substack{l \in L_C \\ \neg(cl(o) \rhd_C l)}} \left\{ T_{1,s_0^2}^2 \to {}^lY_{1,s_0^2}^2 \right\}$$

The grammars as defined above have the resulting effect that events are ignored or routed to the object, depending on $cl(o)$. This grammar removes the nondeterminism of the original NDF module and replaces it with a deterministic choice, based upon the label of the input event.

The value of $cl(o)$ is assumed to be constant. Section 6.5.2 will consider a situation where an object's label can change over time. In each case, the mediator modules themselves do not change the labels that are attached to incoming and outgoing events.

6.5 BIBA'S INTEGRITY

Although much attention has been paid to research into developing systems that obey the confidentiality property, integrity is quickly becoming increasingly important. With the move from military to commercially oriented policy development, many see integrity as one of the properties most deserving attention in the safety area.

This section will begin by describing a popular integrity definition, and then continue to demonstrate how the modelling approach developed in this thesis can be easily applied to it. The definitions of integrity to be used in this section were originally introduced by Biba in 1977.[68] As in the case of confidentiality, the policies presented will relate to Mandatory Access Control or MAC. A separately enforced and implemented Discretionary Access Control (DAC) policy is assumed to exist independently.

Biba's integrity definitions were introduced long before research into issues of composability began. Accordingly, there was no attempt to describe how the satisfaction of the various integrity properties is affected by the interconnection of systems. As in the case of BLP confidentiality described in Section 6.4, the properties cannot be directly expressed as modular transaction properties, because they involve subjects accessing objects under the authority of a Trusted Computing Base (TCB). As in the case with confidentiality, however, the basis for access decisions is a dominates operator between integrity levels. This implies that a translation to a transitive modular transaction property is possible.

[68] K.J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report TR-3153, MITRE Corporation, Bedford, MA, April 1977.

6.5.1 Strict Integrity

The strict integrity policy can be considered the dual of the security policy developed by Bell and LaPadula. Under Biba's formulation, the set of integrity levels might be *crucial*, *very important*, and *important*, representing the potential impact that could occur if the corresponding labelled data was modified improperly. The policy consists of three axioms. If we consider a set of subjects S, a set of objects O, a set of partially ordered integrity labels L_I, a dominance relation $\trianglerighteq_I$, and a function $il : S \cup O \to L_I$ that returns the integrity label of a particular subject or object, then the three axioms can be expressed as

$$\forall s \in S, o \in O, \quad s \text{ can observe } o \Rightarrow il(o) \trianglerighteq_I il(s)$$

$$\forall s \in S, o \in O, \quad s \text{ can alter } o \Rightarrow il(s) \trianglerighteq_I il(o)$$

$$\forall s_1, s_2 \in S, \quad s_1 \text{ can invoke } s_2 \Rightarrow il(s_1) \trianglerighteq_I il(s_2)$$

These axioms are designed to prevent high-integrity information from being (adversely) affected by lower integrity information. The first axiom ensures that high-integrity subjects are only able to read objects whose integrity level dominates their own. The second ensures that modifications to objects are only performed by subjects with dominating integrity levels. The third ensures that a subject dominates the integrity level of any subject that it invokes. In each case, the integrity levels of both subjects and objects are static. We will use the term *strict integrity-safe* to refer to systems that have the integrity property.

As with confidentiality, we must map this notion of subjects and objects into our own event-based formalism. Once again, we can define appropriate mediator modules to control observation and alteration access to objects.

Representing the invocation of subjects can be accomplished by adding appropriate mediator structures to subjects. An invocation of a subject is then treated in the same manner as the alteration of an object.

6.5.2 Low-Water Mark Policy on Objects

An alternative formulation of the integrity policy can be created by having the integrity labels of objects change depending on the integrity labels of the subjects that access them.[69] The following two axioms describe this behaviour:

$$\forall s \in S, o \in O, \quad s \text{ can observe } o \Rightarrow il(o) \rhd_I il(s)$$

$$\forall s \in S, o \in O, \quad s \text{ alters } o \Rightarrow il'(o) = \mathrm{glb}(il(s), il(o))$$

The glb binary operator returns the greatest lower bound or meet of its two arguments. If $i_C = \mathrm{glb}(i_A, i_B)$, then $i_A \rhd i_C$ and $i_B \rhd i_C$ and there is no i_D such that $i_A \rhd i_D$, $i_B \rhd i_D$, and $i_D \rhd i_C$.

The first axiom can be implemented using an appropriately defined observation mediator module. The second axiom is implemented by employing the following structure to maintain the integrity level associated with a particular object. All alterations of objects by subjects are permitted, although they do have an impact on future observation accesses.

[69] This is a variant of Biba's low-water mark policy on objects. Under Biba's formulation, the integrity levels of *both* subjects and objects change.

