Abstract

This thesis presents a formal theory of information flow using category theory. The theory of information flow is used to specify confidentiality requirements and develop systems which satisfy the specifications. A calculus for combining formal statements of confidentiality is developed and a satisfaction relation defined between between systems and statements in the calculus. A number of laws are developed concerning satisfaction, with respect to conjunction, disjunction and refinement. A significant case study is used to illustrate the results presented in this thesis.

Contents

Chapter 1

Introduction

As computer systems become more powerful and cheaper they permeate more of our everyday life. Embedded computer systems are beginning to appear as smart cards storing personal financial and medical details which are accessible to authorized individuals. We rely also on large distributed computer systems for commercial banking and even the provision of goods in supermarkets. With this greater reliance on the automated processing and movement of information comes a duty to demonstrate that computer systems are secure.

The requirement for security in computer systems falls into three parts: **Confidentiality**, which is about controlling disclosure of information; **Integrity**, which is about ensuring that the information stored in a machine is accurate; and **Availability**, which is concerned with preventing the actions of one user from adversely affecting the work of another.

Availability is a concern whenever users share a limited resource. In the case of a computer system this will usually include processing time, memory usage and input/output bandwidth. A computer system must therefore ensure that one user cannot take an unfair share of a shared resource, which can deny service to another user, while at the same time make effective use of the computer's expensive resources.

Integrity is a major concern of many organizations such as banks where the loss of the data can itself represent a loss of billions of pounds. In this case a computer system must ensure that data or resources cannot be modified or used inappropriately. It is the characterization of what is appropriate, for a particular computer system, which is the difficult part of ensuring integrity.

Confidentiality is a concern when the unauthorized disclosure of information can lead to: infringement of personal privacy; failure to meet legal obligations; loss of commercial confidentiality; and even threats to the safety of individuals. One of the problems with controlling the information held in a computer system is that information is itself a nebulous entity. Although it is useful to think of files and records as being held by a computer system, they are actually realized as bits within a machine which can be duplicated and moved with great speed and ease. It is therefore insufficient to guard files, one may have to worry about the subtle relationships between different bits, or words held and moved around a machine.

The aim of this thesis is to present a formal theory of information flow which

can be used to specify confidentiality requirements and develop systems which satisfy these specifications. The meaning of confidentiality is based on information flow. Information flow is concerned with the inference of information by means of observations. The first rigorous formulation of information flow for security was given in [Goguen & Meseguer 82].

In the game of bridge the players start with a knowledge of what cards are in a pack and which ones are present in a hand; with every bid made in the game, a player can infer information about what cards are held by his partner and opponents. Distributed information systems can be thought of in a similar way: each terminal gives observations of the system from which it might be possible to infer what is going on at other terminals. Sometimes this will violate confidentiality and at other times not: this thesis will give ways of specifying which it is.

Category theory has been used in this thesis to formulate the basic concept of information flow. Category theory provides an abstract viewpoint of information flow and this gives an independence from particular models of computation such as state machines. There are various formulations of information flow, [Foley 88, Goguen & Meseguer 82, Goguen & Meseguer 84, Jacob 88a, Sutherland 86], each of these is in a particular model of computation. By not considering the internal structure of a system or how it communicates with its environment only properties which are independent of their representation emerge. This leads to a theory of confidentiality which can be realized in many models of computation.

The thesis begins with a review of the notation and theory for Communicating Sequential Processes, CSP; Timed Communicating Sequential Processes, TCSP; and category theory. In chapter three the basic postulates for an information flow category are stated and discussed. From the postulates a categorical infrastructure is constructed to give a semantics for information flow. In chapter four the semantic model of chapter three is used to give a semantics for a calculus of confidentiality specifications. In chapter five it is shown what it means for a system to: satisfy a confidentiality specification; how satisfaction respects conjunction and disjunction in the calculus of chapter four; and finally investigates the use of a Galois correspondence. Chapter six continues with the construction of categorical infrastructure to support two general theorems. The first general theorem allows a designer to specify a static confidentiality property at an interface which can be lifted to a dynamic confidentiality specification. The second general theorem allows a verifier to check a system's states and interactions with an interface against a confidentiality property of an interface and if successful know that the system satisfies the lifted property. The results of chapter five can then be used to combine systems and confidentiality specifications. The two general theorems are specialized to the concrete category of chapter four. In [Jacob 89] it was shown that functional refinement of a system can conflict with its confidentiality specification. In the second part of chapter six how a system can be functionally refined so that it preserves its confidentiality specification is investigated. A number of useful laws are stated and proved.

In chapter seven the categorical approach taken in this thesis is shown to be general by constructing two more concrete categories. The first category is constructed

from a model of Petri nets. The second category is constructed from the timed trace model of TCSP and allows information flow across real time systems to be reasoned about.

Chapter eight presents a sizeable case study of the security requirement of a Command and Control Information System, CCIS. The requirements for the CCIS are realistic and the security requirements are formalized using the theory developed in this thesis. The security requirement includes two statements involving integrity; the first is formalized as a functional specification and the second as an information flow specification. A design for the CCIS addressing the functional requirements is produced using CSP. Each CSP process is compared against the relevant formalized security policy statement and shown to satisfy it. Using results from the thesis the parallel composition of the individual systems is shown to satisfy the overall security policy.

In the final chapter of this thesis, the results of the research presented in the preceding chapters is discussed. Alternative approaches to the specification and development of secure systems are considered and outline directions for future work given. The thesis ends with an appendix which completes the case study presented in chapter eight. A glossary of symbols is provided at the back of the thesis.

Chapter 2

CSP and Abstract nonsense

This chapter is a brief review of CSP, TCSP and category theory. There are three sections covering: CSP and its trace semantics; TCSP and its timed trace semantics; and category theory.

2.1 The trace model of CSP

CSP is a theory and notation for describing properties of processes. A process is a mechanism which performs events in some prescribed manner and interacts synchronously with its environment. The notation allows a process to be described as a concurrent combination of synchronously interacting processes as well as the usual programming constructs such as sequential composition. The set of events a process, P, may engage in is denoted by αP, the alphabet of the process.

In this thesis the information flow properties of systems, rather than properties such as divergence or deadlock, are only considered. The trace, failures and timed semantics of CSP all contain semantic information which allow observers of a system to infer information about the behaviour of users. The divergences, or stability semantics, of CSP only allow one bit of information to be inferred, that a user has engaged in a divergent behaviour. The divergences and failures models are ignored for simplicity and only consider the trace model of CSP is used.

Abstract Syntax of CSP

The language construct $\perp$ representing the diverging process which engages in no event visible to the environment has been omitted from the abstract syntax of CSP. The letters P, Q, R range over syntactic processes; the letters a, b range over the alphabet αP of a process; and X, Y over subsets of αP. The alphabet of P is assumed to be equal to the alphabet of Q except for the construct $P \| Q$. F denotes an "appropriate" composition of the syntactic operators.

CSP Abstract syntax

$$\begin{aligned} P \quad ::= \quad & STOP_A \mid SKIP \mid (a \rightarrow P) \mid P [\![Q \mid P \sqcap Q \mid P \parallel Q \mid P ; Q \mid \\ & P \backslash X \mid \mu P.F(P) \end{aligned}$$

The atoms $STOP_A$ and $SKIP_A$ denote respectively the deadlocked process, which engages in events from A, and the process which terminates successfully, which engages in events from A. The process $(a \rightarrow P)$ denotes the process which engages in the event a and then behaves like the process P. The process $P[]Q$ is the process which will behave like P or behave like Q depending on what the environment decides, this is known as environmental choice and reflects external non-determinism. The process $P \sqcap Q$ is the process which will behave like P or behave like Q depending on some internal decision, this is known as internal choice and reflects internal non-determinism. The process $P \parallel Q$ denotes the parallel composition of two processes and $P;Q$ denotes the sequential composition of two processes. The process $P \backslash X$ denotes hiding (or abstracting from) the events in X in the process P. Finally the process $\mu P.F(P)$ denotes recursion provided the recursion is guarded.

Semantic models of CSP

A CSP process communicates with its environment via its alphabet of atomic communications or "events". To abstract from implementation concerns, events require the co-operation of all participants and are instantaneous. The requirement for instantaneous communication appears to be unrealistic at first sight. The problem of instantaneous communication is that of modelling communication. Two machines connected by a wire cannot communicate instantaneously but two processes do not necessarily correspond to the two machines. The boundaries of when one process ends and another begins is a point on the wire connecting the two machines. The point on the wire can be in the middle, at either machine or even inside a machine. The time at which a signal leaves one process and enters another is thus instantaneous. If the wire is not to be considered part of either process then it must be modelled explicitly as a mediating process. In timed CSP events have zero time duration, a delta time delay after each event prevents infinitely many events from occurring in a finite time; for a more detailed discussion see [Davies 91].

If a process is thought of as a black box then, to an observer, different mechanisms within the box are indistinguishable if their behaviour is the same. That is "if it walks like a duck and quacks like a duck then it is a duck", at least to an undiscerning observer. A single behaviour of a process up to some point in its history is a finite sequence of observed events. The set of all possible behaviours is called the *traces* of the process.

CSP has been significant as a formal tool for understanding the foundations of concurrency and for understanding how to implement concurrent systems. It has evolved from the trace based model ([Hoare 80]) to provide an adequate treatment of non-determinism, divergence ([Hoare et al 81], [Roscoe 82], [Brookes 83], [Olderog & Hoare 86], [Brookes et al], [Brookes & Roscoe], [Hoare 85]) and time ([Reed & Roscoe 86], [Reed & Roscoe 88], [Reed 88], [Schneider 89], [Davies 91]).

There are various related models of CSP, [Reed 88], which give a hierarchy of meanings to a CSP process. Only one untimed model will be used in this thesis — the trace model [Hoare 80, Roscoe 82, Hoare 85].

The trace model relates each process to a pair $(\alpha P, P)$. The first member of the pair is the alphabet of the process. The alphabet consists of the set of events which the process may engage in synchronously with its environment. The second member of the pair is a prefix closed non-empty subset of $(\alpha P)^*$, known as the traces of the process. The traces represent any sequence of events the process is prepared to engage in up to some unspecified time.

Any prefix closed non-empty subset of finite strings over an alphabet is a process in the trace model. Each trace is an observation of the process. By introducing the trace model, specifications for CSP processes can be written. A specification is a predicate describing a typical member of a set of traces, which is satisfiable if and only if it contains a non-empty prefix closed subset.

The trace model supports a formal treatment of non-diverging, deterministic processes. It does not distinguish between deadlock and divergence, nor does it model internal non-determinism. From the point of view of information flow the trace model supports the formal treatment of transput leaks (information flow due to inputs and outputs) and synchronization leaks (information flow due to the sequence of channels successfully used to communicate with a system), [Jacob 88b].

The failures model of CSP relates each process with a set of pairs, the first component of which is the trace of the process and the second component of which is a finite set of alphabet events which the process may refuse to engage in. This model supports non-determinism, but it does not provide an adequate treatment of certain divergence properties, [Hoare et al 81] and [Brookes et al]. From the point of view of information flow the failures model supports capability leaks, that is information flow due to knowing what the system is next capable of doing (by taking the complement of what it may refuse), as well as transput and synchronization leaks. The readiness model of CSP, [Olderog & Hoare 86], is similar to the failures model of CSP, it records what events the process is ready to engage in and provides a more intuitive explanation of capability leaks. There is a connection between the failures and readiness model explained in [Olderog & Hoare 86]. The failures model is not used in this thesis because of space constraints.

Trace Semantics of CSP

In this subsection the trace semantics for a subset of CSP processes is given. The full syntax is not treated because only a subset of the process algebra will be used to illustrate information flow properties, for a full semantics the reader is referred to [Roscoe 82]. First the trace model $\mathcal{M}_{\mathcal{T}}$ is defined.

Definition 1 *If Σ is a universal alphabet then $\mathcal{M}_{\mathcal{T}}$ is the set of all those subsets S of Σ^* satisfying:*

1. $\langle\rangle \in S$

2. $s^\frown w \in S \Longrightarrow s \in S$ $\diamond$

These two conditions state that each member of $\mathcal{M}_{\mathcal{T}}$ is a non-empty, prefix closed set of traces. If $\mathcal{T}_{\mathcal{R}}$ is the semantic function from the process algebra of CSP to $\mathcal{M}_{\mathcal{T}}$

then

$$
\begin{aligned}
\mathcal{T}_{\mathcal{R}}(STOP_A) &= \{\langle\rangle\} \\
\mathcal{T}_{\mathcal{R}}(SKIP_A) &= \{\langle\rangle, \langle\surd\rangle\} \\
\mathcal{T}_{\mathcal{R}}((a \rightarrow P)) &= \{\langle\rangle\} \\
&\quad \cup \\
&\quad \{\, \langle a\rangle^\frown s \mid s \in \mathcal{T}_{\mathcal{R}}(P) \,\} \\
\mathcal{T}_{\mathcal{R}}(P [\!] Q) &= \mathcal{T}_{\mathcal{R}}(P) \cup \mathcal{T}_{\mathcal{R}}(Q) \\
\mathcal{T}_{\mathcal{R}}(P \sqcap Q) &= \mathcal{T}_{\mathcal{R}}(P) \cup \mathcal{T}_{\mathcal{R}}(Q) \\
\mathcal{T}_{\mathcal{R}}(P \| Q) &= \{\, t \mid t \upharpoonright \alpha P \in \mathcal{T}_{\mathcal{R}}(P) \wedge t \upharpoonright \alpha Q \in \mathcal{T}_{\mathcal{R}}(Q) \,\} \\
\mathcal{T}_{\mathcal{R}}(P;Q) &= \{\, s \mid s \in \mathcal{T}_{\mathcal{R}}(P) \wedge \neg\surd \textbf{ in } s \,\} \\
&\quad \cup \\
&\quad \{\, s^\frown t \mid (s^\frown\langle\surd\rangle \in \mathcal{T}_{\mathcal{R}}(P) \wedge t \in \mathcal{T}_{\mathcal{R}}(Q) \,\}
\end{aligned}
$$

The meaning of $\mu P.F(P)$ is the unique fixed point of the contraction mapping with respect to a complete metric on $\mathcal{M}_{\mathcal{T}}$ represented by F, for a proper treatment of the meaning of recursion in $\mathcal{M}_{\mathcal{T}}$ the reader is referred to [Roscoe 82] or [Reed 88]. Note that $P \sqcap Q$ is indistinguishable from $P [\!] Q$ in $\mathcal{M}_{\mathcal{T}}$.

2.2 TCSP and its timed trace semantics

The abstract syntax of Timed CSP

The letters P, Q, R range over syntactic processes; the letters a, b range over the alphabet αP of a process and X, Y over subsets of αP. F denotes an "appropriate" composition of the syntactic operators.

TCSP Abstract syntax

$$
\begin{aligned}
P \;::=\; & STOP \mid SKIP \mid WAIT\ \ t \mid (a \rightarrow P) \mid P [\!] Q \mid P \sqcap Q \mid P \parallel Q \mid \\
& P_X \|_Y Q \mid P;Q \mid P \backslash X \mid \mu P.F(P)
\end{aligned}
$$

The synchronized parallel operator, $\|$, places two processes in lockstep. In the parallel combination $P \| Q$, the processes P and Q must co-operate on every event that is performed. $P \| Q$ is a special case of alphabet parallelism $P_X \|_Y Q$. The construct $P \| Q$ is ignored.

Notational preliminaries

The reader should consult [Davies & Schneider 89] for an excellent introduction to timed CSP and its associated notation. Most of the notation for TCSP used in this

thesis is defined in this section. The timed trace model of CSP is the simplest model of Timed CSP – timed refusal and stability information is omitted – therefore the times at which events first become available must be identified in order to reach a satisfactory definition of hiding. The identification is achieved by placing a hat upon an event whenever it occurs at the first moment of availability.

Definition 2 *If A is an alphabet then*

$$\hat{A} \triangleq A \cup \{ \hat{a} \mid a \in A \}$$

$\diamond$

A timed event is just a record of when an event occurred.

Definition 3 *If A is an alphabet then*

$$TA \triangleq [0, \infty) \times A$$

$\diamond$

If an observer also knows the time at which an event becomes available then hatted events must be included in a set of timed events.

Definition 4 *If A is an alphabet then*

$$T\hat{A} \triangleq [0, \infty) \times \hat{A}$$

$\diamond$

A timed trace is a finite sequence of observable events in the history of a process, each labelled with the time at which it occurs. The events are presented in chronological order.