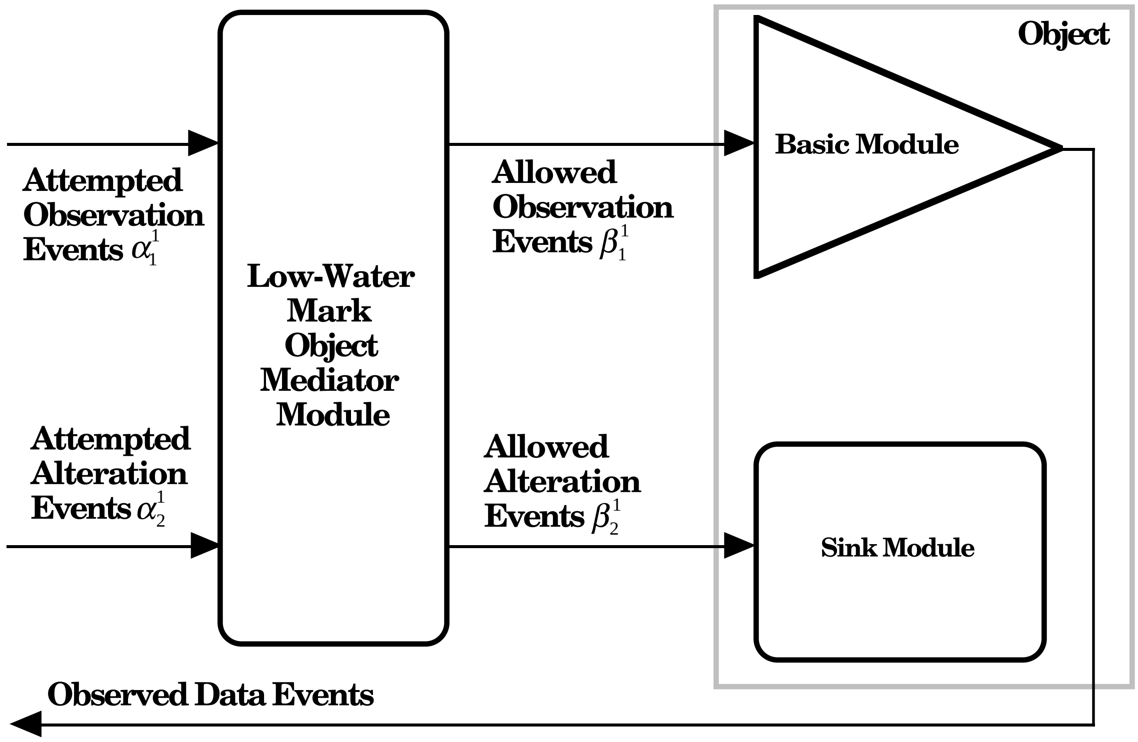

Figure 6.4: Low-Water Mark Object Interface

Figure 6.4 shows a new, unified mediator module controlling both observe and alter access to an object. Whereas before it was possible to have separate observe and alter mediator modules, here they must be unified because of the relationship between observation and alteration defined by the low-water mark policy.

We will now consider the grammatical representation of the above mediator module. To simplify the development, we will consider a system with integrity label set $L_I = \{i_1, i_2, i_3\}$ with $i_3 \triangleright_I i_2$ and $i_2 \triangleright_I i_1$. It is assumed that the object has integrity level i_3 initially.

$$m^1 = 2 \;\; n^1 = 2 \;\; b^1 = 1 \;\; S^1 = \left\{s_1^1, s_2^1, s_3^1\right\} \;\; N^1 = \left\{\left\langle s_3^1, s_2^1\right\rangle, \left\langle s_2^1, s_1^1\right\rangle\right\}$$

$$\forall k \in [1,3],$$

$$\begin{matrix} \forall l \in L_I, \left\langle {}^l\alpha_1^1, {}^l\beta_1^1\right\rangle \in prereq_{s_k^1}^1 \Leftrightarrow i_k \rhd_I l, \\ \left\langle {}^l\alpha_2^1, {}^l\beta_2^1\right\rangle \in prereq_{s_k^1}^1 \Leftrightarrow l \rhd_I i_k \end{matrix} \quad r_{s_k^1}^1 = [0, \infty]$$

$$BPT_{s_0^k}^1 = \bigcup_{\substack{l \in L_I \\ i_k \rhd_I l}} \left\{T_{1,s_k^1}^1 \rightarrow {}^lY_{1,s_k^1}^1 \, {}^lZ_{1,s_k^1}^1\right\} \cup \bigcup_{\substack{l \in L_I \\ l \rhd_I i_k}} \left\{T_{1,s_k^1}^1 \rightarrow {}^lY_{2,s_k^1}^1 \, {}^lZ_{2,s_k^1}^1\right\}$$

In this case, we have created a three-state specification with each state corresponding to the "current" integrity label associated with the object.

6.5.3 Low-Water Mark Policy on Subjects

Both the BLP confidentiality property and the Biba strict integrity property assume that subjects' and objects' assigned labels are static. Satisfaction of the respective property depends only on relative labelling comparisons made at the time of access.

The low-water mark integrity policy assumes that a subject's integrity level can change over time in a way that depends on the integrity level of the object it accesses. For a particular $s \in S$, the value returned by $il(s)$ is non-increasing with respect to time. The value is the low-water mark determined from the previous behaviour of the subject s. The low-water mark is the greatest lower bound of the integrity level of all objects accessed for observation by subject s. The subject is constrained to modify only those objects that possess an integrity level that dominates the subject's integrity level.

This behaviour can be expressed by the following three axioms:

$$\forall s \in S, o \in O, \quad s \text{ observes } o \Rightarrow il'(s) = \text{glb}(il(s), il(o))$$

$$\forall s \in S, o \in O, \quad s \text{ can alter } o \Rightarrow il(s) \rhd_I il(o)$$

$$\forall s_1, s_2 \in S, \quad s_1 \text{ can invoke } s_2 \Rightarrow il(s_1) \rhd_I il(s_2)$$

In practice, this policy has a rather peculiar behaviour, since the set of objects modifiable by a given subject can change with each observation. A subject can sabotage its own processing by making certain objects inaccessible for modification. This is a side effect of the policy and, short of reinitializing the subject, nothing can be done about it.

Under the event-based formalism, we can easily represent the second axiom with an appropriately defined object interface structure. The first axiom is more difficult to contend with, since it requires memory of previous accesses to be resident with each subject. To further complicate this, a subject's integrity level changes depending on which objects it accesses.

In general, a "subject" operates as an agent for some real user. The integrity level of the subject should reflect that of the user. Therefore, it makes little sense to change the integrity level of a subject without changing the levels of all other agents of that user. This has the potential for creating inconsistencies, especially where particular operations are only partially completed at the time of the change. Also, real user integrity levels are assigned using a wide variety of criteria that are not part of the system itself. Reassigning such levels by the system makes little sense.

6.6 DIGITAL LOGIC

The example presented in this section is intended to expand the potential applicability of our modelling approach beyond computer security. We do not use modular transaction properties in this particular example, although labelled events and the associated grammatical representation are used.

If we define $L = \{0,1\}$, we can consider a case where each event represents a transition in logical level, and the label associated with the event indicates the polarity of the transition[70].

---

[70]A further extension would support multiple-valued logic where $|L|$ was the number of allowed levels.

As a simple example, let us consider the simple logical inverter that appears in Figure 6.5:

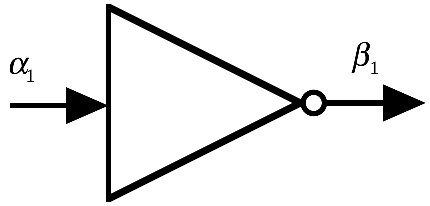

Figure 6.5: Logical Inverter

A sample timing diagram for the inverter is shown in Figure 6.6.

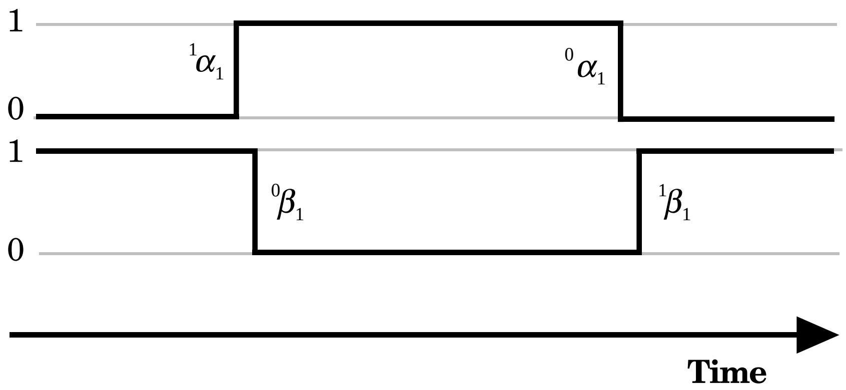

Figure 6.6: Logical Inverter Timing Diagram

It should be evident that this system can be represented easily with the two state structure shown in Figure 6.7.

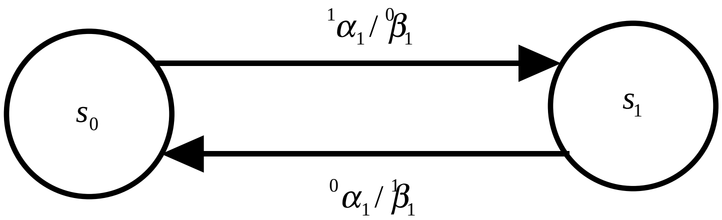

Figure 6.7: Logical Inverter State Diagram

The specification that is required is

$$m = 1 \;\; n = 1 \;\; b = 1 \;\; S = \{s_0, s_1\}$$

$$N = \{\langle s_0, s_1 \rangle, \langle s_1, s_0 \rangle\}$$

$$prereq_{s_0} = \{\langle {}^0\alpha_1, {}^1\beta_1 \rangle\} \;\; r_{s_0} = [1,1] \;\; BPT_{s_0} = \{T_{1,s_0} \rightarrow {}^0Y_{1,s_0} \, {}^1Z_{1,s_0}\}$$

$$prereq_{s_1} = \{\langle {}^1\alpha_1, {}^0\beta_1 \rangle\} \;\; r_{s_1} = [1,1] \;\; BPT_{s_1} = \{T_{1,s_1} \rightarrow {}^1Y_{1,s_1} \, {}^0Z_{1,s_1}\}$$

The specification puts the inverter into an initial state in which it is outputting a "high" logical level, while awaiting a low-to-high transition on its input.

As another example, we can consider the logical AND module[71] that appears in Figure 6.8.

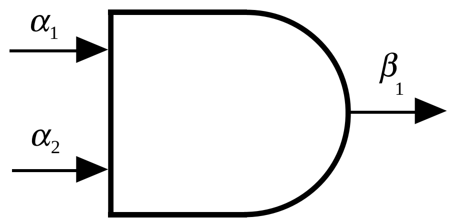

Figure 6.8: Logical AND

A sample timing diagram for this module is shown in Figure 6.9.

[71]This is conceptually very different from the event AND Module that would be specified by a 2,1-GM.

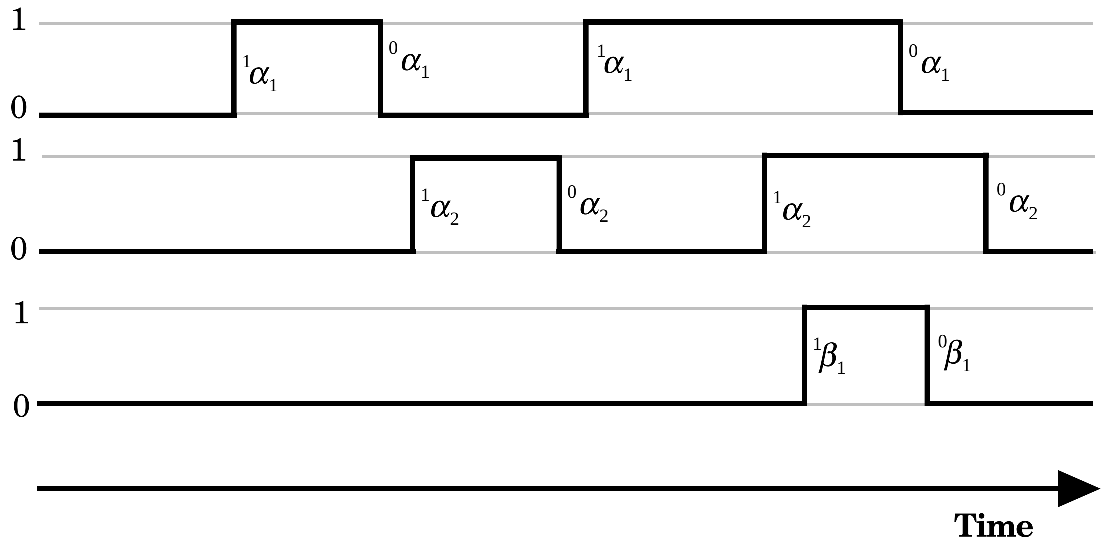

Figure 6.9: Logical AND Timing Diagram

The state specification, which appears in Figure 6.10, is quite complicated.