Definition 5 *If A is an alphabet then*

$$(TA)_{\leq} \triangleq \{ s \in (TA)^* \mid \text{if } (t, a) \text{ precedes } (t', a') \text{ in } s, \text{ then } t \leq t' \}$$

$\diamond$

The operator $\upharpoonright$ is overloaded in timed CSP to denote the restriction of a trace to events before a given time on the right hand side of the $\upharpoonright$ operator. If the argument on the right hand side is a set rather than a time then the conventional meaning of restricting a trace to a set of events is intended.

Definition 6 *If A is a set of events then*

$$\begin{array}{lcll} \langle\rangle \upharpoonright A & \triangleq & \langle\rangle & \\ (\langle(t,a)\rangle^\frown s) \upharpoonright A & \triangleq & \langle(t,a)\rangle^\frown (s \upharpoonright A) & \text{if } a \in A \\ & \triangleq & s \upharpoonright A & \textit{otherwise} \\ (\langle(t,\hat{a})\rangle^\frown s) \upharpoonright A & \triangleq & \langle(t,\hat{a})\rangle^\frown (s \upharpoonright A) & \text{if } a \in A \\ & \triangleq & s \upharpoonright A & \textit{otherwise} \end{array}$$

◇

Restriction can be generalised to sets of traces.

Definition 7 *If A is a set of events and T is a set of timed traces then*

$$T \upharpoonright A = \{\, s \upharpoonright A \mid s \in T \,\}$$

◇

The operator *hstrip* strips the hats from a timed trace.

Definition 8 (Stripping)

$$\begin{array}{rcl} \mathit{hstrip}(\langle\rangle) & \triangleq & \langle\rangle \\ \mathit{hstrip}(\langle(t,a)\rangle^\frown s) & \triangleq & \langle(t,a)\rangle^\frown \mathit{hstrip}(s) \\ \mathit{hstrip}(\langle(t,\hat{a})\rangle^\frown s) & \triangleq & \langle(t,a)\rangle^\frown \mathit{hstrip}(s) \end{array}$$

◇

Stripping can also be extended to sets of traces in the obvious way.

Definition 9

$$\mathit{Hstrip}(T) \triangleq \{\, \mathit{hstrip}(t) \mid t \in T \,\}$$

◇

It is sometimes necessary to know what events a process is willing to engage in or what events are present in a trace.

Definition 10 *The events a process, P, engages in is denoted $\sigma(P)$. The events present in a timed trace, t, is denoted $\sigma(t)$.* ◇

A time delay can be added to a timed trace.

Definition 11

$$\begin{aligned} \langle\rangle + t &\triangleq \langle\rangle \\ (\langle (t_1, a)\rangle^\frown s) + t &\triangleq \langle (t_1 + t, a)\rangle^\frown (s + t) \end{aligned}$$

$\diamond$

An equivalence relation $\cong$ on traces is defined as follows

Definition 12 (Equivalence) *If u and v are traces then*

$$u \cong v \iff u \text{ is a permutation of } v$$

$\diamond$

As both u and v are timed traces, only events occurring at the same time may be interchanged. The following auxiliary definitions are used in other definitions.

Definition 13

$$CL_{\cong}(S) \triangleq \{ s \mid \exists w \in S \cdot s \cong w \}$$

$\diamond$

The closure operator $CL_{\cong}$ is needed to ensure that events which occur simultaneously can be recorded in either order.

The final definition in this section is needed in the semantic definition of parallel composition.

Definition 14 *If v and w are timed traces, which may contain hatted events, such that*

$$hstrip(v) = hstrip(u)$$

then they can run in parallel giving the trace

$$t = v \odot w$$

such that the n^{th} event of t is hatted if and only if the n^{th} event of v or w is hatted.
$\diamond$

The timed trace model of TCSP

The semantic treatment of Timed CSP is quite different from that of untimed CSP, but the intuition behind their use remains the same. The notion of process alphabet introduced in [Hoare 85] is discarded in Timed CSP; synchronisation of events is achieved by means of an alphabeticised parallel operator.

For the semantics of all the Timed CSP, TCSP, operators the reader is directed to [Reed 88]. The semantics for the subset of TCSP processes and operators used in this thesis are given; first

Definition 15 *If Σ is the universal alphabet then TM_T is the set of all those subsets S of $(T\hat{\Sigma})^*_{\leq}$ satisfying:*

1. $\langle\rangle \in S$
2. $s^\frown w \in S \Longrightarrow s \in S$
3. $s \in S \Longrightarrow hstrip(s) \in S$
4. $s \in S \wedge s \cong w \Longrightarrow w \in S$
5. $s^\frown\langle(t,a)\rangle \in S \Longrightarrow \exists t', t'' \cdot t' \leq t \ \wedge \ (s \upharpoonright t')^\frown\langle(t',\hat{a})\rangle \in S \ \wedge$
   $(t' \leq t'' < t \Longrightarrow (s \upharpoonright t'')^\frown\langle(t'',a)\rangle \in S)$
6. $\forall t \in [0,\infty) \cdot \exists n(t) \in \mathbb{N} \cdot \forall s \in S \cdot (end(s) \leq t \Longrightarrow \#s \leq n(t))$

where $end(s)$ denotes the latest time of any of the timed events in s. $\diamond$

The first two conditions are inherited from the trace model of CSP. The third condition states that when an event becomes available, $\hat{a}$, it can also be communicated "normally", a. The fourth condition states that the order of events which happen at the same time is irrelevant. The fifth condition states that when an event is available it must have been continuously available since *becoming* available at some earlier time. The last condition ensures that infinitely many events cannot occur in a finite period. Next the semantic function for the timed process algebra is defined.

Definition 16 *If $TCSP$ denotes the process algebra for Timed CSP and TM_T is the semantic domain then*

$$\mathcal{T}_T : TCSP \rightarrow TM_T$$

is the semantic function. $\diamond$

The deadlock process $STOP$ can neither perform any external events nor make any internal progress.

Definition 17 $(STOP)$

$$\mathcal{T}_T(STOP) = \{\langle\rangle\}$$

$\diamond$

The delay operator, $WAIT\ t$, models the delayed successful termination of a process, introducing a delay of time t before the special event $\surd$, denoting termination, becomes available.

Definition 18 (Waiting)

$$\mathcal{T}_T(WAIT\ t) = \{\langle\rangle\} \cup \{\langle(t,\hat{\surd})\rangle\} \cup \{\,\langle(t',\surd)\rangle \ \mid \ t \leq t'\,\}$$

$\diamond$

The sequential composition operator is used to transfer control from one process to another. For the process $P;Q$ transfer of control from P to Q occurs once P signals successful termination, denoted by the special event $\surd$. The sequential composition operator hides $\surd$ from the environment so that the $\surd$ occurs as soon as possible.

Definition 19 (Sequential composition) *If P and Q are processes then*

$$\mathcal{T}_T(P;Q) = \left\{ \begin{array}{ll} CL_{\cong} & (\{\, s \mid s \in \mathcal{T}_T(P) \wedge \surd \notin \sigma(s) \,\} \cup \\ & \{s^\frown(w+t) \mid s^\frown\langle(t,\hat{\surd})\rangle \in \mathcal{T}_T(P) \ \wedge \ \surd \notin \sigma(s) \wedge w \in \mathcal{T}_T(Q)\} \\ &) \end{array} \right.$$

$\diamond$

The process $a \rightarrow P$ represents a process which is initially prepared to engage in an event a. A short delay, δ, follows the occurrence of the event a and the process then behaves as P. The delay introduced by the prefix operator models the minimum time required for a sequential process to recover from participation in an event.

Definition 20 (Prefixing)

$$\begin{array}{rcl} \mathcal{S}(a \rightarrow P) & = & \{\langle\rangle\} \cup \{\, \langle(0,\hat{a})\rangle^\frown(s+\delta) \mid s \in \mathcal{T}_T(P) \,\} \\ & & \cup \{\, \langle(t,a)\rangle^\frown(s+t+\delta) \mid s \in \mathcal{T}_T(P) \wedge t \geq 0 \,\} \end{array}$$

$\diamond$

The notion of non-deterministic and environmental choice remain unchanged from the trace model with the introduction of time.

Definition 21 (Internal Choice)

$$\mathcal{T}_T(P \sqcap Q) = \mathcal{T}_T(P) \cup \mathcal{T}_T(Q)$$

$\diamond$

Definition 22 (External Choice)

$$\mathcal{T}_T(P [\!] Q) = \mathcal{T}_T(P) \cup \mathcal{T}_T(Q)$$

$\diamond$

The alphabeticised parallel operator provides for synchronisation in Timed CSP. In parallel combination $P_X \|_Y Q$, process P can perform only those events drawn from the set X. Similarly, process Q is restricted to events from the set Y. The two processes must synchronise on events from the intersection $X \cap Y$.

Definition 23 (Alphabeticised parallel composition)

$$\mathcal{T}_T(P_X \|_Y Q) = \{\, s \mid \exists s_P \in \mathcal{T}_T(P); s_Q \in \mathcal{T}_T(Q) \cdot s \in s_P \, {}_X\|_Y s_Q \,\}$$

where

$$u_X \|_Y v \quad \triangleq \quad \left\{ \begin{array}{l} s : T\hat{\Sigma}^*_{\leq} \mid hstrip(s) \upharpoonright (X \cup Y) = hstrip(s) \\ \wedge hstrip(s) \upharpoonright X = hstrip(u) \wedge hstrip(s) \upharpoonright Y = hstrip(v) \\ \wedge s \upharpoonright (X - Y) = u \upharpoonright (X - Y) \wedge s \upharpoonright (Y - X) = v \upharpoonright (Y - X) \\ \wedge s \upharpoonright (X \cap Y) = u \upharpoonright (X \cap Y) \odot v \upharpoonright (X \cap Y) \end{array} \right\}$$

$\diamond$

Example 1 If

$$\begin{aligned} P &\triangleq (WAIT1; a \rightarrow STOP) \\ Q &\triangleq (a \rightarrow STOP) \end{aligned}$$

then

$$\begin{aligned} \mathcal{T}_T(P) &= \{\, \langle (t, a) \rangle \mid 1 \leq t \,\} \\ \mathcal{T}_T(Q) &= \{\, \langle (t, a) \rangle \mid 0 \leq t \,\} \end{aligned}$$

and

$$\mathcal{T}_T(P_{\{a\}} \|_{\{a\}} Q) = \{\, \langle (t, a) \rangle \mid 1 \leq t \,\}$$

△

An example of two traces acting in parallel is the following.

Example 2 If

$$\begin{aligned} u &\triangleq \langle (0, a), (1, b), (1, c) \rangle \\ v &\triangleq \langle (0, d), (1, b), (1, c) \rangle \end{aligned}$$

$X = \{a, b, c\}$ and $Y = \{b, c, d\}$ then

$$u_X \|_Y v = \{\langle (0, a), (0, d), (1, b), (1, c) \rangle, \langle (0, d), (0, a), (1, b), (1, c) \rangle\}$$

Note that a and d occur at the same time and so may be observed in either order. △

2.3 Category theory — some abstract nonsense

Categories

This section is a brief review of the category theory needed to enable a reader to, in principle, follow the categorical arguments. The basic components of a category are *objects*, *arrows* and *composition* of arrows. The following are examples of categories:

Category	Objects	Arrows
Set	all sets	all functions between sets
Top	all topological spaces	all continuous functions between topological spaces
Vect	vector spaces	linear transformations
Grp	groups	group homomorhisms

The arrows of these categories are composed together in the usual functional way. The category axioms represent a very weak abstraction and in many categories arrows are not functions. For example any preorder is a category: the objects are elements of

the preorder and the ordering relation, $\leq$, defines the arrows between objects. There is an arrow from an object x to an object y if and only if $x \leq y$. A consequence of this definition is that there is at most one arrow between any two objects in a preorder category. Composition of arrows in a preorder category is transitivity of the ordering relation.

Another example of a category where the arrows are not functions is a monoid. The category has a single object, the set underlying the monoid, and its arrows are elements of the monoid. Composition of arrows is given by multiplication of the monoid elements. For an introduction to category theory, [Barr & Wells 90, Goguen 89] or the first four chapters of [Goldblatt 84] are recommended.

Definition 24 *A category $\mathcal{C}$ consists of:*

- *a collection obj($\mathcal{C}$) of $\mathcal{C}$-objects*
- *for each A, B $\in$ obj($\mathcal{C}$), a collection $\mathcal{C}(A,B)$ of $\mathcal{C}$-arrows (or $\mathcal{C}$-morphisms) from A to B. An arrow f in $\mathcal{C}(A,B)$ is denoted $f : A \rightarrow B$.*
- *for each A, B, C $\in$ obj($\mathcal{C}$), an infix composition operation*
  $\_;\_ : \mathcal{C}(A,B) \times \mathcal{C}(B,C) \rightarrow \mathcal{C}(A,C)$

  such that

  1. *If $(A,B) \neq (A',B')$ then $\mathcal{C}(A,B) \cap \mathcal{C}(A',B') = \{\}$.*
  2. *(existence of identities) for each $A \in$ obj($\mathcal{C}$), there is an arrow $id_A \in \mathcal{C}(A,A)$ such that for any arrow $g \in \mathcal{C}(A,B)$, $id_A;g = g$ and for any arrow f in $\mathcal{C}(B,A)$, $f;id_A = f$.*
  3. *(associativity of composition) for any $f \in \mathcal{C}(A,B)$, $g \in \mathcal{C}(B,C)$ and $h \in \mathcal{C}(C,D)$, $f;(g;h) = (f;g);h$.*

$\diamond$

Products and Coproducts

The concept of a product plays a central role in this thesis. In the category of sets a product corresponds to cartesian product. In a preorder a product of two objects is their greatest lower bound. It is this universality which makes the categorical definition of product so important.

Definition 25 (Product) *A product in a category $\mathcal{C}$ of two objects c and d is a $\mathcal{C}$-object $c \times d$ together with a pair of arrows,*

$$pr_c : c \times d \rightarrow c$$
$$pr_d : c \times d \rightarrow d$$

such that for any pair of arrows of the form $f : e \to d$ *and* $g : e \to c$ *there is exactly one arrow* $\langle f, g\rangle : e \to c \times d$ *making the diagram of figure 2.1 commute, i.e such that*

$$\langle f, g\rangle; pr_c = f \text{ and } \langle f, g\rangle; pr_d = g$$

$\langle f, g\rangle$ *is called the product arrow of* f *and* g *with respect to the projections* pr_c *and* pr_d. ◇

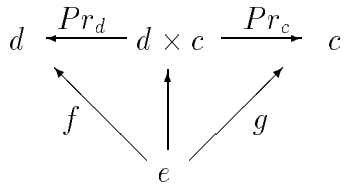


Figure 2.1: A product

The dual of a categorical construction is that construction with all the arrows reversed. For example the dual of a product is a coproduct which is defined as follows.

Definition 26 (Coproduct) *A coproduct in a category* $\mathcal{C}$ *of two objects* c *and* d *is a* C*-object* $c + d$ *together with a pair of arrows,*

$$i_c : c \to c + d$$
$$i_d : d \to c + d$$

such that for any pair of arrows of the form $f : c \to e$ *and* $g : d \to e$ *there is exactly one arrow* $[f, g] : c + d \to e$ *such that*

$$i_c; [f, g] = f \text{ and } i_d; [f, g] = g$$

$[f, g]$ *is the coproduct arrow of* f *and* g *with respect to the inclusions* i_c *and* i_d. ◇

Note that it is indeed the case that the definition of a coproduct is the same as product with the arrows reversed.

Definition 27 (Initial object) *An object* **0** *is initial in a category if for every object,* x*, in that category there is one and only one arrow from* **0** *to* x. ◇

Remark 1 *The coproduct of the empty diagram is simply an object with the property that for any other object,* c*, in* $\mathcal{IF}$ *there is exactly one arrow from the coproduct object to* c*. In other words the coproduct object of the empty diagram (no objects and no arrows) is an initial object for* $\mathcal{IF}$*. Thanks go to Joseph Goguen for explaining this to me.*

The following series of lemmas illustrate the equational reasoning of category theory, show how the definition of product and coproduct is used in proofs and provide some simple but useful results.

Lemma 1 *In any category with products* $\langle pr_c, pr_d \rangle = id_{c \times d}$.

Proof

Suppose the object e, in figure 2.1, is $c \times d$ and that f and g are pr_d and pr_c respectively. The product arrow from $d \times c$ to $d \times c$ is $\langle pr_d, pr_c \rangle$, further it is the unique arrow, h, which satisfies the equations.

$$h; pr_d = pr_d \text{ and } h; pr_c = pr_c.$$

Substituting the identity arrow $id_{d \times c}$ for h shows that $id_{d \times c}$ also satisfies the two equations, therefore by uniqueness $id_{d \times c} = \langle pr_d, pr_c \rangle$. □

To prove the rest of the lemmas the following definition is needed.