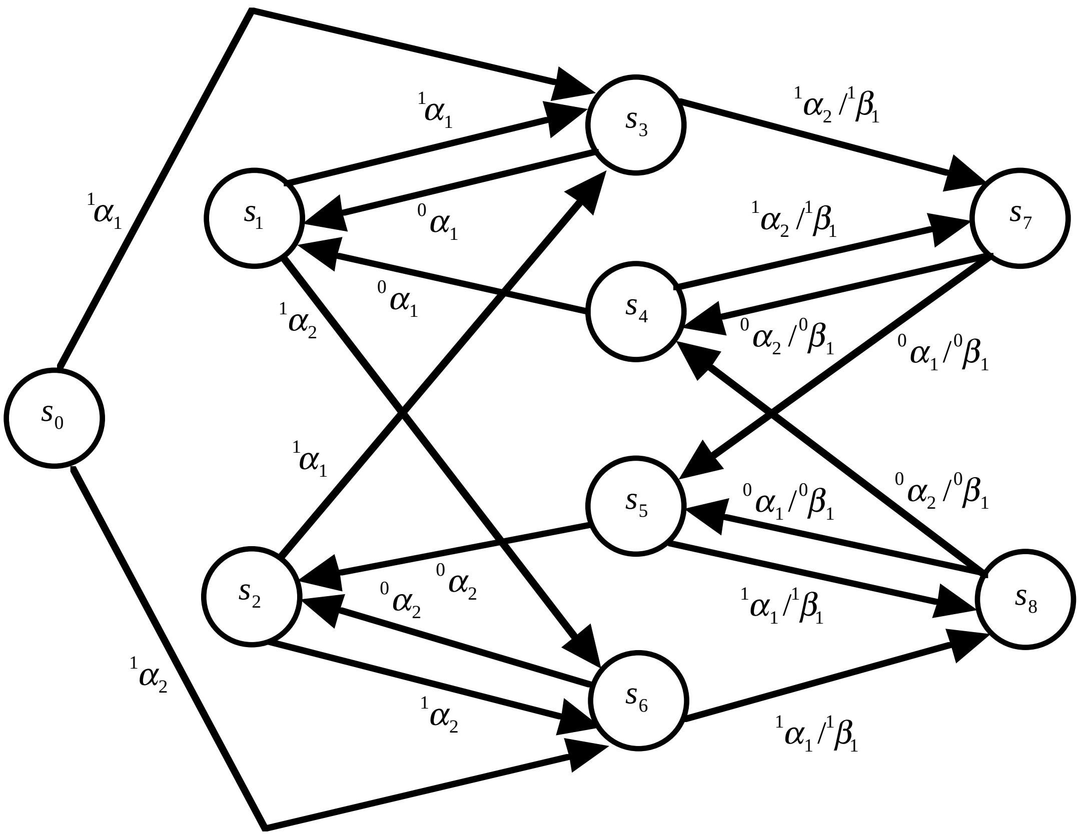

Figure 6.10: Logical AND State Diagram

A state has been introduced to represent each of the eight possible relevant histories for each of the three I/O events. In addition, we have introduced a special initial state s_0 to simplify the specification. The specification of the module is as follows:

$$m = 2 \;\; n = 1 \;\; b = 1 \;\; S = \left\{ s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 \right\}$$

$$N = \left\{ \begin{array}{c} \langle s_0, s_3 \rangle, \langle s_0, s_6 \rangle \\ \langle s_1, s_3 \rangle, \langle s_1, s_6 \rangle, \langle s_2, s_3 \rangle, \langle s_2, s_6 \rangle \\ \langle s_3, s_1 \rangle, \langle s_3, s_7 \rangle, \langle s_4, s_1 \rangle, \langle s_4, s_7 \rangle \\ \langle s_5, s_2 \rangle, \langle s_5, s_8 \rangle, \langle s_6, s_2 \rangle, \langle s_6, s_8 \rangle, \\ \langle s_7, s_4 \rangle, \langle s_7, s_5 \rangle, \langle s_8, s_5 \rangle, \langle s_8, s_4 \rangle \end{array} \right\}$$

$$prereq_{s_0} = \varnothing \;\; r_{s_0} = [1,1] \;\; BPT_{s_0} = \left\{ T_{1,s_0} \rightarrow \varepsilon \right\}$$

$$prereq_{s_1} = \varnothing \;\; r_{s_1} = [1,1] \;\; BPT_{s_1} = \left\{ T_{1,s_1} \rightarrow {}^0Y_{1,s_1} \right\}$$

$$prereq_{s_2} = \varnothing \;\; r_{s_2} = [1,1] \;\; BPT_{s_2} = \left\{ T_{1,s_2} \rightarrow {}^0Y_{2,s_2} \right\}$$

$$prereq_{s_3} = \varnothing \;\; r_{s_3} = [1,1] \;\; BPT_{s_3} = \left\{ T_{1,s_3} \rightarrow {}^1Y_{1,s_3} \right\}$$

$$prereq_{s_4} = \left\{ \langle {}^0\alpha_2, {}^0\beta_1 \rangle \right\} \;\; r_{s_4} = [1,1] \;\; BPT_{s_4} = \left\{ T_{1,s_4} \rightarrow {}^0Y_{2,s_4} \, {}^0Z_{1,s_4} \right\}$$

$$prereq_{s_5} = \left\{ \langle {}^0\alpha_1, {}^0\beta_1 \rangle \right\} \;\; r_{s_5} = [1,1] \;\; BPT_{s_5} = \left\{ T_{1,s_5} \rightarrow {}^0Y_{1,s_5} \, {}^0Z_{1,s_5} \right\}$$

$$prereq_{s_6} = \varnothing \;\; r_{s_6} = [1,1] \;\; BPT_{s_6} = \left\{ T_{1,s_6} \rightarrow {}^1Y_{2,s_6} \right\}$$

$$prereq_{s_7} = \left\{ \langle {}^1\alpha_2, {}^1\beta_1 \rangle \right\} \;\; r_{s_7} = [1,1] \;\; BPT_{s_7} = \left\{ T_{1,s_7} \rightarrow {}^1Y_{2,s_7} \, {}^1Z_{1,s_7} \right\}$$

$$prereq_{s_8} = \left\{ \langle {}^1\alpha_1, {}^1\beta_1 \rangle \right\} \;\; r_{s_8} = [1,1] \;\; BPT_{s_8} = \left\{ T_{1,s_8} \rightarrow {}^1Y_{1,s_8} \, {}^1Z_{1,s_8} \right\}$$

Given that we can represent an inverter circuit and an AND circuit, it is possible to construct a representation for any digital logic circuit. Such representations might be useful in studying the behaviour of complicated circuits. Extra labelling information could be added to help deal with other constraints. For example, a modular transaction property based on the fact that all of the output events from a logical gate must occur at a later time than all of the input events could allow circuit timing considerations to be modelled.

6.7 NEEDHAM-SCHROEDER PROTOCOL

This section will apply the model to a key distribution protocol introduced by Needham and Schroeder[72] and later modified by Denning and Sacco[73]. The example is presented to identify the potential applicability of the modelling approach to authentication protocols between communicating users or processes.

The steps in the basic protocol in which user A acquires a key K from authentication server S to share with another user B are as follows:[74]

1. A sends to S the plaintext message (A, B) stating both A's and B's identity.

2. S sends to A the ciphertext message $E_A(B, K, T, C)$, where E_A is the enciphering transformation derived from A's private key, K is the session key, T is the current date and time, and $C = E_B(A, K, T)$, where E_B is the enciphering transformation derived from B's private key.

3. A sends to B the message C from Step 2.

The first step in modelling this protocol is to define a set of event symbols corresponding to the activities described the three steps above. Since it is at the centre of the system, we will concentrate on modelling the actions of user A, but it is of course possible to model user B and the authentication server S.

---

[72]R.M. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993-999, December 1978.
[73]D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):553–536, August 1981.
[74]For simplicity, we have omitted the two handshake steps that would occur following Step 3.

Event	**Event Symbol**
A sends to S the plaintext message (A,B)	${}^{(A,B)}\beta_1$
A receives the ciphertext message $E_A(B,K,T,C)$	${}^{E_A(B,K,T,C)}\alpha_1$
A sends to B the message C	${}^{C}\beta_2$
A sends to B a message using the session key K	${}^{K(message)}\beta_2$
A receives a message from B encrypted using the session key K	${}^{K(message)}\alpha_2$

The behaviour of A is then easily represented with the following specification:

$$m=2 \;\; n=2 \;\; b=1 \;\; S=\{s_0,s_1,s_2,s_3\} \;\; N=\{\langle s_0,s_1\rangle,\langle s_1,s_2\rangle,\langle s_2,s_3\rangle\}$$

$$prereq_{s_0}=\varnothing \;\; r_{s_0}=[1,1] \;\; BPT_{s_0}=\{T_{1,s_0}\to{}^{(A,B)}Z_{1,s_0}\}$$

$$prereq_{s_1}=\varnothing \;\; r_{s_1}=[1,1] \;\; BPT_{s_1}=\{T_{1,s_1}\to{}^{E_A(B,K,T,C)}Y_{1,s_1}\}$$

$$prereq_{s_2}=\varnothing \;\; r_{s_2}=[1,1] \;\; BPT_{s_2}=\{T_{1,s_2}\to{}^{C}Z_{2,s_2}\}$$

$$prereq_{s_3}=\varnothing \;\; r_{s_3}=[0,\infty] \;\; BPT_{s_3}=\begin{Bmatrix} T_{1,s_3}\to{}^{K(message)}Y_{2,s_3} \\ T_{1,s_3}\to{}^{K(message)}Z_{2,s_3} \end{Bmatrix}$$

In this situation we do not require delay or shuffle productions in any of the states.

The system remains in state s_3 for repeated exchanges of messages. A further extension to the protocol and specification would allow the session to be terminated.

With the model of a single user's behaviour in hand, it becomes possible to consider a case where two or more users are participating in an authentication protocol. In this case, we could represent each of the users by separate modules and compose them with appropriate interconnections to represent the overall system.

6.8 SUMMARY

This chapter began by describing a new confidentiality property called prerequisite confidentiality. It was shown that this property fell into the class of modular transaction properties introduced in Chapter 5.

Next, we introduced the standard modelling notion of subjects and objects. The applicability of this notion within the language-based modular representation was then described. Standard definitions of both confidentiality and integrity were then instantiated using the model and several examples provided. Example applications in the areas of digital circuits and key distribution protocols were then presented.

7 SUMMARY AND CONCLUSIONS

7.1 SUMMARY

We began by introducing a number of basic ideas relating to the modelling of systems, composability, and nondeterminism. The claim was made that the event trace representation was a good one, especially because of its flexibility with respect to timing. Only the relative ordering of events was important. The difficulty with event traces was that it becomes quite easy to lose the functional structure resulting from causality. The fact that one event causes another is easily lost when several hundred other events are interposed between the two. Thus, any attempt to use event traces as a basis for system modelling must include a finite representation that fully captures the temporal and causal relationships between events.

The thesis was then put forward. A grammatical approach can be used to maintain the overall structure of event sequences, as well as allowing the causality between input and output events to be easily represented. In addition, the notion of composability—that an overall operation is made up of a sequence of sub-operations—can be properly supported by the grammatical representation. With such a representation, a number of desirable properties can be represented and their composability properties investigated and proven.

As a result of this discussion, it was claimed that any modelling methodology that is likely to be effective must have the following characteristics:

- Representations of real systems must be tractable, usually through some finite representation.

- The model must support composability in a useful way.
- The model must be able to represent relevant system behaviour while ignoring irrelevant activities.
- The model must maintain the causal and temporal relationships between events.
- It must be possible to represent a wide variety of properties under the model and to determine which properties are maintained under composition.
- The model must be able to represent nondeterministic behaviour.

The next chapter described a sampling of related research in the area. This included a discussion of confidentiality and integrity modelling as well as some examples of more general system specification techniques. Each of the models has varying success at achieving the goals in the areas of composability and nondeterminism. However, none adequately deal with the issues of causality and none develop a representation that is tractable when real systems are modelled.

We then went on to describe the language and grammar-based representation. First, the specification structure was introduced for single, stand-alone elements. Next, the composition of several specifications was described. Finally, labelling of events was introduced and a notion of properties based upon these labels was described. The composability properties of a broad class of input/output properties were proven in Section 5.4.

With the modelling framework in hand, we next applied the methodology to a number of areas, including confidentiality, integrity, digital circuits, and protocol specification.

7.2 CONCLUSIONS

The language-based approach of this thesis succeeds in achieving all the objectives listed above. In this sense, it is better than any of the previous modelling approaches, as described in Chapter 2.

Because of the use of formal languages and grammars to represent the structure of the event sequences, we have the desired finite representation. In addition, because it is possible to construct a parser for such languages, we have a real method for identifying those event sequences that properly fall into the set of those that are valid for the specified system.