Definition 28 (product arrows) *If* $f : a \to b$ *and* $g : c \to d$ *then*

$$f \times g : a \times c \to b \times d$$

is the arrow $\langle pr_a; f, pr_c; g \rangle$, *provided* $a \times c$ *and* $b \times d$ *are defined.* ◇

Lemma 2 *In any category with products* $\langle g, k \rangle; (f \times h) = \langle g; f, k; h \rangle$.

Proof

If

$$g : e \to a, \quad k : e \to c$$

and

$$h : c \to d, \quad f : a \to b$$

and both $a \times c$ and $b \times d$ exist, then

$$\begin{array}{lcll}
(\langle g, k \rangle; (f \times h)); pr_d & = & \langle g, k \rangle; ((f \times h); pr_d) & \text{[associativity of arrows]} \\
& = & \langle g, k \rangle; (\langle pr_a; f, pr_c; h \rangle; pr_d) & \text{[by defn. of product arrow]} \\
& = & \langle g, k \rangle; (pr_c; h) & \text{[by defn. of product]} \\
& = & (\langle g, k \rangle; pr_c); h & \text{[associativity of arrows]} \\
& = & k; h & \text{[by defn. of product].}
\end{array}$$

Similarly

$$\begin{array}{lcll}
(\langle g, k \rangle; (f \times h)); pr_b & = & \langle g, k \rangle; ((f \times h); pr_b) & \text{[associativity of arrows]} \\
& = & \langle g, k \rangle; (\langle pr_a; f, pr_c; h \rangle; pr_b) & \text{[by defn. of product arrow]} \\
& = & \langle g, k \rangle; (pr_a; f) & \text{[by defn. of product]} \\
& = & (\langle g, k \rangle; pr_a); f & \text{[associativity of arrows]} \\
& = & g; f & \text{[by defn. of product].}
\end{array}$$

Finally by definition of product $\langle g; f, k; h\rangle$ is the unique arrow which satisfies

$$\langle g; f, k; h\rangle; pr_d = k; h \text{ and } \langle g; f, k; h\rangle; pr_b = g; f$$

therefore

$$\langle g, k\rangle; (f \times h) = \langle g; f, k; h\rangle$$

□

Lemma 3 *In any category with products* $(g \times k); (f \times h) = (g; f) \times (k; h)$*, assuming the arrows exist.*

Proof

Let pr_e and $pr_{e'}$ be projection arrows for the domain of the arrow $g \times k$.

$$\begin{array}{lcll} (g \times k); (f \times h) & = & \langle pr_e; g, pr_{e'}; k\rangle; (f \times h) & \text{[by definition 28]} \\ & = & \langle (pr_e; g); f, (pr_{e'}; k); h\rangle & \text{[by lemma 2]} \\ & = & \langle pr_e; (g; f), pr_{e'}; (k; h)\rangle & \text{[by associativity of arrows]} \\ & = & (g; f) \times (k; h) & \text{[by definition 28]} \end{array}$$

□

Lemma 4 *In any category, where* $a \times b$ *exists,* $id_a \times id_b = id_{a\times b}$*.*

Proof

By the identity axiom

$$id_{a\times b}; pr_a = pr_a; id_a \quad \text{and} \quad id_{a\times b}; pr_b = pr_b; id_b.$$

By definition of product

$$(id_a \times id_b); pr_b = pr_b; id_b \text{ and } (id_a \times id_b); pr_a = pr_a; id_a.$$

By uniqueness of the product arrow

$$id_a \times id_b = id_{a\times b}$$

□

Lemma 5 *Let* $\gamma : d \to c$*,* $f : c \to a$ *and* $g : c \to b$ *then*

$$\gamma; \langle f, g\rangle = \langle \gamma; f, \gamma; g\rangle,$$

in any category where $a \times b$ *exists.*

Proof

$$\begin{array}{lcll} \langle \gamma; f, \gamma; g\rangle; pr_b & = & \gamma; g & \text{[by defn of product]} \\ \langle \gamma; f, \gamma; g\rangle; pr_a & = & \gamma; f & \text{[by defn of product]} \end{array}$$

also

$$\begin{array}{lcll} \gamma; \langle f, g\rangle; pr_b & = & \gamma; g & \text{[by defn of product]} \\ \gamma; \langle f, g\rangle; pr_a & = & \gamma; f & \text{[by defn of product]} \end{array}$$

therefore by uniqueness of the product arrow

$$\gamma;\langle f,g\rangle = \langle \gamma;f, \gamma;g\rangle$$

□

Functors and adjunctions

A functor is a transformation from one category to another that preserves the categorical structure of its source.

Example 3 A monotonic function from one preorder to another is a functor when the preorders are viewed as categories. The objects of the category are the objects in the preorder and there is an arrow from x to y if and only if $x \leq y$. △

Definition 29 *A* functor *F from a category $\mathcal{C}$ to a category $\mathcal{D}$ is a function that assigns*

1. *to each $\mathcal{C}$-object* d, *a $\mathcal{D}$-object $F(d)$;*
2. *to each $\mathcal{C}$-arrow $f : d \to e$ a $\mathcal{D}$-arrow $F(f) : F(d) \to F(e)$ such that*

   i *$F(id_d) = id_{F(d)}$, for all $\mathcal{C}$-objects* d,

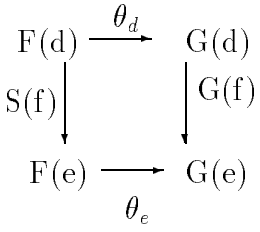
   ii *$F(f;g) = F(f);F(g)$, whenever $f;g$ is defined.*

◇

Categories were defined beforehand as collections of objects with arrows between them. Introducing functors enables a more abstract view of categories as objects, with functors as arrows between them. It turns out to be useful to pile abstraction on abstraction and consider functors as objects.

Given two categories $\mathcal{C}$ and $\mathcal{D}$ a new category is constructed denoted Funct$(\mathcal{C},\mathcal{D})$ whose objects are the functors from $\mathcal{C}$ to $\mathcal{D}$. The arrows of the category are the subject of the next definition.

Definition 30 (Natural transformations) *A natural transformation from a functor $F : \mathcal{C} \to \mathcal{D}$ to a functor $G : \mathcal{C} \to \mathcal{D}$ is an assignment θ that provides, for each $\mathcal{C}$-object* d, *a $\mathcal{D}$-arrow $\theta_d : F(d) \to G(d)$, such that for any $\mathcal{C}$-arrow $f : d \to e$ the following diagram commutes.*



The symbolism $\theta : F \xrightarrow{\bullet} G$ denotes that θ is a natural transformation from F to G. The arrows θ_d are called the components of θ. $\diamond$

Joseph Goguen in his paper [Goguen 89] states as dogma that

To each natural relationship between two functors $F, G : \mathcal{A} \to \mathcal{B}$ corresponds a natural transformation from F to G (or from G to F).

Natural transformations are used in this thesis to define what a statement about confidentiality is, and whether one confidentiality statement is "better" than another. These are both examples of natural relationships between functors. The existence of natural transformations is shown repeatedly in this thesis by proving the equality

$$F(f); \theta_e = \theta_d; G(f)$$

for any $f : d \to e$. The concept of a natural transformation is used in the definition of a colimit taken from [Mac Lane].

Definition 31 (Colimit) *Let C and J be categories (J is an index category). The diagonal functor*

$$\Delta : C \to C^J$$

sends each object c to the constant functor Δc (the functor which has the value c at each object $i \in J$ and the value 1_c at each arrow of J. If $f : c \to c'$ is an arrow of C, Δf is the natural transformation, $\Delta f : \Delta c \xrightarrow{\bullet} \Delta c'$, which has the same value f at each object i of J.

Each functor $F : J \to C$ is an object of C^J. A universal arrow $\langle r, u \rangle$ from F to Δ is called a **Colimit** *for F. It consists of an object r of C together with a natural transformation $u : F \xrightarrow{\bullet} \Delta r$ which is universal among natural transformations $\tau : F \xrightarrow{\bullet} \Delta c$.* $\diamond$

If J is the discrete category $\{0, 1\}$, a functor $F : \{0, 1\} \to C$ is a pair of objects a, b of C. The colimit of this functor is the coproduct of a and b.

Remark 2 *If the indexing category J is an arbitrary category then the objects and arrows of J might not be a set. In this thesis only colimits where the objects and arrows of J form a set are considered, these are called* **small** *colimits. In particular this means that all coproducts are small. The further assumption that all products are small is made. These assumptions are made explicit in the next chapter.*

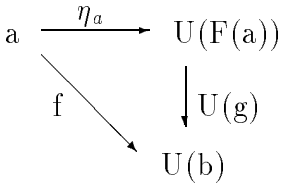
In the category **Set** the objects are sets and the arrows are total functions. In this case the collection of all objects in **Set** is not small because there is no set of all sets. Any collection of sets which is itself a set is guaranteed a product or coproduct. In **Set**, product and coproduct correspond to cartesian product and disjoint union respectively.

The last concept in category theory to be covered is that of the adjunction, it incorporates the notion of a natural transformation linking two functors which map to and from two categories. Adjunctions underly most of category theory and appear in many situations in general mathematics. Adjunctions are used in two ways in this thesis, in the simpler form of a Galois correspondence between preorders and in its general form. The Galois correspondence is used to give results about how information flow properties of two systems behave under product composition. The general form of the adjunction is used to show how an invariant property of an interface to a system can give information flow security properties.

Definition 32 (Adjunctions) *If $\mathcal{A}$ and $\mathcal{B}$ are categories and*

$$F : \mathcal{A} \to \mathcal{B} \qquad \textit{and} \qquad U : \mathcal{B} \to \mathcal{A}$$

are functors, then F is **left adjoint** *to U and U is* **right adjoint** *to F provided there is a natural transformation $\eta : Id_{\mathcal{A}} \xrightarrow{\bullet} F;U$ such that for any objects $a : \mathcal{A}$ and $b : \mathcal{B}$ and any arrow $f : a \to U(b)$, there is a unique arrow $g : F(a) \to b$ such that the following diagram commutes.*



$\diamond$

This definition says that there is a way to convert any arrow $f : a \to U(b)$ to an arrow $g : F(a) \to b$ such that g solves the equation $f = \eta_a; U(?)$ and that solution is unique. The existence of the unique arrow $g : F(a) \to b$ such that $f = \eta_a; U(g)$ is an arrow-lifting property which arises in many areas of mathematics such as group theory. The following example of an adjunction and arrow-lifting is taken from [Barr & Wells 90] and is extended in chapter 6 to illustrate lifting properties of an interface.

Example 4 Let **Mon** denote the category of monoids, whose objects are monoids and whose arrows are monoid homomorphisms. Let U : **Mon** $\to$ **Set** denote the underlying set functor which forgets the monoidal algebraic structure just leaving the set of elements in the monoid.

A monoid is a triple $(U(M), \cdot, 1)$, where $U(M)$ is the underlying set of M. For example the trivial monoid consisting of just the identity element has a singleton set for $U(M)$. 1 denotes the identity of the monoid for the binary operator '$\cdot$' on $U(M)$. A homomorphism of monoids is a triple (M, f, N) where f is a function from $U(M)$ to $U(N)$ that preserves the binary operation. The underlying functor U is defined on homomorphisms by $U(M, f, N) = f$.

Given a set X the free monoid $F(X) = (X^*, ^\frown, \langle\rangle)$, there is also an arrow of **Set** associated with X denoted η_X. The arrow η_X is an inclusion function from X to the set of all strings over X, X^*. For example if $x \in X$ then $\eta_X(x) = \langle x \rangle$ the sequence of length one. F and U form an adjoint pair; the details of the proof of this can be found in [Barr & Wells 90]. $\triangle$

Chapter 3

A model of information flow

In this chapter a mathematical model of information flow is constructed. This model provides the semantic space for an abstract calculus presented in the next chapter. There are several reasons for being interested in the foundations for such a calculus. The main reason is to justify to a sceptical reader why the axioms and laws for the calculus are necessarily consistent? Where have they come from? Does the calculus have any connection with the confidentiality of real systems?

Questions like these should not be asked by people who use a calculus but by those who select a method for specifying and constructing secure systems, because they have to be sure that it works. This chapter is intended for those people.

The answers to these questions are usually given by deciding on a reasonable way of viewing confidentiality for systems, a mathematical model. The model should correspond closely with our intuition for what confidentiality means so that there are few questions about its appropriateness. The axiomatic basis and laws for the calculus, and associated development methods, can be checked against the model. If one has accepted the model then one has accepted the laws and axioms which pass the checks.

3.1 Information flow and confidentiality

Category theory is used to give a general semantics for confidentiality because it is particularly suited for this role. For example in set theory the definition of cartesian product has the unfortunate property that

$$\{a, b\} \in (a, b)$$

this is accidental and not an intrinsic property of the concept of product. The definition of product in category theory is free of such "implementation" problems and can be found in many guises; for example as cartesian product in set theory or greatest lower bound in lattice theory. It is the abstractness of categorical concepts which makes it suitable for describing a calculus for confidentiality which can be used in many models of computation. Category theory can also be used to make simple ideas

appear to be difficult and profound, this is not the intention of using category theory in this thesis.

The mathematical model used to formalize our intuition of confidentiality will be that of information flow. The particular form of information flow will depend upon the particular model of computation of interest. For example if the model of computation involved probabilities then the definition of information flow would also involve probabilities. In other models of computation which didn't involve probability the definition of information flow would also change to reflect this. There are however universal properties which information flow has, it is these which are captured using category theory.

To describe and reason about information flow the categorical approach taken is at two levels: the first is at an abstract level where no concrete interpretation is put upon the objects and arrows of the category. At the second level the concrete category can be obtained by, for example, instantiating the objects with sets and the arrows with relations. In an abstract category general definitions and results can be formulated independently of any one model of computation. In each concrete category the general results can be specialized to give results for particular models of computation.

In this thesis a system, an observation of a system, and the inference that can be made, from a system and an observation of it, are all objects in an information flow category. The object representing the system denotes all the information about that system; whereas the object representing an observation denotes only some of the information about that system. The inference about a system from an observation of it, is a combination of these two pieces of information. The two objects representing a system and an observation of it are combined by the categorical construction of product. A similar, but more complete, intuition to this occurs in [Goguen 92], where Sheaf theory is used to relate observations together to form a system. One of the results of this paper is a formulation of information flow security. The relationship between the formulation in [Goguen 92] and the approach of this thesis is discussed in the final chapter of this thesis.

The arrows in an Iflow category provide a link between objects and are an essential part of the definition of inference. In the abstract the arrows are too general to give a good intuition for. In particular concrete interpretations the arrows denote restriction. For example when objects represent behaviours over events, arrows represent restriction of behaviours to particular events. Some assumptions, arising from a joint effort with Joseph Goguen, are made about the abstract information flow category $\mathcal{IF}$.

Postulate 1 *$\mathcal{IF}$ has all small products and coproducts.*

The motivation for $\mathcal{IF}$ having products and coproducts is that they are generalisations of greatest lower bound and least upper bound which represent the independent parallel and non-deterministic choice forms of interconnection of information. The principle that systems can be put together using limits and colimits has been known for some time, for example [Goguen 73b], or for a more recent treatment [Goguen 92].

As discussed in chapter 2, the requirement for *small* products and coproducts is a category theoretic condition which means that any collection of objects which is a *set* has a product and coproduct. Categorical product and coproduct appear as cartesian product and disjoint union in the category of sets and total functions. All of these are concrete instances of the abstract definition of product and coproduct.

To reason about inferences of systems from observations some categorical construction is needed to combine the observation and the system. Categorical product is the fundamental construction which is used for constructing an inference of a system from an observation. The step of composing systems together via categorical product is also made. Using the same categorical construction simplifies the theory needed to reason about confidentiality. Systems are also composed via the dual of product, coproduct. Product and coproduct composition of systems turn out to, more or less, coincide with parallel and choice composition of systems. To avoid exceptions to theorems developed in the abstract category all small products and coproducts are assumed to exist. The condition that $\mathcal{IF}$ has all small products and coproducts means that only collections of objects which form a set can be guaranteed to have a product or a coproduct. To apply the theory developed in the abstract category suitable restrictions are assumed in the concrete categories.

Recall from chapter 2 that the coproduct of the empty set of objects and arrows is an initial object. An initial object is used to represent inconsistent inference which can then be avoided. The initial object also has a role to play in making confidentiality statements where one doesn't care about the confidentiality properties of a system. This is analogous to allowing a specification of a system to be 'false', in which case any system would satisfy such a specification. An initial object in a preorder category is a bottom for that preorder. The category **Set** also satisfies this assumption, the initial object is the empty set. The form of the following assumption is taken from [Walters 89].

Postulate 2 *The category $\mathcal{IF}$ is distributive, that is the following arrow is an isomorphism*

$$\nabla_{X\times(Y+Z)}((id_X \times i_1) + (id_X \times i_2)) : X \times Y + X \times Z \to X \times (Y + Z)$$

where $\nabla_X : X+X \to X$ is the codiagonal functor; $i_1 : X \to X+Y$ and $i_2 : Y \to X+Y$ are injections.

This assumption allows product to distribute over coproduct, this is important because inference about systems is calculated by taking the product of a system, P, with an observation, o, to give $P \times o$ as the inference. Postulate 2 allows inference to be distributed over the coproduct of two systems, $P + Q$. This means that the inference about the coproduct of two systems, $(P + Q) \times o$, can be decomposed into the inference about the individual systems, $P \times o$ and $Q \times o$.

Postulate 2 also allows implication and negation to be defined in a calculus of information flow. In a preorder the assumption means that for any element b, the function defined by forming the greatest lower bound of its argument with b preserves

least upper-bounds. In **Set** this assumption holds since cartesian product distributes over disjoint union.

Remark 3 *If*

$$\coprod_{i:I}(b \times x_i)$$

denotes the coproduct of the products of b with each x_i then when I is empty, the coproduct will be of the empty diagram. As discussed in remark 1 of chapter 2 the coproduct of the empty diagram is an initial object. By postulate 2 product distributes over coproduct to give an isomorphism to

$$b \times (\coprod_{i:I} x_i)$$

which is the product of b with the initial object, when I is empty. To summarise, in $\mathcal{IF}$ the product of any object b with an initial object, $\mathbf{0}$, is isomorphic to $\mathbf{0}$. Thanks go to Joseph Goguen for explaining this to me.

The isomorphism of the product $b \times o$ with o, for any object $b : \mathcal{IF}$, makes categorical product strict. That is if an inconsistent inference arises then any further observation combined with that inference is inconsistent.

3.2 Limiting Inference

Before formalizing statements about confidentiality it is worth considering what informally is meant by confidentiality in this thesis.

Example 5 An Automated Teller Machine, ATM, is a machine which dispenses money to customers of banks. A CSP process description of a simple ATM which only dispenses money to two customers is given.

$$\begin{aligned} ATM \quad \triangleq \quad & pin_1 \rightarrow amount_1?n \rightarrow dispense_1!n \rightarrow receipt_1!n \rightarrow ATM \\ & \mid pin_2 \rightarrow amount_2?n \rightarrow dispense_2!n \rightarrow receipt_2!n \rightarrow ATM \end{aligned}$$

The events pin_1 and pin_2 are the Personal Identification Numbers of each customer which authenticate the identity of that customer to the ATM. The events $amount_1?n$ and $amount_2?n$ are requests from each customer for n pounds of cash. The events $dispense_1!n$ and $dispense_2!n$ denote the release of the requested amount of money. Finally the events $receipt_1!n$ and $receipt_2!n$ denote an acknowledgement by the machine that n pounds have been released before returning to a state in which the ATM is ready to participate in another cash transaction.
If one customer is limited to the events

$$CUST_1 \triangleq \{ pin_1, amount_1?n, dispense_1!n, receipt_1!n \mid n : \mathbb{N} \}$$

and the other to the events

$$CUST_2 \triangleq \{\, pin_2, amount_2?n, dispense_2!n, receipt_2!n \mid n : \mathbb{N} \,\}$$

then these sets form the basis of a customer's potential observations. Both $CUST_1$ and $CUST_2$ are "windows" through which a customer can interact with the ATM. Some potential observations through $CUST_1$ in the trace model of CSP are

$$\langle\rangle, \langle pin_1\rangle, \langle pin_1, amount_1?5\rangle,$$
$$\langle pin_1, amount_1?5, dispense_1!5, pin_1, receipt_1!5, amount_1?10\rangle$$

A reasonable requirement for confidentiality would be that customer 1 must not be able to infer from its interactions with the ATM how much money customer 2 receives and vice-versa. For example when customer 1 makes the interaction

$$\langle pin_1, amount_1?5, dispense_1!5\rangle$$

then that customer has no basis for inferring what customer 2 might have received from the ATM even if customer 1 has the CSP design of the ATM to refer to. A second Automated Teller Machine, ATM2, might be designed as follows:

$$\begin{aligned} ATM2 \triangleq\ & pin_1 \rightarrow amount_1?m \rightarrow dispense_1!m \rightarrow pin_2 \rightarrow \\ & amount_2?n \rightarrow dispense_2!n \rightarrow receipt_1!m \rightarrow receipt_2!m \rightarrow ATM2 \end{aligned}$$

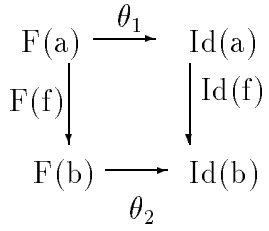
With this design of an ATM the amount of money customer 1 receives is leaked to customer 2. In this case the confidentiality requirement is not satisfied. △

By inspection of the description of the ATM's of example 5 one could be convinced whether or not it satisfied a confidentiality requirement, however for more realistic systems and requirements it is much more difficult to see that a statement about confidentiality holds. The first step in constructing a theory for the mathematical analysis and construction of confidential systems is the abstract characterization of a confidentiality statement.

Definition 33 *A confidentiality statement, $\mathcal{S}$, is a functor $F : \mathcal{IF} \rightarrow \mathcal{IF}$ along with a natural transformation*

$$\theta : F \stackrel{\bullet}{\rightarrow} Id_{\mathcal{IF}}$$

This means that for all $f : a \rightarrow b$ the following diagram commutes



where θ_1 and θ_2 are components of the natural transformation θ. $\diamond$

Remark 4 *Joseph Goguen pointed out to me that there can be more than one natural transformation to the identity from a functor F. This means that a confidentiality statement, $\mathcal{S}$, is not characterized by the functor, but is precisely characterized by the natural transformation to the identity. The natural transformation includes the functor F as part of its meaning. For notational convenience $\mathcal{S}$ is also used to denote the functor associated with the natural transformation. When the reader sees the term "confidentiality statement" it should be understood as the natural transformation to the identity.*

Naturality is a basic coherence condition for confidentiality statements. The functor F maps an observation to the object which denotes the required degree of confidentiality, that is the maximum amount of inference which is allowed from an observation. Any arrows between observations will be preserved by F because it is a functor. This is reasonable because if there is a dependence between observations then some dependence should be present between inferences. There is no guarantee however that the object representing the desired degree of confidentiality is consistent with the original observation, never mind the dependencies between these inference objects. Both of these problems are solved by requiring a natural transformation to the identity functor.

The natural transformation is the constraint which associates what a potential observer is allowed to infer with what that observer has seen. It is a healthiness condition which means that what can be inferred from an observation must be consistent with the original observation. If confidentiality statements allowed functors which gave results with no arrow back to the observation then no system would be able to satisfy such a confidentiality statement since, as shall be seen, this is a property of systems. In the same way the naturality condition ensures that the dependencies between observations are not only preserved by the functor F, but also remain consistent with the original dependencies.

Example 6 Later in this chapter it will be shown that there is a preorder category called Safety-Iflow which satisfies the postulates for the abstract category $\mathcal{IF}$. Functors from Safety-Iflow to itself are monotonic functions. Continuing example 5, if a transaction of customer 2 is defined by

$$TCUST_2 \triangleq \{ \langle Pin_2, amount_2?n, dispense_2!n \rangle \mid n \in \mathbb{N} \}$$

then one functor from $\mathcal{IF}$ to $\mathcal{IF}$ is

$$\forall l : CUST_1^* \cdot$$
$$F(CUST_1, \{l\}) = \left(\alpha ATM, \left\{ tr : \alpha ATM^* \;\middle|\; \begin{array}{l} tr \upharpoonright CUST_1 = l \\ \wedge \\ tr \upharpoonright CUST_2 \in (\frown / TCUST_2^*) \end{array} \right\} \right)$$

where $\frown /$ is the operator which flattens a sequence of sequences. An example of a local observation is,

$$(CUST_1, \langle pin_1, amount_1?5, dispense_1!5 \rangle),$$

the set $CUST_1$ sets the context in, or the "window" through, which the observation was made. The function F ranges over all the potential observations that can be made through $CUST_1$. The result of applying F to a local observation is a pair which when restricted to the events in $CUST_1$ is equal to the local observation. Thus restriction forms the basis for the arrows of the natural transformation to the identity function over Safety-Iflow, which is the condition necessary for a confidentiality statement. If F is interpreted informally then F places a limit on what can be inferred from a local observation. It says that from an observation which customer 1 could make through $CUST_1$ the most that can be inferred is that zero or more transactions of customer 2 could have occurred. $\triangle$

A confidentiality statement abstracts the concept of a security specification defined in [Jacob 88a]. Another example of confidentiality statements over a simple $\mathcal{IF}$ category is given.

Example 7 Let $\mathcal{IF}$ be the lattice $\{\bot, a, \top\}$, functors from $\mathcal{IF}$ to $\mathcal{IF}$ are monotonic functions. The following functors give rise to confidentiality statements.

$$\begin{array}{lcll} \mathcal{S}_1 & = & \{\bot \mapsto \bot,\ a \mapsto \bot,\ \top \mapsto \bot\} & \\ \mathcal{S}_2 & = & \{\bot \mapsto \bot,\ a \mapsto \bot,\ \top \mapsto a\} & \\ \mathcal{S}_3 & = & \{\bot \mapsto \bot,\ a \mapsto \bot,\ \top \mapsto \top\} & \\ \mathcal{S}_{3'} & = & \{\bot \mapsto \bot,\ a \mapsto a,\ \top \mapsto a\} & \\ \mathcal{S}_4 & = & \{\bot \mapsto \bot,\ a \mapsto a,\ \top \mapsto \top\} & (= Id_{\mathcal{IF}}) \end{array}$$

The functors $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_{3'}$ and $\mathcal{S}_4$ all have natural transformations to the identity functor. To convince the reader that this is so the arrow, $\bot \rightarrow a$, will be examined to see how it is transformed under $\mathcal{S}_3$.

For convenience let f be the name of the arrow $\bot \rightarrow a$. For θ to be a natural transformation it must be shown that

$$\mathcal{S}_3(f); \theta_a = \theta_\bot; Id_{\mathcal{IF}}(f)$$

Under $\mathcal{S}_3$ the arrow f becomes the identity arrow of $\bot$, $id_\bot$. Let $\theta_a = f$ and $\theta_\bot = id_\bot$ then

$$
\begin{array}{rcll}
\mathcal{S}_3(f);\theta_a & = & id_\perp;\theta_a & [\text{by defn. of } \mathcal{S}_3(f)] \\
& = & id_\perp; f & [\text{by defn. of } \theta_a] \\
& = & \theta_\perp; Id_{\mathcal{IF}}(f) & [\text{by defn. of } \theta_\perp \text{ and } Id_{\mathcal{IF}}]
\end{array}
$$

The other arrows in $\mathcal{IF}$, i.e. $a \rightarrow \top$, $\perp \rightarrow \top$ and the identity arrows, can be similarly checked. The functors of the confidentiality statements map the objects in the lattice to objects which are less than or equal to themselves, hence there will always exist an arrow back to the original object.

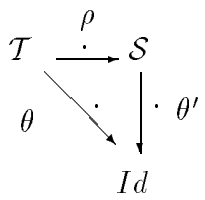
The following functors do not give rise to confidentiality statements because there are no natural transformations to the identity functor.

$$
\begin{array}{c}
\{\perp \mapsto a,\ a \mapsto a,\ \top \mapsto \top\} \\
\{\perp \mapsto \top,\ a \mapsto \top,\ \top \mapsto \top\} \\
\{\perp \mapsto \perp,\ a \mapsto \top,\ \top \mapsto \top\} \\
\{\perp \mapsto a,\ a \mapsto a,\ \top \mapsto a\}
\end{array}
$$

$\triangle$

To formalize the notion of limiting inference an ordering needs to be defined. A notion of one confidentiality statement being no worse than another is formalized by the following definition.

Definition 34 *Let $\mathcal{S}$ and $\mathcal{T}$ be two confidentiality statements. $\mathcal{S}$ is at least as secure as $\mathcal{T}$, written $\mathcal{S} \trianglerighteq \mathcal{T}$, if there exists a natural transformation, ρ, from $\mathcal{T}$ to $\mathcal{S}$ and the following diagram commutes*



$\diamond$

The natural transformation from $\mathcal{T}$ to $\mathcal{S}$ captures the concept of "granularity of structure", the image of an observation under the functor $\mathcal{T}$ has less granularity than under $\mathcal{S}$. The greater the granularity the greater the uncertainty in the inference from an observation, hence $\mathcal{S}$ is more secure than $\mathcal{T}$. The commuting diagram guarantees the basic consistency of comparing confidentiality statements. Thanks go to Joseph Goguen for help in formulating definitions 33 and 34 correctly.

Example 8 G is another function over the local observations that can be made of an ATM.

$$\forall l : CUST_1^* \cdot$$
$$G(CUST_1, \{l\}) = \left(\alpha ATM, \left\{ tr : \alpha ATM^* \;\middle|\; \begin{array}{l} tr \upharpoonright CUST_1 = l \\ \wedge \\ tr \upharpoonright CUST_2 \in CUST_2^* \end{array} \right\} \right)$$

The function G ranges over all the potential observations that can be made through $CUST_1$. The result of applying G to a local observation is a pair which when restricted to the events in $CUST_1$ is equal to the local observation. For example

$$G(CUST_1, \{\langle pin_1 \rangle\}) = \left(\alpha ATM, \left\{ tr : \alpha ATM \;\middle|\; \begin{array}{l} tr \upharpoonright CUST_1 = \langle pin_1 \rangle \\ \wedge \\ tr \upharpoonright CUST_2 \in CUST_2^* \end{array} \right\} \right)$$

which by definition of the result is equal to $\{pin_1\}$ when restricted back through the window $CUST_1$. If G is interpreted informally then it says that a potential observer at $CUST_1$ is limited to inferring that some interaction by customer 2 might have occurred but there is no way of determining what events might have occurred. This is obviously too strong for an ATM, for example

$$\langle dispense_1 ! n \rangle$$

would be a behaviour required by G but would not be allowed by any reasonable functional requirement; unless one wants an ATM to throw money at you before you've asked for it! For the purposes of illustrating the ordering relation this conflict with functional requirements is ignored. Comparing the function F of example 6 with G it can be seen that

$$G \;\trianglerighteq\; F$$

since

$$\begin{array}{l} \forall l : CUST_1^* \cdot F(CUST_1, \{l\}) = \\ (\alpha ATM, \{\, tr : \alpha ATM^* \;\mid\; tr \upharpoonright CUST_1 = l \wedge tr \upharpoonright CUST_2 \in CUST_2^* \,\}) \end{array}$$

and

$$\begin{array}{l} \{\, tr : \alpha ATM^* \;\mid\; tr \upharpoonright CUST_1 = l \wedge tr \upharpoonright CUST_2 \in (\frown / TCUST)_2^* \,\} \\ \subseteq \\ \{\, tr : \alpha ATM^* \;\mid\; tr \upharpoonright CUST_1 = l \wedge tr \upharpoonright CUST_2 \in CUST_2^* \,\} \end{array}$$

therefore F allows a potential observer to infer more that G. △

Example 9 Using the same confidentiality statements from example 7 and applying definition 34 the confidentiality statements can be compared.

$$\begin{array}{lcl} \mathcal{S}_4 & \trianglerighteq & \mathcal{S}_3 \\ \mathcal{S}_4 & \trianglerighteq & \mathcal{S}_{3'} \\ \mathcal{S}_3 & \trianglerighteq & \mathcal{S}_2 \\ \mathcal{S}_{3'} & \trianglerighteq & \mathcal{S}_2 \\ \mathcal{S}_2 & \trianglerighteq & \mathcal{S}_1 \end{array}$$

Note that there is no natural transformation $\mathcal{S}_3 \stackrel{\bullet}{\rightarrow} \mathcal{S}_{3'}$ or $\mathcal{S}_{3'} \stackrel{\bullet}{\rightarrow} \mathcal{S}_3$. △

Using the properties of natural transformations the next lemma begins to show how a partial order can be imposed over the set of confidentiality statements.

Lemma 6 *The order relation* $\trianglerighteq$ *over confidentiality statements is a preorder*

Proof

The proof that $\trianglerighteq$ is reflexive is trivial when it is noted that the identity arrow on objects in the category provide the components of a natural transformation from a functor to itself. The identity arrows also ensure that the diagram of definition 34 commutes.