It was shown in Section 4.2.2 that individual specifications could be interconnected to form composite systems. The resulting composite specifications were themselves members of the class of specifications developed in Section 3.2. Thus, we have a clearly defined hierarchical method for developing system representations in a bottom-up manner. In Sections 5.3 and 5.4, it was demonstrated that a large class of properties could be represented under the model and that all properties in that class were maintained under the composition operation.

The causality between events is clearly maintained and enforced though the prerequisite relation. This affects the nature of the shuffle productions that are permitted in a specification so that causally related events always have the desired temporal ordering.

Finally, nondeterminism is represented by having all possibilities allowed by the grammatical specification. Previous approaches might have found the generation of such an exhaustive specification to be arduous, but the grammatical representation allows nondeterminism to be represented very compactly.

Many of the problems have been solved or reduced by this specification approach, but at the expense of requiring the modeller to be more cunning and skilful. He or she must identify the events pertinent to the circumstances being modelled, and not complicate the model with events that are not relevant to the properties being represented. We argue that this is a good thing: the modeller is forced to concentrate on the events that affect the model and side issues of no significance do not feature.

7.3 DIRECTIONS FOR FUTURE WORK

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

- Developing a software-based tool for describing and analyzing systems based upon the grammatical representational approach of this thesis. It should be obvious to the reader that although all the modules and compositions are mathematically tractable, the notation required to represent such systems can become overwhelming. The development of an automated tool would allow the description and analysis of much larger and more complicated systems.

- Further work on emergent properties. As has been noted several times throughout this thesis, we have been concerned with deciding whether properties are preserved by composition. However, there is a larger class of situations in which new and different properties emerge in a composite when two systems are interconnected. A further investigation of emergent properties and an extension to the model of this thesis would be appropriate.

- Our work has investigated the composition of modules that each enforce a security policy. We have concentrated on the case in which all modules enforce the same policy. This is not always the case of interest. It would be interesting to examine the composition of security policies themselves. It is not necessarily the case that every security policy can coexist with every other policy. Some policies might allow coexistence, but their coexistence could be inconsistent with composability. The examination of such heterogeneous policy situations would tie in directly with a study of emergent properties, since the two concepts are interrelated.

- A look at the range of possible modular transaction properties—are transitive relations between inputs and outputs the *largest* class that are composable? In connection with the investigation of emergent properties it would be interesting to look into characterizing the maximal class of event-label properties that are preserved under the composition operation.

- Further work on representing the activities of the environment. Under our representation, events were generated by and sent to the external environment. When properties were introduced, the type of events that were accepted or generated by the system was restricted. This is an *open system* approach because the environment surrounding the system of interest is left undefined. For example, we do not know the mechanism by which the external environment accepts and generates events. It would be useful to investigate whether the model could be extended to a *closed system* approach in which a formal description of the environment was included as a formally modelled entity.

- Section 4.4 briefly mentioned the issue of deadlock. We conjectured that our composition technique eliminated the possibility of deadlock in a composite system. It would be useful to investigate the potential for deadlock further in the context of the event-based modular representation developed in this thesis. This would likely include the development of necessary and sufficient conditions for deadlock-free specifications.

- In this thesis, we have been primarily concerned with bottom-up, hierarchical development. To this end, we have developed a mechanism for building up the specification of a complicated system from its subcomponents. Another aspect of hierarchical design and specification involves top-down analysis—taking a high-level specification and breaking it down into simpler subcomponents. It would be interesting to study how a specification based upon formal grammars and languages might be applied to this problem of decomposition.

[This page should be blank.]

Bibliography

Abadi, Martín and Leslie Lamport. The Existence of Refinement Mappings. Technical Report 29, Digital Systems Research Center, Palo Alto, CA, August 1988.

Abadi, Martín and Leslie Lamport. Composing Specifications. Technical Report 66, Digital Systems Research Center, Palo Alto, CA, October 1990.

Alpern, Bowen and Fred B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):114–133, October 1985.

Anderson, J.P. *Computer Security Technology Planning Study*, ESD–TR–73–51, Vol. I, AD–758 206, ESD/AFSC, Hanscom AFB, Bedford, MA, October 1972.

Bell, D. Elliott and Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report TR–2547, MITRE Corporation, Bedford, MA, March 1973.

Biba, K.J. Integrity Considerations for Secure Computer Systems. Technical Report TR–3153, MITRE Corporation, Bedford, MA, April 1977.

Canadian System Security Centre. *The Canadian Trusted Computer Product Evaluation Criteria*, Version 3.0e, January 1993.

Clark, D.D. and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1987, pp. 184–194.

Clark, D.D. and D.R. Wilson. Evolution of a Model for Computer Integrity. *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, October 1988.

Denning, D.E. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–242, May 1976.

Denning, D.E. and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):553–536, August 1981.

Fischer, Charles N. and Richard J. LeBlanc, Jr. *Crafting a Compiler*. The Benjamin/Cummins Publishing Company, Menlo Park, CA, 1988.

Gibbons, Alan. *Algorithmic Graph Theory*. Cambridge University Press, 1989.

Goguen, Joseph A., and José Meseguer. Security Policies and Security Models. *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, April 1982, pp. 11–20.

Goguen, Joseph A., and José Meseguer. Unwinding and Inference Control. *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, April 1984, pp. 75–86.

Hinton, Heather M. An Environment-Based Approach to the Composition of Safe Systems. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, January 1993.

Hopcroft, John E. and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Series in Computer Science and Information Processing, Addison-Wesley, 1969.

Hopcroft, John E. and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science, Addison-Wesley, 1979.

Johnson, Dale M., and F. Javier Thayer. Security and the Composition of Machines. *Proceedings of the Security Foundations Workshop*, Franconia, NH, June 1988, pp. 72–89.

Lee, E.S., P.I.P. Boulton, D.M. Lewis, M. Stumm, and B.W. Thomson. A Trusted Network Architecture. Technical Report 228, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, 1988.

Lee, E.S., P.I.P. Boulton, B.W. Thomson, and R.E. Soper. Composable Trusted Systems. Technical Report 272, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, May 1992, Final Report of Contract W2213–9–6351/01–SZ.

Lin, Ping. The Composability of Behaviorally Secure Systems. Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ontario, 1991.

McCullough, Daryl. Specifications for Multi-Level Security and a Hook-Up Property. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987, pp. 161–166.

McCullough, Daryl. Noninterference and the Composability of Security Properties. *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988, pp. 177–186.