The proof of transitivity is also simple with the additional fact that natural transformations compose to give another natural transformation. The commutative property of definition 34 comes from substitution. If $\mathcal{T} \trianglerighteq \mathcal{V}$ then there is a

$$\rho_1 : \mathcal{V} \xrightarrow{\bullet} \mathcal{T} \quad \text{such that} \quad \rho_1; \theta_T = \theta_\mathcal{V}.$$

If $\mathcal{S} \trianglerighteq \mathcal{T}$ then there is a

$$\rho_2 : \mathcal{T} \xrightarrow{\bullet} \mathcal{S} \quad \text{such that} \quad \rho_2; \theta_\mathcal{S} = \theta_T.$$

Substituting $\rho_2; \theta_\mathcal{S}$ for θ_T in $\rho_1; \theta_T = \theta_\mathcal{V}$ gives

$$\rho_1; \rho_2; \theta_\mathcal{S} = \theta_\mathcal{V}.$$

This gives a composite natural transformation $(\rho_1; \rho_2)$ which satisfies the commuting diagram condition for $\mathcal{S} \trianglerighteq \mathcal{V}$. □

Any preorder can be viewed as a category, so the preorder of confidentiality statements gives rise in turn to a skeletal category where isomorphic objects are considered to be equal. In the case of a skeletal preorder

$$x = y \iff (x \geq y \wedge y \geq x)$$

i.e. a skeletal preorder is a partial order.

Definition 35 (Iflow-order) *The skeletal preorder category of confidentiality statements is denoted* Iflow-order *and* $\trianglerighteq$ *is used to denote the ordering.* ◇

The overloading of $\trianglerighteq$ will cause no confusion as only Iflow-order will be discussed for the rest of the thesis.

3.2.1 The structure of Iflow-order

In this section Iflow-order will be shown to be a complete distributive lattice. A complete lattice is a partial order with a top and a bottom where every set of elements in the partial order has a greatest lower bound and a least upper bound. A lattice is distributive if the greatest lower bound operator distributes over the least upper bound operator.

By definition 34 the identity functor is greater than any other confidentiality statement: it allows no more information to be inferred than that contained in the observation made.

Lemma 7 *For every confidentiality statement, $\mathcal{T}$, $Id_{\mathcal{IF}} \trianglerighteq \mathcal{T}$*

Proof

Follows directly from definition 34 and the natural transformation from $Id_{\mathcal{IF}}$ to itself provided by the identity arrows of a category, this also guarantees that the diagram of definition 34 will commute. □

This lemma shows that the identity functor is the top for the desired lattice. For notational convenience the following definition is made.

Notation 1 *The unique arrow from the initial object* **0** *to any other object c in $\mathcal{IF}$ is denoted* $!c$.

Example 9 gave one instance of a confidentiality statement, $\mathcal{S}_1$, which was weaker than the other confidentiality statements. $\mathcal{S}_1$ mapped every element in the lattice of example 7 to bottom, the initial object. Obviously the functor which maps every element to an initial object has a natural transformation to every other confidentiality statement. This is proved in the next lemma.

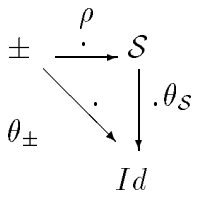
Lemma 8 *The confidentiality statement which maps every object to an initial object,* **0**, *of $\mathcal{IF}$ is the bottom element of the order $\trianglerighteq$.*

Proof

If $\pm$ denotes the functor which maps every object to **0**, $f : a \rightarrow b$ is any arrow in $\mathcal{IF}$ and F is any functor from $\mathcal{IF}$ to itself then

$$\begin{array}{lcll} \pm(f); !F(b) & = & id_{\mathbf{0}}; !F(b) & \text{[By defn. of } \pm\text{]} \\ & = & !F(a); F(f) & \text{[By uniqueness of } !F(b)\text{]} \end{array}$$

This shows that there is a natural transformation from $\pm$ to every other functor of $\mathcal{IF}$. The commutative diagram of definition 34 holds for $\pm$ because composing the arrows of the natural transformation ρ with the arrows of the natural transformation, $\theta_{\mathcal{S}}$, to the identity gives another natural transformation consisting of arrows from an initial object to the identity. But arrows from the initial object are unique therefore they must be the same as the arrows in the natural transformation, $\theta_{\pm}$, from $\pm$ to the identity. Hence the diagram



must commute and by definition of $\trianglerighteq$, $\pm$ is indeed the bottom of Iflow-order. □

To reason about conjunction and disjunction in a calculus the following pieces of notation are needed.

Notation 2 *The product of the results of evaluating a set of confidentiality statements E is denoted*

$$\bigotimes E$$

When only two confidentiality statements $\mathcal{S}$ and $\mathcal{T}$ are involved, the product of the results of $\mathcal{S}$ and $\mathcal{T}$ is denoted by an infix operator

$$\mathcal{S} \otimes \mathcal{T}.$$

The coproduct of the results of evaluating a set of confidentiality statements E is denoted

$$\bigoplus E$$

for two confidentiality statements the coproduct of the results of $\mathcal{S}$ and $\mathcal{T}$ is denoted by an infix operator

$$\mathcal{S} \oplus \mathcal{T}.$$

Example 10 If $\mathcal{S}$ and $\mathcal{T}$ are confidentiality statements then

$$\forall x : \mathcal{IF} \cdot (\mathcal{S} \otimes \mathcal{T})(x) = \mathcal{S}(x) \times \mathcal{T}(x)$$

and

$$\forall x : \mathcal{IF} \cdot (\mathcal{S} \otimes \mathcal{T})(x) = \mathcal{S}(x) + \mathcal{T}(x)$$

△

Lemma 9 *In Iflow-Order $\bigotimes$ and $\bigoplus$ are greatest lower bound and least upper bound respectively.*

Proof

If $\mathcal{S}$ and $\mathcal{T}$ are confidentiality statements then

$$\forall x : \mathcal{IF} \cdot \mathcal{S}(x) \times \mathcal{T}(x) \to \mathcal{S}(x)$$

(via the projection arrows to $\mathcal{S}(x)$) which gives a natural transformation

$$\rho : \mathcal{S} \otimes \mathcal{T} \stackrel{\bullet}{\to} \mathcal{S}.$$

By the definition of a confidentiality statement there is a natural transformation

$$\theta_{\mathcal{S}} : \mathcal{S} \stackrel{\bullet}{\to} \mathrm{Id}_{\mathcal{IF}}$$

therefore by composing the arrows of the projection with the arrows of the natural transformation $\theta_{\mathcal{S}}$ there is a natural transformation θ' which is equal to $\rho; \theta_{\mathcal{S}}$ by construction. This means that $\mathcal{S} \otimes \mathcal{T}$ is a confidentiality statement such that $\mathcal{S} \otimes \mathcal{T} \trianglerighteq \mathcal{S}$. Similarly it can be shown that $\mathcal{S} \otimes \mathcal{T} \trianglerighteq \mathcal{T}$.

If H is another confidentiality statement such that

$$H \trianglerighteq \mathcal{S} \quad \wedge \quad H \trianglerighteq \mathcal{T}$$

then by definition of product H must have a natural transformation to $\mathcal{S} \otimes \mathcal{T}$, thus $\mathcal{S} \otimes \mathcal{T}$ is the greatest lower bound of $\mathcal{S}$ and $\mathcal{T}$ in Iflow-Order. The dual of this argument shows that $\mathcal{S} + \mathcal{T}$ is the least upper bound of $\mathcal{S}$ and $\mathcal{T}$.

In general for some indexing set I the greatest lower bound and least upper bound exist for any set of confidentiality statements indexed by I, since by postulate 1 all small products and coproducts are assumed to exist in $\mathcal{IF}$. □

The next theorem provides the basis for the formal meaning of implication and negation in CS.

Theorem 1 *Iflow-order is a complete lattice.*

Proof By lemmas 7, 8 and 9 Iflow-Order has a top, bottom, a greatest lower bound and least upper bound for every set of confidentiality statements.

A simpler proof, due to Joseph Goguen, is to note that Iflow-order is just a skeletal category of a comma category over the endofunctor category of a complete and cocomplete category. □

The formal definition of implication uses the notion of relative pseudo complement which exists in relative pseudo-complemented lattices.

Definition 36 (Relative pseudo-complement) *In a preorder, S, with $\otimes$, the relative pseudo-complement $(b \Longrightarrow c)$ is defined up to equivalence by the axiom*

$$\forall x : S \cdot x \leq (b \implies c) \iff (x \otimes b) \leq c$$

◇

Proposition 1 *Iflow-order is a distributive lattice*

Proof This follows from postulate 2 which requires product to distribute over coproduct. □

Since Iflow-order is a complete distributive lattice a more convenient formulation of the relative pseudo-complement can be constructed.

Lemma 10 *In a complete distributive lattice, S, the relative pseudo-complement*

$$(b \Longrightarrow c) = \bigoplus \{\, x \in S \mid x \otimes b \leq c \,\}$$

i.e. the least upper bound of those x whose greatest lower bound with b is less than or equal to c.

Proof

(Case 1: Suppose $y \leq \bigoplus\{ x \in S \mid x \otimes b \leq c \}$)

$$\bigoplus\{ x \otimes b \mid x \in S \wedge x \otimes b \leq c \} = (\bigoplus\{ x \in S \mid x \otimes b \leq c \}) \otimes b$$
[by distributivity.]
$$\Longrightarrow \quad \bigoplus\{ x \in S \mid x \otimes b \leq c \} \otimes b \leq c$$
[since c is an upper-bound of $\bigoplus\{ x \otimes b \mid x \in S \wedge x \otimes b \leq c \}$]
$$\Longrightarrow \quad y \otimes b \leq c \quad \text{[By transitivity]}$$

(Case 2: Suppose $y \otimes b \leq c$) Under this assumption

$$y \leq \bigoplus\{ x \in S \mid x \otimes b \leq c \}$$

follows by definition of least upper bound.

By definition 36

$$\bigoplus\{ x \in S \mid x \otimes b \leq c \}$$

satisfies the condition for the relative pseudo-complement. □

This lemma allows the statement that the inference of one observer implies the inference of another observer to be expressed in the calculus presented in the next chapter. The final operation which needs to be defined is the pseudo-complement which will give the formal meaning to negation in the calculus.

Definition 37 (pseudo-complement) *For any object p in a complete lattice,*

$$\sim p \triangleq (p \Longrightarrow \bot)$$

or equivalently

$$\sim p \triangleq \bigoplus\{ x \in S \mid x \otimes p = \bot \}$$

◇

Theorem 2 *Iflow-order with the operators*

$$\otimes \ , \ \oplus \ , \Longrightarrow \textit{and} \ \sim$$

is a Heyting algebra

Proof By lemmas 9, 8, 10 and definition 37. □

Lemma 11 *Iflow-order with the operators*

$$\otimes \ , \ \oplus \ , \Longrightarrow \textit{and} \ \sim$$

is not a boolean algebra

Proof

In a boolean algebra the least upper bound of an element a with its negation equals the top of the lattice,

$$a \oplus \sim a = \top$$

but not in Iflow-order with its operators. As a counter example let $\mathcal{IF}$ be as defined in example 7. Using the definition of $\sim$,

$$\sim \mathcal{S}_{3'} = \mathcal{S}_1$$

which is the bottom confidentiality statement. By definition of $\oplus$

$$\mathcal{S}_{3'} \oplus \sim \mathcal{S}_{3'} = \mathcal{S}_{3'}$$

therefore

$$\mathcal{S}_{3'} \oplus \sim \mathcal{S}_{3'} \neq \top$$

□

In a Heyting algebra all the theorems of the intuitionistic propositional calculus can be proved and any model for a Heyting algebra is a model for Heyting's calculus. To get to grips with this abstract model of information flow it would be useful to have a concrete instance. The next section provides a concrete example of an information flow category based on sets of traces.

3.3 A concrete category for traces

A preorder category over sets of traces is introduced and used to give a concrete instance of the abstract Iflow category. The preorder also provides a suitable semantic space for the trace model of CSP. First the operator of restriction over traces, defined in [Hoare 85], is extended to sets of traces.

Definition 38 *For any set of traces T and alphabet A,*

$$T \upharpoonright A \triangleq \{ t \upharpoonright A \mid t \in T \}$$

◇

Example 11 If $T = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, b, b\rangle\}$ then

$$T \upharpoonright \{a\} = \{\langle\rangle, \langle a\rangle\}$$

△

The concrete instance of the category $\mathcal{IF}$ is defined next.

Definition 39 (Safety-Iflow) *The objects of Safety-Iflow are all pairs of the form:*

$$(A, T)$$

where A is any subset of an alphabet Σ and T is any subset of A^*, $A = \{\}$ only when $T = \{\langle\rangle\}$.

A morphism exists from (A, T) to (B, U) if and only if

$$B \subseteq A \wedge T \upharpoonright B \subseteq U \text{ and is denoted } (A, T) \preceq (B, U)$$

$\diamond$

Safety-Iflow allows objects (A, T) such that T is not prefix closed and A or T can be empty.

Example 12 If

$$(A, T) = (\{a, b, c\}, \{\langle a, b\rangle, \langle a, c, b\rangle\})$$

and

$$(B, S) = (\{a, b\}, \{\langle a, b\rangle, \langle b, a\rangle\})$$

then there is a morphism from (A, T) to (B, S) since

$$(B, T \upharpoonright B) = (\{a, b\}, \{\langle a, b\rangle\})$$

$\triangle$

The following lemma shows that Safety-Iflow is a preorder category, that is there is only one arrow between different objects, and in particular is a partial order.

Lemma 12 *Safety-Iflow is a partial order*

Proof

Follows from the properties of $\upharpoonright$ and $\subseteq$. $\square$

Considering a partial order as a preorder category means that least upper bound of a set of objects is the coproduct of those objects and the greatest lower bound is the product of those objects.

Example 13 Suppose there are two observers of a system: observer A records $(\{a, b\}, \{\langle\rangle, \langle a\rangle\})$ and observer B records $(\{a, c\}, \{\langle a\rangle, \langle a, a\rangle\})$. Observer A can infer that either nothing has happened or the event 'a' has occurred, but no 'b' events have occurred. Observer B can infer that either the event 'a' has occurred once or the event 'a' has occurred twice, but no 'c' events have occurred.

Pooling the observers knowledge together means finding the strongest inference which is most consistent with their individual observations, that is the categorical product:

$$(\{a, b\}, \{\langle\rangle, \langle a\rangle\}) \times (\{a, c\}, \{\langle a\rangle, \langle a, a\rangle\})$$

or

$$(\{a, b, c\}, \{\langle a \rangle\})$$

the collaborating observers can be certain in their inference that only the single event 'a' has occurred.

$\triangle$

Notation for least upper bound and greatest lower bound is introduced.

Notation 3 *If X is any collection of sets in a preorder then*

- $\bigvee X$ *denotes the least upper bound of* X
- $\bigwedge X$ *denotes the greatest lower bound of* X

The next lemma shows that Safety-Iflow is a complete lattice.

Lemma 13 *The greatest lower bound and least upper bound of a non-empty collection of objects X in Safety-Iflow are*

$$\bigwedge X \triangleq (V, \bigcap_{(A,T):X} \{ t : V^* \mid t \restriction A \in T \})$$

$$\bigvee X \triangleq (Z, \bigcup_{(A,T):X} T \restriction Z)$$

where

$$V \triangleq \bigcup_{(A,T):X} A \quad \text{and} \quad Z \triangleq \bigcap_{(A,T):X} A.$$

Proof

The proof is by structural induction over the set of objects in Safety-Iflow. We start by looking at the greatest lower bound for a set of objects. There are two base cases:

Base Case 1

$$\bigwedge \{(A, T)\} = (A, T)$$

Base Case 2

$$\begin{aligned}
&\bigwedge \{(A, T), (B, S)\} = (A \cup B, \{ t : (A \cup B)^* \mid t \restriction A \in T \wedge t \restriction B \in S \}) \\
&= (A \cup B, \{ t : (A \cup B)^* \mid t \restriction A \in T \} \cap \{ t : (A \cup B)^* \mid t \restriction B \in S \}) \\
&= (A \cup B, \bigcap_{(C,R):\{(A,T),(B,S)\}} \{ t : (A \cup B)^* \mid t \restriction C \in R \})
\end{aligned}$$

Induction Step

$$\bigwedge (Y \cup \{(B,S)\}) = \bigwedge Y \times (B,S)$$
$$= (V, \bigcap_{(C,R):Y} \{\, t : V^* \mid t \restriction C \in R \,\}) \times (B,S)$$
[By induction hypothesis]

$$= (V \cup B, \{\, s : (V \cup B)^* \mid s \restriction V \in \bigcap_{(C,R):Y} \{\, t : V^* \mid t \restriction C \in R \,\} \wedge s \restriction B \in S \,\})$$
$$= \left(\begin{array}{ll} (V \cup B, & \{\, s : (V \cup B)^* \mid s \restriction V \in \bigcap_{(C,R):Y} \{\, t : V^* \mid t \restriction C \in R \,\} \,\} \\ & \cap \\ & \{\, s : (V \cup B)^* \mid s \restriction B \in S \,\}) \end{array} \right)$$
$$= \left(\begin{array}{ll} (V \cup B, & \bigcap_{(C,R):Y} \{\, t : (V \cup B)^* \mid t \restriction C \in R \,\} \\ & \cap \\ & \{\, s : (V \cup B)^* \mid s \restriction B \in S \,\} \end{array} \right)$$
$$= (V \cup B, \bigcap_{(C,R):Y \cup \{(B,S)\}} \{\, t : (V \cup B)^* \mid t \restriction C \in R \,\})$$

A similar induction argument demonstrates the second part of the lemma. There are two base cases:

Base Case 1

$$\bigvee \{(A,T)\} = (A,T)$$

Base Case 2

$$\bigvee \{(A,T),(B,S)\} = (A \cap B, \{\, t : (A \cap B)^* \mid t \in T \vee t \in S \,\})$$
$$= (A \cap B, \{\, t : (A \cap B)^* \mid t \in T \,\} \cup \{\, t : (A \cap B)^* \mid t \in S \,\})$$
$$= (A \cap B, T \restriction (A \cap B) \cup S \restriction (A \cap B))$$
$$= (A \cap B, \bigcup_{(C,R) \in \{(A,T),(B,S)\}} R \restriction (A \cap B))$$

Induction Step

$$\bigvee (Y \cup \{(B,S)\}) = \bigvee Y + (B,S)$$
$$= (Z, \bigcup_{(C,R):Y} R \restriction Z) + (B,S)$$
$$= (Z \cap B, \{\, t : (Z \cap B)^* \mid t \in \bigcup_{(C,R):Y} R \restriction Z \vee t \in S \,\})$$
$$= (Z \cap B, \{\, t : (Z \cap B)^* \mid t \in \bigcup_{(C,R):Y} R \restriction Z \,\} \cup \{\, t : (Z \cap B)^* \mid t \in S \,\})$$
$$= (Z \cap B, \bigcup_{(C,R):Y} R \restriction (Z \cap B) \cup S \restriction (Z \cap B))$$
$$= (Z \cap B, \bigcup_{(C,R):(Y \cup \{(B,S)\})} R \restriction (Z \cap B)$$

□

Example 14 Let

$$X = \{(\{a\}, \{\langle a \rangle\}), (\{a, b\}, \{\langle a, b \rangle\})\}$$

therefore

$$V = \{a, b\} \text{ and } Z = \{a\}.$$

$$\begin{aligned} \bigwedge X &= (V, \{\, t : V^* \mid t \restriction \{a\} \in \{\langle a \rangle\} \,\} \cap \{\, t : V^* \mid t \restriction \{a, b\} \in \{\langle a, b \rangle\} \,\}) \\ &= (\{a, b\}, \{\, t^\frown\langle a \rangle^\frown s \mid t, s \in \{b\}^* \,\} \cap \{\langle a, b \rangle\}) \\ &= (\{a, b\}, \{\langle a, b \rangle\}). \end{aligned}$$

$$\begin{aligned} \bigvee X &= (Z, \{\langle a \rangle\} \restriction Z \cup \{\langle a, b \rangle\} \restriction Z) \\ &= (\{a\}, \{\langle a \rangle\} \cup \{\langle a \rangle\}) \\ &= (\{a\}, \{\langle a \rangle\}). \end{aligned}$$

△

The next few lemmas and propositions show that Safety-Iflow is an Iflow category.

Lemma 14 *Every set of objects in Safety-Iflow has a product and coproduct.*

Proof

Product and coproduct correspond to the greatest lower and least upper bound respectively in a partial order. Any collection of objects is guaranteed a product and a coproduct by Safety-Iflow being a complete lattice. □

This lemma shows that postulate 1 is satisfied by Safety-Iflow. The particular form of the initial object is given by the next lemma.

Lemma 15 *The initial and terminal objects of Safety-Iflow are* $(\Sigma, \{\})$ *and* $(\{\}, \{\langle\rangle\})$ *respectively.*

Proof

Direct from definition 39. □

Sub-lattices of Safety-Iflow which form cartesian closed subcategories of Safety-Iflow are now examined.

Definition 40

$$\textit{Safe-Iflow}_A \triangleq \{\, (A, T) \mid (A, T) \in \text{Safety-Iflow} \,\}$$

◇

That is Safe-Iflow$_A$ is the set of elements of Safety-Iflow with a fixed alphabet, A. The following lemma shows that the monotonic function $[(A, S) \times \_]$ preserves colimits in the subcategory Safe-Iflow$_A$.

Lemma 16 *For each $A \subseteq \Sigma$, if $(A, S) \in$ Safe-Iflow$_A$ then $[(A, S) \times \_]$ preserves least upper bounds*

Proof

$$\begin{aligned}
[(A,S) \times _](A, \bigcup_{(A,T):X} T) &= (A, \{\, t : A^* \mid t \in S \wedge t \in \bigcup_{(A,T):X} T \,\}) \\
&= (A, \bigcup_{(A,T):X} \{\, t : A^* \mid t \in S \wedge t \in T \,\}) \\
&= \bigcup_{(A,T):X} [(A,S) \times _](A,T)
\end{aligned}$$

□

The concrete information flow categories can now be characterized.

Proposition 2 *For each $A \subseteq \Sigma$, Safe-Iflow$_A$ is an Iflow category*

Proof By lemma 14 postulate 1 holds. By definition of bottom in a partial order, the greatest lower bound (or product) of any other element with bottom is bottom. By lemma 16 postulate 2 holds. □

This proposition shows that by fixing the alphabet for a set of sets of traces all the operators for a Heyting algebra can be given a meaning. The abstract and concrete model are used in the next chapter to give a semantic space for an abstract calculus of information flow and a semantic space for a particular calculus of information flow over traces.

Chapter 4

A calculus of Information Flow

In this chapter the semantic models of chapter 3 are used to give a meaning to two calculi. The first is an abstract calculus which is based purely on the categorical semantics. The second calculus is a concrete instance of the abstract calculus. This calculus is based on the concrete trace model of chapter 3 and satisfies the laws that CS does plus some which are peculiar to the trace model of computation. The final part of the chapter gives examples of how the concrete calculus can be used to specify confidentiality requirements.

4.1 CS an abstract calculus

The syntax of CS

If S and T range over syntactic propositions, then the syntax of CS is defined as:

$$S ::= CStmt \mid \bot \mid \top \mid (S \wedge_{\text{IF}} T) \mid (S \vee_{\text{IF}} T) \mid (S \implies_{\text{IF}} T) \mid \neg_{\text{IF}} S$$

The semantics of CS

In the definition of the semantics of CS below lambda abstractions are used to denote anonymous functions.

If

$\mathcal{M} : CS \rightarrow$ Iflow-order	is the semantic function,
$\mathcal{S}$	ranges over confidentiality statements,
S and T	range over syntactic propositions and
$\mathcal{M}(\mathcal{S})$	is defined in a particular concrete category

then

$$\begin{aligned} \mathcal{M}(\bot) &= (\lambda x : \mathcal{IF} \cdot \mathbf{0}) \\ \mathcal{M}(\top) &= Id_{\mathcal{IF}} \\ \mathcal{M}(S \wedge_{\text{IF}} T) &= \mathcal{M}(S) \otimes \mathcal{M}(T) \end{aligned}$$

$$
\begin{aligned}
\mathcal{M}(S \vee_{\mathrm{IF}} T) &= \mathcal{M}(S) \oplus \mathcal{M}(T) \\
\mathcal{M}(S \Longrightarrow_{\mathrm{IF}} T) &= \bigoplus\{\, x \in \text{Iflow-order} \mid \mathcal{M}(T) \trianglerighteq x \otimes \mathcal{M}(S) \,\} \\
\mathcal{M}(\neg_{\mathrm{IF}} S) &= \bigoplus\{\, x \in \text{Iflow-order} \mid x \otimes \mathcal{M}(S) = \bot \,\}
\end{aligned}
$$

The Logic of CS

The logic of CS is of interest to those who want to conduct formal proofs in the abstract calculus CS or wish to provide tool support for CS or any of its concrete instances. The logic of CS is that of intuitionistic logic, that is classical logic with the law of the excluded middle

$$A \vee \neg A = \mathit{true}$$

removed. The particular axiomatisation given here can be replaced by the reader's favourite axiomatisation of intuitionistic logic. A model for intuitionistic logic is Arend Heyting's algebra. In the previous chapter a categorical model for a Heyting algebra was developed and the semantic definition of CS related it to this model. The axioms and inference rule of Heyting's calculus are sound in a model for a Heyting algebra (see pages 185-186 of [Goldblatt 84]). This means that the axiomatisation of Heyting's calculus will provide a sound axiomatisation of CS with respect to the semantics.

The axiomatisation of CS will not be complete for every model of computation because in general the concrete information flow categories will satisfy assumptions in addition to those satisfied by the abstract information flow category. For example the concrete category over traces when used as a model for a concrete instance of CS would allow at least two more axioms to be added.

Axioms for CS

[I]	$S \Longrightarrow_{\mathrm{IF}} (S \wedge_{\mathrm{IF}} S)$
[II]	$(S \wedge_{\mathrm{IF}} T) \Longrightarrow_{\mathrm{IF}} (T \wedge_{\mathrm{IF}} S)$
[III]	$(S \Longrightarrow_{\mathrm{IF}} T) \Longrightarrow_{\mathrm{IF}} ((S \wedge_{\mathrm{IF}} \mathcal{Z}) \Longrightarrow_{\mathrm{IF}} (T \wedge_{\mathrm{IF}} \mathcal{Z}))$
[IV]	$((S \Longrightarrow_{\mathrm{IF}} T) \wedge_{\mathrm{IF}} (T \Longrightarrow_{\mathrm{IF}} \mathcal{Z})) \Longrightarrow_{\mathrm{IF}} (S \Longrightarrow_{\mathrm{IF}} \mathcal{Z})$
[V]	$T \Longrightarrow_{\mathrm{IF}} (S \Longrightarrow_{\mathrm{IF}} T)$
[VI]	$(S \wedge_{\mathrm{IF}} (S \Longrightarrow_{\mathrm{IF}} T)) \Longrightarrow_{\mathrm{IF}} T$
[VII]	$S \Longrightarrow_{\mathrm{IF}} (S \vee_{\mathrm{IF}} T)$
[VIII]	$(S \vee_{\mathrm{IF}} T) \Longrightarrow_{\mathrm{IF}} (T \vee_{\mathrm{IF}} S)$
[IX]	$((S \Longrightarrow_{\mathrm{IF}} \mathcal{Z}) \wedge_{\mathrm{IF}} (T \Longrightarrow_{\mathrm{IF}} \mathcal{Z})) \Longrightarrow_{\mathrm{IF}} ((S \vee_{\mathrm{IF}} T) \Longrightarrow_{\mathrm{IF}} \mathcal{Z})$
[X]	$\neg_{\mathrm{IF}} S \Longrightarrow_{\mathrm{IF}} (S \Longrightarrow_{\mathrm{IF}} T)$
[XI]	$((S \Longrightarrow_{\mathrm{IF}} T) \wedge_{\mathrm{IF}} (S \Longrightarrow_{\mathrm{IF}} \neg_{\mathrm{IF}} T)) \Longrightarrow_{\mathrm{IF}} \neg_{\mathrm{IF}} S$

There is a single rule of inference:

RULE OF DETACHMENT

$$\frac{S \qquad S \Longrightarrow_{\mathrm{IF}} T}{T}$$

This rule is also known by the name Modus Ponens. It takes a pair of theorems, an implication and its antecedent, and detaches the consequent as a new theorem.

4.2 CST a concrete calculus

In this section a language for specifying confidentiality properties is presented. The basis for the language CST is taken from the semantic model of traces for CSP, [Hoare 85], with some generalizations. The trace model of systems allows confidentiality properties, for systems, to be stated in terms of inferring behaviours over events from an observed sequence of events. Events are atomic actions of systems which are observable by the environment in which a system operates. Systems can be thought of as entities which evolve and communicate with an environment by synchronizing upon a set of such actions. Although systems are not considered explicitly in this chapter, the design and semantics of the calculus CST has been done in order to include the notion of a system satisfying a confidentiality specification.

The full language includes negation and implication but at the price of fixing the alphabet of events over which one reasons. This is a disadvantage when a component of an unknown system is specified since the alphabet to be fixed will also be unknown. Fortunately this is not a real problem, because without a fixed alphabet one still has a powerful language with conjunction and disjunction. These operators allow complex systems to be specified in terms of components with different alphabets. When all the events of interest are known the alphabet can be fixed and implication and negation used.

An abstract syntax of CST

If S and T range over syntactic confidentiality statements, then the abstract syntax of CST is defined as:

$$S ::= CStmt \mid \bot \mid \top \mid (S \wedge_{\mathrm{IF}} T) \mid (S \vee_{\mathrm{IF}} T) \mid (S \Longrightarrow_{\mathrm{IF}} T) \mid \neg_{\mathrm{IF}} S$$

$$CStmt ::= \textbf{From } l : A^* \text{ “ } \mid \text{”} C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^* \mid \textbf{From } l : B^* \text{ “ } \mid \text{”} D \textbf{ MAY\_INFER } P(tr)_1 \textbf{ Over } tr : X'^* \text{ , } CStmt$$

An atomic confidentiality statement of the form

$$\textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

and should be read as

"from each observation l drawn from the set A^* constrained by the predicate C the most that may be inferred is the set of sequences of operations characterized by the predicate $P(tr)$ over the set X^*."

The mathematical meaning of the confidentiality statement is that it is a function. The domain of the function is

$$\{\, l : A^* \mid C \,\}$$

that is, the subset of A^* restricted to where C is true. The range of the function is

$$\{\, tr : X^* \mid P(tr) \,\}$$

that is, the subset of X^* restricted to where P is true. There is only one condition which the function must satisfy: the set X must contain the set A as a subset, since the events which an observer can see must be a subset of the set which the observer can infer. The function describes the maximum amount of inference which is allowed for any candidate system. It does this by specifying the minimum amount of uncertainty about what particular sequence of operations gave rise to an observed behaviour. The predicate $P(tr)$ must be consistent with the observation l in the sense that the set of sequences of operations characterized by $P(tr)$, when restricted to the events of A must all equal l.

Note that if the constraining predicate C is $true$ then

$$\textbf{From } l : A^* \mid true \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

will be written as

$$\textbf{From } l : A^* \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

The set X must contain the set A as a subset since the events which an observer can see must be a subset of the set which the system of interest performs. The predicate describes the maximum amount of inference which can be performed on a candidate system by specifying the minimum amount of uncertainty about what particular behaviour gave rise to an observed behaviour. The predicate P must be consistent with the observation l in the sense that the set of behaviours characterized by P, when restricted to the events of A must all equal l.

4.2.1 The semantics of CST

To give the semantics for CST the concrete category Safety-Iflow, defined in section 3.3 of chapter 3, is used. Using this concrete category the semantic function for an atomic confidentiality statement is given. This can then be plugged into to semantic description of CS to give the semantic description of CST. With the abbreviation

$$S_1 \triangleq \textbf{From } l : A^* \text{ “}\mid\text{”} C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

the semantic function for atomic confidentiality statements over traces can be defined.

$$\mathcal{M}(S_1) = (\lambda x : \{\, (A, \{l\}) \mid l \in \{\, s : A^* \mid C \,\}\,\} \cdot (X, \{\, tr : X^* \mid P(tr) \,\}))$$
$$\mathcal{M}(S_1 \ , \ CStmt) = \mathcal{M}(S_1) \cup \mathcal{M}(CStmt)$$

This semantic function for atomic confidentiality statements can be plugged into the semantic definition of CS to give the full semantics for CST. Defining another semantic function over atomic confidentiality statements in a different model of computation, which is shown to satisfy the assumptions of an Iflow category, will give another calculus of confidentiality. For example if timed traces were used then plugging an appropriate semantic function into CS would give a new calculus, called "CStime" maybe, in which confidentiality statements involving time could be expressed.