Murata, Tadao. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

National Computer Security Center. *Department of Defense Trusted Computer Security Evaluation Criteria*, DOD 5200.28–STD, December 1985.

Needham, R.M. and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM,* 21(12):993–999, December 1978.

Nestor, John P. Security Modelling Using Hierarchical State Machines. Master's Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ontario, December 1990.

Nestor, John P. Security Modelling Using Hierarchical State Machines. *Proceedings of the 7th Annual Computer Security Applications Conference*, San Antonio, TX, December 1991, pp. 110–119.

Parker, Donn B. Restating the Foundation of Information Security. *Proceedings of the Second Annual North American System Security Symposium*, Toronto, Ontario, October 1991.

Peterson, James L. and Abraham Silberschatz. *Operating System Concepts*. Second Edition. Addison-Wesley Publishing Company, 1985, pp. 271–306.

Ramadge, Peter J.G. and W. Murray Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

Révész, György E. *Introduction to Formal Languages*. Dover Publications, Inc., New York, 1991.

Stoll, Robert R. *Set Theory and Logic*. Dover Publications, Inc., New York, 1979.

Sutherland, David. A Model of Information. *Proceedings of the 9th National Computer Security Conference,* 1986, pp. 175–183.

Turnbull, Christopher J.M. Deterministic Left to Right Parsing. Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, 1974.

Turnbull, Christopher J.M. Generalized Deterministic Left to Right Parsing. *Acta Informatica 12*, 1979, pp. 187–207.

[This page should be blank.]

A GLOSSARY

A wide variety of terminology has appeared throughout this thesis. The most significant terms are collected here for convenient reference.

**-security property*	Allows alteration of an object by a subject only if the confidentiality level of the object dominates the confidentiality level of the subject.
alteration	An action by a subject than involves changing an object.
antisymmetric	A relation ρ is antisymmetric iff whenever $x\rho y$ and $y\rho x$ then $x = y$.
associative	An operation $*$ is associative iff $A*(B*C)=(A*B)*C$.
assurance	Our level of confidence that the implemented system conforms to the guaranteed specification.
authenticity	A safety property that requires every entity in a system to be authentically identified. No entity can masquerade as a different entity.
availability	The property that a system's services are accessible when needed and without undue delay.
basic production set	The basic production set generates a language in which input event(s) are immediately followed by their corresponding output event(s). The ordering of causally unrelated events is arbitrarily specified.
block	A substructure of a module that is causally independent from all other such substructures in the module.

causally related	Two events are causally related if there is a functional relationship that implies that the occurrence of one event (eventually) results in the occurrence of the other event.
complete transaction	The occurrence of a complete set of input events and the corresponding, causally related output events at a particular module.
composability	A situation in which a property fulfilled by an overall system is equivalent to properties of the individual components that make up the composite system.
confidentiality	The property that information is not made available or disclosed to an unauthorized user, process or object.
connectivity	A safety property that requires that a system is interconnected in a known way. Static connections cannot be altered, and dynamic connections must be under the control of the faithfulness policy.
delay production set	The productions in this set allow output events to be "delayed" past more recently occurring input events. In doing so, the module allows for buffering of input events.
discretionary access control (DAC)	Controls non-administratively controlled. Under discretionary access control, authorization and delegation do not require administrative intervention.
emergent property	A property of a composite system that is different from the properties held by each of the components.
environment	In open system modelling, the entity that is outside the system being modelled.
event	A noteworthy happening in the context of the system model.
event trace	An ordered sequence of events.

faithfulness	A safety property that requires that every entity in the system have a predictable response to a defined sequence of stimuli.
function	A relation such that no two distinct members have the same first coordinate.
guarantee	The properties of the system specification.
integrity	A safety property that reflects the quality of data in a computer system.
invocation	Execution of a process by another process.
label	A tag attached to an event, user, or process, that identifies certain model-relevant characteristics.
mandatory access control (MAC)	Administratively controlled. Under mandatory access control, authorization and delegation require administrative intervention.
mediator module	A module interposed between a subject and an object that controls access to the object by the subject according to some policy.
model	A description or analogy used to help visualize something that cannot be or is difficult to observe directly.
modular prerequisite relation	A relation on the set of events that defines those events that are prerequisites of other events in the system being modelled.
modular transaction property	A property that is expressed as a transitive relation between the labels associated with group of input events and the labels associated with group of output events that make up a particular complete transaction.
module grammar	A formal grammar that specifies the behaviour of a module. Sentences in the language generated from a module grammar are valid event sequences formed from a properly ordered succession of input and output events for the specified module.
object	A passive entity that is maintained by the system.

observation	A read-only operation in which information is transferred from one entity (the observee) to another entity (the observer).
partial ordering	A relation ρ in a set X is a partial ordering iff ρ is reflexive (in X), antisymmetric, and transitive.
partial transaction	A transaction is one in which output events are pending, or an incomplete set of inputs has been provided to a module.
property	A (desired) quality (of a system) that is relevant to the modelling exercise.
reflexive	A relation ρ in a set X is reflexive iff $x\rho x$ for each x in X.
simple security property	Allows observation of an object by a subject only if the confidentiality level of the subject dominates the confidentiality level of the object.
spontaneous event	An event that, as far as the modelled system specification is concerned, has not been specifically caused by the occurrence of another modelled event.
subject	A person, or agent acting on his or her behalf, such as a process. Subjects are active entities that perform actions to accomplish various tasks.
symmetric	A relation ρ in a set X is symmetric iff $x\rho y$ implies $y\rho x$ for each x and y in X.
system	A special subject that controls when and how other subjects access the objects that are available.
transitive	A relation ρ in a set X is transitive iff $x\rho y$ and $y\rho z$ imply $x\rho z$ for each x, y and z in X.
utility	A safety property that requires that information be in a form that is appropriate for the desired purpose.

B Annotated Algorithms

A number of complicated mathematical algorithms have been presented in this thesis. This appendix provides those algorithms with extra information regarding their operation.

Algorithm 4.1a: Unification Procedure

procedure $\mathit{Unify}(z, y)$

We must first look for those productions that contain either the output symbol, the input symbol, or both corresponding to either endpoint of the connection of z to y.

$\mathit{outProds}, \mathit{inProds}, \mathit{bothProds} := \varnothing$

$\forall x \in \mathit{BPT}^{C}_{\langle s_1, s_2 \rangle}$,

If we find such a production, then we store it in the appropriate set for later processing. The substring function returns true if the first argument is a substring of the second argument.

if $\mathit{substring}(z, \mathit{rhs}(x)) \wedge \mathit{substring}(y, \mathit{rhs}(x))$ **then**

$\mathit{bothProds} := \mathit{bothProds} \cup \left\{ T^{C}_{1,\langle s_1, s_2 \rangle} \rightarrow \mathit{rhs}(x) \right\}$

elsif $\mathit{substring}(z, \mathit{rhs}(x))$ **then** $\mathit{outProds} := \mathit{outProds} \cup \left\{ T^{C}_{1,\langle s_1, s_2 \rangle} \rightarrow \mathit{rhs}(x) \right\}$

elsif $\mathit{substring}(y, \mathit{rhs}(x))$ **then** $\mathit{inProds} := \mathit{inProds} \cup \left\{ T^{C}_{1,\langle s_1, s_2 \rangle} \rightarrow \mathit{rhs}(x) \right\}$

If either end of the connection has no associated production, then we have discovered an incompatibility. Record this state in the list of bad states and exit the unification procedure.

if $\mathit{bothProds} = \varnothing \wedge \left((\mathit{outProds} \neq \varnothing \wedge \mathit{inProds} = \varnothing) \vee (\mathit{outProds} = \varnothing \wedge \mathit{inProds} \neq \varnothing) \right)$

then $\mathit{badstate} := \mathit{badstate} \cup \langle s_1, s_2 \rangle$, **exit procedure**

Perform the pairwise concatenation of those productions containing the output non-terminal with those containing the input non-terminal. In each case, the non-terminals corresponding to the endpoints are deleted and any duplicate symbols removed in creating the new unified transaction production. The delstring function removes all occurrences of the first argument from the second argument.

$\forall p_{out} \in outProds, p_{in} \in inProds,$

$$BPT^{C}_{\langle s_1,s_2\rangle} := BPT^{C}_{\langle s_1,s_2\rangle} \cup \left\{ T^{C}_{1,\langle s_1,s_2\rangle} \rightarrow remdups\begin{pmatrix} delstring(z, rhs(p_{out})) \\ +delstring(y, rhs(p_{in})) \end{pmatrix} \right\} \setminus p_{in}$$

$\forall p_{out} \in outProds, p_{both} \in bothProds,$

$$BPT^{C}_{\langle s_1,s_2\rangle} := BPT^{C}_{\langle s_1,s_2\rangle} \cup \left\{ T^{C}_{1,\langle s_1,s_2\rangle} \rightarrow remdups\begin{pmatrix} delstring(z, rhs(p_{out})) \\ +delstring(y, delstring(z, rhs(p_{both}))) \end{pmatrix} \right\}$$

$$BPT^{C}_{\langle s_1,s_2\rangle} = BPT^{C}_{\langle s_1,s_2\rangle} \setminus outProds$$

$\forall p_{both} \in bothProds,$

$$BPT^{C}_{\langle s_1,s_2\rangle} := BPT^{C}_{\langle s_1,s_2\rangle} \cup \left\{ T^{C}_{1,\langle s_1,s_2\rangle} \rightarrow remdups\left(delstring\left(y, delstring\left(rhs\left(p_{both}\right), z\right)\right)\right) \right\} \setminus p_{both}$$

end *Unify*

Algorithm 4.1b: Transaction Production Unification (Mainline Code)

$badstate := \varnothing$

For each possible composite state
$\forall \langle s_1, s_2 \rangle \in S^C$

$BPT^C_{\langle s_1, s_2 \rangle} := \varnothing$

Relabel the left-hand side of all of the transaction productions.
$\forall k \in [1,2], x \in BPT^k_{s_k}$,

$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \to rhs(x) \right\}$

Perform the unification for each possible connection.
$\forall k_1, k_2 \in [1,2], j \in [1, n^{k_1}], i \in [1, m^{k_2}]$,

if $connected(\beta_j^{k_1}, \alpha_i^{k_2})$ **then** $Unify(Z_j^{k_1}, Y_i^{k_2})$

Sort the right-hand sides of each of the new composite transaction productions.
$\forall x \in BPT^C_{\langle s_1, s_2 \rangle}$,

$BPT^C_{\langle s_1, s_2 \rangle} := BPT^C_{\langle s_1, s_2 \rangle} \cup \left\{ T^C_{1,\langle s_1, s_2 \rangle} \to sortseq(rhs(x)) \right\} \setminus x$

Algorithm 4.2: Composite Shuffle Production Set Construction

$SP_s^C := \varnothing$

First add the productions within each block. Scan through each single-symbol combination pair in the right-hand side of the production and check to see whether a shuffle production is permitted by the prerequisite relation.

$$\forall u \in BPT_s^C,\ v,x,z \in \left(\bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\} \right)^*,\ w,y \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\}$$

s.t. $rhs(u) = vwxyz,$

$$\textbf{if} \left(w \notin \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\} \vee y \notin \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \right) \wedge \langle w,y \rangle \notin prereq_s^C \ \textbf{then}$$

$$SP_S^C := SP_S^C \cup \{ wy \to yw \}$$

Now add the inter-block shuffle productions. Pair the single symbol combinations from the right-hand side of transaction productions from different blocks and create the required shuffle productions.

$\forall k_1 \in [1, b^C - 1], k_2 \in [k_1 + 1, b^C],\ p_1, p_2 \in BPT_s^C,$

$$u,w,x,z \in \left(\bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\} \right)^*, v,y \in \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \cup \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\},$$

s.t. $lhs(p_1) = T_{k_1,s}^C \wedge lhs(p_2) = T_{k_2,s}^C \wedge rhs(p_1) = uvw \wedge rhs(p_2) = xyz,$

$$\textbf{if} \left(v \notin \bigcup_{\substack{k\in[1,2]\\ j\in[1,n^k]}} \left\{ Z_{j,s}^k \right\} \vee y \notin \bigcup_{\substack{k\in[1,2]\\ i\in[1,m^k]}} \left\{ Y_{i,s}^k \right\} \right) \ \textbf{then}\ SP_s^C := SP_s^C \cup \{ vy \to yv \}$$