4.2.2 Atomic confidentiality statements and laws

A confidentiality statement is over a fixed number of interfaces. In practice a confidentiality statement will be over many interfaces because there will be many interfaces participating in an information system. When more than one interface is involved the allowed inference over each group is stacked, this forms a *single* confidentiality statement over the interfaces. For example the confidentiality statement

From $l : Other^*$ **MAY_INFER** $tr \upharpoonright Other = l$ **Over** $tr : Other^*$
From $l : Unclass^*$ **MAY_INFER** $tr \upharpoonright Unclass = l$ **Over** $tr : (Conf \cup Unclass)^*$

is over two interfaces called $Other$ and $Unclass$. This specification is about a system whose behaviour is taken from $(Other \cup Conf \cup Unclass)^*$. The system can be observed through the interface given by $Unclass^*$ and all the other interfaces are lumped together in $Other^*$. This confidentiality statement deals with a different system from the following

From $l : Unclass^*$ **MAY_INFER** $tr \upharpoonright Unclass = l$ **Over** $tr : (Conf \cup Unclass)^*$

the two are not comparable. The stacked form of an atomic statement represents one confidentiality requirement. Confidentiality requirements usually (if only implicitly) legislate for the entire behaviour of a system, rather than just part of it. For example, denying unclassified users the ability to open classified files usually implies that files cannot be reached in any other way.

Convention

In practice a confidentiality statement over a fixed number of interfaces will be of the form

From $l : Other^*$ **MAY_INFER** $tr = l$ **Over** $tr : Other^*$
From $l : Unclass^*$ **MAY_INFER** $tr \upharpoonright Unclass = l$ **Over** $tr : CU^*$

which says that there are two permitted interfaces to the system; over the interface $Other$ no inference, apart from what has been observed, is allowed; over $Unclass$ the only inference which can be made is that some unclassified operations may have occurred. In this case the convention shall be to omit the parts which forbid any inference. This convention means that the previous confidentiality statement is written as

From $l : Unclass^*$ **MAY_INFER** $tr \upharpoonright Unclass = l$ **Over** $tr : CU^*$

over the two interfaces *Other* and *Unclass*. This will be unambiguous because the number of interfaces will have been fixed. Other atomic statements are bottom and top, and the confidentiality statements can be combined using conjunction, disjunction and implication. These are all described below.

Bottom

The confidentiality statement $\bot$ (bottom) is used to say that we do not care about the confidentiality properties of a system, any system will satisfy $\bot$. The confidentiality statement $\bot$ is equivalent to *false* in classical logic which, considered as an ordinary specification, cannot by satisfied by any system. This oddity arises from the nature of satisfaction for confidentiality properties. A system satisfies a confidentiality statement if, for each common observation, the inference which can be calculated based on that system is less than the inference specified by the confidentiality statement. The relation "less than" means that the set of behaviours inferred from the system *is greater than* the set of behaviours specified by the confidentiality statement; an observer is less certain about what may have caused a particular observation. The confidentiality statement $\bot$ means that for any observation one can infer the empty set of behaviours. For a genuine observation of a system the calculated inference will always be non-empty, therefore the set of behaviours inferred from a system will be greater than the empty set of behaviours specified by $\bot$ and thus any system will satisfy $\bot$. A consequence of this real dichotomy between functional and confidentiality properties is that functional refinement will not in general preserve confidentiality properties. This is discussed in section 6.2.1 of chapter 6.

$$\bot = \textbf{From } l : A^* \textbf{ MAY\_INFER } false \textbf{ Over } tr : X^*$$

Top

The confidentiality statement $\top$, top, is the dual to $\bot$. It is the strongest confidentiality statement it is possible to make, however it is possible to satisfy it. The confidentiality statement $\top$ means that the most that may be inferred from an observation l is that the system did l and nothing else. This means that the observer is allowed to see everything the system does, ultimate privacy comes when there is nothing to keep private!

$$\top = \textbf{From } l : A^* \textbf{ MAY\_INFER } tr = l \textbf{ Over } tr : A^*$$

Conjunction

The conjunction of two confidentiality statements has the usual meaning of requiring a property A and a property B. In CST it means that one may infer A and B, so that the conjoined inference is greater than either A or B. This means that the conjunction is weaker than its components because it allows more to be inferred. This meaning for conjunction addresses the current problems of building secure systems. Usually the security requirements are too strong and they impair the functionality of the system.

Normally it is the functionality which is the most important requirement. Conjoining the requirements together gives the possibility of a weakening. If a weakening does occur then there is some conflict between requirements. Analysis of the conjoined requirement can reveal this weakening which can then be interpreted to see if it is actually significant. A weaker confidentiality requirement makes it more likely that a system can be built which satisfies its functionality and confidentiality requirements.

Conjunction is the key operator for writing confidentiality specifications. It allows a number of simple atomic confidentiality to be written and then conjoined to give an overall confidentiality specification.

Conjunction in CST is only defined for confidentiality statements over the same set of observations. This is no real problem because a confidentiality statement can always be extended to cover new observations.

Lemma 17 *If*

$$\mathcal{S} \triangleq \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

and

$$\mathcal{T} \triangleq \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P'(tr) \textbf{ Over } tr : Y^*$$

are confidentiality statements then

$$\begin{array}{l} \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T} \\ = \\ \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \restriction X \wedge P'(tr) \restriction Y \textbf{ Over } tr : (X \cup Y)^* \end{array}$$

Proof

By the semantic definition of $\wedge_{\mathrm{IF}}$

$$\mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T} = \mathcal{M}(\mathcal{S}) \otimes \mathcal{M}(\mathcal{T})$$

In Safety-Iflow $\otimes$ is defined as the conjunction of predicates with respect to the alphabets X and Y. Applying this definition of $\otimes$ gives

$$\begin{array}{l} \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T} \\ = \\ \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \restriction X \wedge P'(tr) \restriction Y \textbf{ Over } tr : (X \cup Y)^* \end{array}$$

□

Notice that the conjunction between two confidentiality statements becomes a conjunction between two predicates, suitably restricted, over $(X \cup Y)^*$.

Disjunction

Disjunction is the dual of conjunction and has a dual law.

Lemma 18 *If*

$$\mathcal{S} \triangleq \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

and

$$\mathcal{T} \triangleq \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P'(tr) \textbf{ Over } tr : Y^*$$

are confidentiality statements then

$$\begin{array}{l} \mathcal{S} \vee_{\mathrm{IF}} \mathcal{T} \\ = \\ \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \upharpoonright X \vee P'(tr) \upharpoonright Y \textbf{ Over } tr : (X \cap Y)^* \end{array}$$

Proof

By the same argument as lemma 17 but using the semantic definition of $\vee_{\mathrm{IF}}$. □

Implication

The implication operator is used for reasoning about confidentiality specifications rather than being used as part of a confidentiality specification. This is no different from the use of implication in functional specifications. If

$$(\mathcal{S}_1 \Longrightarrow_{\mathrm{IF}} \mathcal{S}_2) = \top$$

that is $\mathcal{S}_1 \Longrightarrow_{\mathrm{IF}} \mathcal{S}_2$ is true in the calculus CST, then the confidentiality statement $\mathcal{S}_2$ is better than the confidentiality statement $\mathcal{S}_1$. This differs from the use of implication in functional specifications where $A \Longrightarrow B$ means that A is better than B! This oddity is related to that noted in the discussion of $\bot$. It is the notion of "better than" for confidentiality which is different from the notion of "better than" for functionality. With this in mind it is not surprising if everything is better than bottom, that is bottom is the least confidential confidentiality statement! In the calculus CST this is expressed as the theorem

$$(\bot \Longrightarrow_{\mathrm{IF}} \mathcal{S}) = \top$$

for every confidentiality statement $\mathcal{S}$.

Negation

Lemma 19 *If P is a predicate with a variable tr ranging over the traces in X^* and is not equal to false and*

$$\mathcal{S} \triangleq \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : X^*$$

then

$$\begin{array}{l} \neg_{\mathrm{IF}} \mathcal{S} \\ = \\ \textbf{From } l : A^* \mid C \textbf{ MAY\_INFER } (\neg P(tr) \wedge tr \upharpoonright A = l) \textbf{ Over } tr : X^* \end{array}$$

Proof

By the semantic definition of negation

$$\mathcal{M}(\neg_{\text{IF}}\ \mathcal{S}) = \bigoplus\{\, x \in \text{Safety-Iflow} \mid x \otimes \mathcal{M}(S) = \bot \,\}$$

it must be shown that if

$$\mathcal{R} \triangleq \textbf{From}\ l : A^* \mid C\ \textbf{MAY\_INFER}\ (\neg P(tr) \wedge tr \restriction A = l)\ \textbf{Over}\ tr : X^*$$

then

- $\mathcal{R} \wedge_{\text{IF}} \mathcal{S} = \bot$;
- and for every confidentiality statement $\mathcal{T}$, such that $\mathcal{S} \wedge_{\text{IF}} \mathcal{T} = \bot$, $\mathcal{T} \Longrightarrow_{\text{IF}} \mathcal{R}$.

By the law for conjunction

$\mathcal{R} \wedge_{\text{IF}} \mathcal{S} =$
From $l : A^* \mid C$ **MAY_INFER** $(P(tr) \wedge \neg P(tr) \wedge tr \restriction A = l)$ **Over** $tr : X^*$
$=$ **From** $l : A^* \mid C$ **MAY_INFER** $false$ **Over** $tr : X^*$
$= \bot$ for a fixed alphabet X

It has been shown that $\mathcal{R}$ is in the set

$$\{\, x \in \text{Safety-Iflow} \mid x \otimes \mathcal{M}(S) = \bot \,\}$$

to show that $\mathcal{R}$ is the least upper bound of the set all that remains to be shown is that for every confidentiality statement $\mathcal{T}$ in the set $\mathcal{R}$ is greater than $\mathcal{T}$. This can be re-stated as for every $\mathcal{T}$, such that $\mathcal{S} \wedge_{\text{IF}} \mathcal{T} = \bot$, $\mathcal{T} \Longrightarrow_{\text{IF}} \mathcal{R}$.

$\mathcal{T}$ must be of the form

$$\textbf{From}\ l : A^* | C\ \textbf{MAY\_INFER}\ P'(tr) \wedge tr \restriction A = l\ \textbf{Over}\ tr : X^*$$

because the observations are over the set

$$\{\, l : A^* \mid C \,\}$$

and by definition a confidentiality statement must be consistent with the original observation (hence the predicate $tr \restriction A = l$). To show that

$$\mathcal{T} \Longrightarrow_{\text{IF}} \mathcal{R}$$

all that needs to shown is that

$$(P' \wedge tr \restriction A = l) \Longrightarrow (\neg P \wedge tr \restriction A = l)$$

This can be re-written as

$$
\begin{aligned}
&\neg(P' \wedge tr \restriction A = l) \vee (\neg P \wedge tr \restriction A = l) \\
&= (\neg P' \vee \neg tr \restriction A \neq l) \vee (\neg P \wedge tr \restriction A = l) \\
&= \neg P' \vee ((\neg P \vee tr \restriction A \neq l) \wedge (tr \restriction A = l \vee tr \restriction A \neq l)) \\
&[\text{By distributing } \vee \text{ over } \wedge] \\
&= \neg P' \vee ((\neg P \vee tr \restriction A \neq l) \wedge true) \\
&= \neg P' \vee (\neg P \vee tr \restriction A \neq l) \\
&= (\neg P' \vee \neg P) \vee tr \restriction A \neq l \\
&= \neg(P' \wedge P) \vee tr \restriction A \neq l \\
&= true \vee tr \restriction A \neq l \\
&[\text{Since } \mathcal{T} \wedge_{\mathrm{IF}} \mathcal{S} = \bot \text{ implies } P' \wedge P = false] \\
&= true
\end{aligned}
$$

therefore $\mathcal{T} \implies_{\mathrm{IF}} \mathcal{R}$ and consequently $\mathcal{R} = \neg_{\mathrm{IF}} S$. □

The negation of a confidentiality statement negates the predicate which characterizes the required amount of confidentiality with the proviso that the new characterization is consistent with the observation that gave rise to it. This law does not hold for the special cases of $\top$ and $\bot$.

4.3 Using the Calculus CST

In this section some confidentiality requirements are formalized within the calculus CST. Four examples are given: the first is a simple mandatory confidentiality policy; the second example is a Chinese walls type confidentiality requirement similar to those used by commercial companies; the third example of a confidentiality requirement is given which prevents aggregation of information; finally an approach to formalizing discretionary security is given.

In the examples given in this section interactions with a system are modelled by events in the alphabet of the system. The events can be tuples or functions or just simple names. No particular interpretation is necessary for the general theory and in different examples events can be interpreted to suit the occasion.

In the model of computation used by CSP, events with the same name synchronize with each other. This means that events are not modelling commands entered via a keyboard, for example. This is because any command can be entered and the machine will always respond, even if it is to inform the user that the command cannot be performed at this time or that the command is illegal. With the assumption of the synchronization of events only successful and legal commands are being modelled. This is equivalent to entering any command into the machine until a successful one is chosen. In [Jacob 88b] Jacob describes how information can not only flow from a user who outputs to one that inputs, transput leaks, but also via the sequence of inputs a user can do, synchronization leaks. The trace model developed for CSP is good enough to cope with both synchronization and transput leaks. Information can also flow via what events are refused at a particular point, the refusal model of

CSP is good enough to cope with this kind of information flow. If absolute time is considered part of an observation then the timed models of CSP are necessary.

4.3.1 A simple confidentiality property

If *Conf* and *Unclass* denote messages on confidential and unclassified channels then

$$CONSISTENCY \triangleq tr \upharpoonright Unclass = l$$

is the predicate which links possible inferences back to the trace of an observation l; and

$$CONFIDENTIALITY \triangleq tr \upharpoonright Conf \in Conf^*$$

is the predicate which proscribes that the most that can be inferred is that some trace of confidential communication has taken place; conjoining these together in the predicate calculus gives

$$FILT\_INFERENCE \triangleq CONSISTENCY \wedge CONFIDENTIALITY$$

and with the abbreviation, $CU \triangleq Conf \cup Unclass$, a confidentiality specification for a message filter is

$$\mathcal{S} \triangleq \textbf{From } l : Unclass^* \textbf{ MAY\_INFER } FILT\_INFERENCE \textbf{ Over } tr : CU^*$$

This confidentiality specification says that the most that can be inferred from an observation of events visible to an unclassified user is that any number of confidential events might have occurred. Another version of this specification which is semantically equivalent is

$$\textbf{From } l : Unclass^* \textbf{ MAY\_INFER } CONSISTENCY \textbf{ Over } tr : CU^*$$

This is because it is always the case that a trace drawn from CU^* when restricted to *Conf* will be a member of $Conf^*$ (even if it is the empty trace in $Conf^*$). Although the predicate is simpler it is sometimes useful to leave a predicate such as $tr \upharpoonright Conf \in Conf$ to act as a reminder when analysing confidentiality requirements for conflicts.

The specification $\mathcal{S}$ is purely about the confidentiality properties a system should have. It says nothing about the functionality of the message filter. The message filter could be a buffer or a system which prioritises messages depending on their content. Separating concerns of functionality from confidentiality allows a cleaner theory of system development and a factorization of effort of a system developer. Another advantage is that the confidentiality requirements can be analysed separately from the functional requirements.

4.3.2 A Chinese Walls Policy

The previous example of a confidentiality requirement was independent of the actions of a system user. A system was required to keep certain information from some user

in all circumstances. This is known as a mandatory security policy. An example of a confidentiality requirement which depends upon a system user's actions is the *Chinese Wall Security Policy*, first formalized in [Brewer & Nash 89]. This confidentiality requirement is based on the legal requirement of the UK Stock Exchange to forbid the use of insider knowledge. For example a market analyst who gives advice to companies must not deal with two competing companies where the analyst has insider knowledge of the plans or status of one which might affect the other. There is clearly a conflict of interest in this case but the analyst is free to advise companies which are not in competition with each other and may draw on general market information. In [Brewer & Nash 89] the Chinese Wall Security Policy is described as a combination of commercial discretion with legally enforceable mandatory controls. The discretion comes from the analyst having free choice on which companies it can advise provided there is no conflict of interest. Many other instances of Chinese Walls are found in the financial world.

To show how this kind of confidentiality requirement can be expressed some definitions are made. Suppose for simplicity there are only two competing companies. The sets of events which reveal inside information for the two companies A and B are denoted, respectively,

$$CO_A \text{ and } CO_B$$

The set of events which reveal inside information about a bank which is not in competition to companies A or B is denoted

$$BANK$$

The set of events which will give inside information is

$$INSIDE\_INFO \triangleq CO_A \cup CO_B \cup BANK$$

such that

$$CO_A \cap CO_B = \{\} \wedge CO_A \cap BANK = \{\} \wedge CO_B \cap BANK = \{\}$$

This is a reasonable assumption because if there were events in common then inside information would already be accessible to another organization. An analyst must also be able to access general market information, denoted

$$GENERAL\_INFO$$

which can refer to the competing companies and the bank. The general market information does not reveal inside information and this is expressed by the following condition

$$GENERAL\_INFO \cap INSIDE\_INFO = \{\}$$

The total set of events which is potentially available to an analyst is the union of the inside information and the general market information.

$$TOTAL \triangleq GENERAL\_INFO \cup INSIDE\_INFO$$

At this stage nothing more needs to be known about the events which are available. Later on in the development of a system the specific events used would be defined; at that stage the events that are needed would be known. The sets, such as CO_A, can be instantiated by any set of events as long as they satisfy the disjointness conditions defined above.

The confidentiality requirement for a Chinese Wall between insider information on company A and company B is presented in three parts. The first part of the confidentiality statement consists of interactions with a system before an analyst has gained any privileged information about either of the competing companies.

$$\begin{array}{l}\textbf{From } l : TOTAL^* \mid l \upharpoonright INSIDE\_INFO = \langle\rangle \textbf{ MAY\_INFER } tr = l \\ \textbf{Over } tr : TOTAL^*\end{array}$$

This says that for any interaction which does not involve either of the competing companies, characterized by the constraint $l \upharpoonright INSIDE\_INFO = \langle\rangle$, the analyst is allowed to see everything. At this point the analyst is not committed to dealing with one company or the other, for this reason the events of CO_A and CO_B must be available.

The second part of the confidentiality statement consists of interactions with a system when an analyst has engaged in some interaction with company A and hence may have become privy to inside information. If

$$SIDE_A \triangleq GENERAL\_INFO \cup CO_A \cup BANK$$

is company A's side of the Chinese wall then

$$\textbf{From } l : SIDE_A^* \textbf{ MAY\_INFER } tr \upharpoonright SIDE_A = l \textbf{ Over } tr : TOTAL^*$$

This says that on company A's side of the Chinese wall no interactions with company B are possible and the analyst cannot infer anything about any potential interaction with company B.

The third part of the confidentiality statement complements the second part, it deals with company B's side of the Chinese wall. If

$$SIDE_B \triangleq GENERAL\_INFO \cup CO_B \cup BANK$$

then

$$\textbf{From } l : SIDE_B^* \textbf{ MAY\_INFER } tr \upharpoonright SIDE_B = l \textbf{ Over } tr : TOTAL^*$$

This says that on company B's side of the Chinese wall no interactions with company A are possible and the analyst cannot infer anything about any potential interaction with company A. The full specification is formed by stacking the three parts to form a confidentiality statement over three interfaces.

$$\mathcal{CW} \triangleq \left\{ \begin{array}{l} \textbf{From } l : TOTAL^* \mid l \upharpoonright INSIDE\_INFO = \langle\rangle \\ \quad \textbf{MAY\_INFER } tr = l \textbf{ Over } tr : TOTAL^* \\ \\ \textbf{From } l : SIDE_A^* \textbf{ MAY\_INFER } tr \upharpoonright SIDE_A = l \textbf{ Over } tr : TOTAL^* \\ \\ \textbf{From } l : SIDE_B^* \textbf{ MAY\_INFER } tr \upharpoonright SIDE_B = l \textbf{ Over } tr : TOTAL^* \end{array} \right.$$

One way of implementing the policy formalized here might be to give a classification label to the operations in $SIDE_A$ and $SIDE_B$ so that they can be distinguished by some access control mechanism. A dynamic security level could then be associated with a user so that when an analyst accesses inside information about company A, the access control mechanism changes the clearance of the analyst and prevents him from looking at company B's details. At present the formalization of security policies such as the Chinese Wall is done by building a formal model of such an implementation. This tends to obscure the real issue with unnecessary details about how the operations are labelled and how a user has its clearance changed. It also causes design decisions to be taken too early in the development process. One of the benefits of abstract specifications is that they separate the *what* from the *how*. The specification $\mathcal{CW}$ describes *what* is required and not *how* it could be achieved, unlike access control models.

4.3.3 The NSA phone book requirement

The previous requirement showed that the behaviour of a subject can affect its right to information. Another interesting requirement is the prevention of aggregation of information. An example of this appears in [Goguen & Meseguer 84]: in this paper a collection of N items of a given type is not sensitive, but a collection of greater than N items is sensitive. An example of this is the familiar phone-book problem where the entries in a phone-book for an agency X have one classification, but the entire phone-book, or even a set of more than N entries from the book, has a greater classification.

To formalize this requirement the set of observations must be specified. Only two levels of clearance are considered, Low and Medium. The telephone directory enquiries of these two clearances are the sets

$$[Enquiry_L, Enquiry_M]$$

The events in these two sets might take the form of enquiries such as: "Give me all the telephone numbers of people whose names start with O"; or "What is Smith's telephone number?". The events in the set $Enquiry_M$ might allow more kinds of enquiries than $Enquiry_L$, or just allow more numbers to be know. Two functions from sequences of Low enquiries to the natural numbers and Medium enquiries to the natural numbers are assumed.

$$aggregate_L : Enquiry_L^* \rightarrow \mathbb{N}$$

$$aggregate_M : Enquiry_M^* \rightarrow \mathbb{N}$$

These functions denote the quantity of telephone numbers which can be inferred either directly or indirectly. The alphabet of the telephone directory is denoted by Directory.

$$[Directory]$$

The Directory alphabet includes the alphabets $Enquiry_L$ and $Enquiry_M$ as well as any other interactions. Another function which gives the quantity of telephone numbers which can be inferred is assumed.

$$aggregate : Directory^* \rightarrow \mathbb{N}$$

These functions are not elaborated on further except that no initial knowledge about telephone numbers is assumed.

$$aggregate(\langle\rangle) = 0 \ \wedge \ aggregate_L(\langle\rangle) = 0 \ \wedge \ aggregate_M(\langle\rangle) = 0$$

Like the previous example no more details about events are needed to write down the confidentiality specification. To show compliance between a system and the specification, the definition of the events and functions must be known. When a system is defined the exact nature of the events will be known and the functions $aggregate_L$, $aggregate_M$ and $aggregate$ can be precisely defined. As long as functions satisfy the assumption about no initial knowledge then they are suitable for use in the confidentiality specification. The appropriateness of the definition of the events and functions is more difficult to establish. For example the definition of the aggregate functions is a modelling assumption which, if done badly, may have no relevance to the real world. The classification of an event as a Low level enquiry or a Medium level enquiry is also a modelling assumption. There can be no formal correctness criteria for modelling the real world; but this formal approach points to where modelling decisions are made and need to be justified.

The confidentiality specification for the telephone directory is

$$\mathcal{TD} \triangleq \left\{ \begin{array}{l} \textbf{From } l : Enquiry_L^* \mid aggregate_L(l) < N_1 \\ \quad \textbf{MAY\_INFER } tr \upharpoonright Enquiry_L = l \ \wedge \ aggregate(tr) < N_1 \\ \quad \textbf{Over } tr : Directory^* \\ \\ \textbf{From } l : Enquiry_M^* \mid aggregate_M(l) < N_2 \\ \quad \textbf{MAY\_INFER } tr \upharpoonright Enquiry_M = l \ \wedge \ aggregate(tr) < N_2 \\ \quad \textbf{Over } tr : Directory^* \end{array} \right.$$

This confidentiality specification says that for any Low sequence of enquiries it is not possible to deduce N_1 or more telephone numbers; for any Medium level sequence of enquiries it is not possible to deduce N_2 or more telephone numbers. The constraints on the set of possible interactions are needed because, in the case of Low enquiries, it would be meaningless to prevent someone from deducing N_1, or more, telephone numbers if they have already been given N_1, or more, telephone numbers.

4.3.4 Discretionary security

Discretionary security refers to the ability of "user B" of a system to allow "user A" access to information held by "user B". To give an example of formalizing this type

of requirement only two users are assumed with a set of interactions available to each of them.

$$[User_A, User_B]$$

To allow $User_A$ to access information held by $User_B$ there must be a set of interactions to facilitate this, for this example these set of interactions are denoted by the name

$$[B\_Info]$$

This set is a subset of the set of interactions $User_A$ can make.

$$B\_Info \subset User_A$$

The alphabet of the system with which $User_A$ and $User_B$ interact is denoted by

$$[Total]$$

$User_A$ and $User_B$ are subsets of the alphabet of the system.

$$User_A \cup User_B \subset Total$$

Finally the event

$$grant_A : User_B$$

denotes the granting, by $User_B$, of access to information by $User_A$, via interactions in B_Info. Nothing more is known about the events. When a system is built any events can be used as long as there is an appropriate event which can instantiate the event $grant_A$, and there are appropriate sets of events for $User_A$, $User_B$ and B_Info.

The confidentiality specification is

$$\mathcal{DS} \triangleq \begin{cases} \textbf{From } l : User_A^* \mid l \upharpoonright B\_Info = \langle\rangle \\ \quad \textbf{MAY\_INFER } tr \upharpoonright User_A = l \textbf{ Over } tr : Total^* \\ \\ \textbf{From } l : User_A^* \mid l \upharpoonright B\_Info \neq \langle\rangle \\ \quad \textbf{MAY\_INFER } tr \upharpoonright User_A = l \ \wedge \ \ tr \upharpoonright (\{grant_A\} \cup B\_Info)_0 = \langle grant_A \rangle \\ \quad \textbf{Over } tr : Total^* \end{cases}$$

where the subscript 0 denotes the first element of the restricted trace $tr \upharpoonright (\{grant_A\} \cup B\_Info)$.

This specification says that $User_A$ cannot infer anything about $User_B$ without performing some interaction in B_Info, such as "read B's file". If $User_A$ does perform an interaction in B_Info then $User_B$ must have performed the event $grant_A$. Unfortunately a system can satisfy this specification by exhibiting all the traces required and allowing one more trace, where information flows to $User_A$ without the event $grant_A$ being performed. This is because confidentiality is about what a system should not do and what is required is a specification of what a system should do. A functional specification says what a system should do and the requirement that the

event $grant_A$ always occurs before any events in B_Info occur is captured by the following specification.

$$\forall\, tr : Total^* \cdot \quad (tr \upharpoonright B\_Info \neq \langle\rangle \wedge tr \upharpoonright (\{grant_A\} \cup B\_Info)_0 = \langle grant_A \rangle) \\ \vee\, tr \upharpoonright B\_Info = \langle\rangle$$

This specification can only be satisfied by systems which only allow $User_A$ to perform events from B_Info after $User_B$ performs the event $grant_A$, that is grants $User_A$ access to its information. The confidentiality specification ensures that there is no other way of gaining information from $User_B$.

The difference between this functional specification and the confidentiality specification lies in the fact that different notions of satisfaction are used. For a system to satisfy the confidentiality specification, $\mathcal{DS}$, it must have at least as many behaviours as $\mathcal{DS}$. For a system to satisfy the functional specification it must have no more behaviours. In chapter 6 the combination of confidentiality and functionality is examined and results given for when a system can satisfy both.

Chapter 5

Satisfaction and information flow

In this chapter, it is shown what it means for a system to satisfy a specification of confidentiality; how satisfaction respects conjunction and disjunction in CS and its concrete interpretations; and finally the use of a Galois correspondence is investigated.

5.1 Systems and confidentiality statements

To compare a system with a confidentiality statement it must be assumed that a system like an observation is an object in an information flow category, $\mathcal{IF}$.

Assumption: A system is an object in $\mathcal{IF}$.

Example 15 In Safety-Iflow a system is a pair

$$(A, T)$$

where A is the alphabet of the system and T is a non-empty, prefix closed set of traces. $\triangle$

The observations of a system are difficult to characterize at an abstract level; therefore the only thing which is assumed is that observations of a system form a subcategory of $\mathcal{IF}$. The subcategory of observations of a system is a "window" onto that system, they are a subset of all the possible observations of a system.

Notation 4 *A set* **of sets** *of observations of a system is denoted by W, each $w : W$ is called a window.*

In $\mathcal{IF}$ a window is just a set of observations which is a subcategory of $\mathcal{IF}$. In concrete categories, such as Safety-Iflow, an observation of a system and a window onto a system can be precisely characterized.

Definition 41 *If B is an alphabet and (A, T) is a system and $B \subseteq A$ then*

$$\{\, (B, l) \mid l \in B^* \wedge l \in T \upharpoonright B \,\}$$

is the set of observations through B and is a window onto (A,T). To make discussion of windows less verbose B will be referred to as a window, meaning the set of observations above. $\diamond$

The term window is used for a set of observations rather than interface because the term interface is reserved for a different meaning in chapter 6.

From an observation of a system and knowledge of the system's behaviours, the behaviours of the system which might have caused that observation can be deduced. In general this deduction will give more than the actual behaviour which gave rise to the observation. This means that there is some uncertainty associated with the observation of a system. It is that uncertainty about a system's behaviour with respect to an observation which this section captures and uses for comparison with a confidentiality statement.

In $\mathcal{IF}$ the uncertainty associated with an observation and a system will have an arrow going to both objects. The object representing the strongest inference one can make from an observation of a system coincides with the categorical product of the observation and that system. The first step in constructing the strongest inference over a set of observations is to define a functor which puts a system and observation together as a product.

Lemma 20 *Let $f : c \to d$ be any arrow of $\mathcal{IF}$ and S a given object of $\mathcal{IF}$ then*

$$\begin{aligned} [S \times \_](c) &= S \times c \\ [S \times \_](f) &= id_S \times f \end{aligned}$$

then $[S \times \_]$ is a functor.

Proof

To prove that $[S \times \_]$ is a functor the preservation of identities is first checked.

$$\begin{aligned} [S \times \_](id_c) &= id_S \times id_c \\ &= id_{S \times c} \qquad \text{[By lemma 4 in chapter 2]} \end{aligned}$$

therefore $[S \times \_]$ preserves the identity arrow.

To conclude the proof the preservation of arrow composition is checked. Suppose there are arrows $f : c \to d$ and $g : d \to e$ then

$$\begin{aligned} [S \times \_](f);[S \times \_](g) &= (id_S \times f);(id_S \times g) \\ &= (id_S;id_S) \times (f;g) \qquad \text{[by lemma 3]} \\ &= [S \times \_](f;g) \\ &\quad \text{[by the identity axiom and definition of } [S \times \_] \text{]} \end{aligned}$$

$\square$

Example 16 If

$$T \triangleq a \to b \to STOP$$

then in Safety-Iflow

$$T = (\{a, b\}, \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\})$$

from this the following can be deduced:

$$[T \times \_](\{b\}, \{\langle b\rangle\}) = (\{a, b\}, \{\langle a, b\rangle\})$$

if $(\{b\}, \{\langle b\rangle\})$ is observed then the behaviour $\langle a, b\rangle$ over the alphabet $\{a, b\}$ gave rise to it;

$$[T \times \_](\{a\}, \{\langle a\rangle\}) = (\{a, b\}, \{\langle a\rangle, \langle a, b\rangle\})$$

if $(\{a\}, \{\langle a\rangle\})$ is observed then the either $\langle a\rangle$ or $\langle a, b\rangle$ over the alphabet $\{a, b\}$ gave rise to it;

$$[T \times \_](\{a\}, \{\langle a, a\rangle\}) = (\{a, b\}, \{\})$$

the observation $(\{a\}, \{\langle a, a\rangle\})$ is inconsistent with the system T, that is there is no behaviour in T which could result in the observation $(\{a\}, \{\langle a, a\rangle\})$. $\triangle$

For notational convenience and to suggest the underlying motivation for $[S \times \_]$ the following notation is introduced.

Notation 5 *If W is a set of windows onto a system S then*

INFER $W\ S$

denotes the functor $[S \times \_]$ restricted to each set of observations of S through each window $w : W$.

INFER $W\ S$ is chosen as the denotation for $[S \times \_]$ because, in the concrete Safety-Iflow, $[S \times \_]$ is an abstraction of the inference function defined in [Jacob 88a].

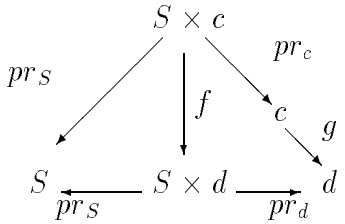


Figure 5.1: Product diagram underlying natural transformation.

The next proposition shows that **INFER** $W\ S$ is just another way of writing a confidentiality statement.

Proposition 3 *If S is a system and W is the set of windows through which S is observed then the functor* **INFER** $W\ S$ *is a confidentiality statement.*

Proof Let $g : c \rightarrow d$ be an arrow in the domain of **INFER** $W\ S$, consider figure 5.1, by definition of product the arrow f in figure 5.1 is $\langle pr_S, pr_c; g\rangle$, therefore

$$\begin{array}{rcll} \langle pr_S, pr_c; g\rangle & = & \langle pr_S; id_S, pr_c; g\rangle & \text{[By the identity axiom]} \\ & = & id_S \times g & \text{[By definition 28]} \end{array} \quad (5.1)$$

By definition of product

$$\begin{array}{rcll} [S \times \_](g); pr_d & = & (id_S \times g); pr_d & \text{[by definition of } [S \times \_]] \\ & = & \langle pr_S, pr_c; g\rangle; pr_d & \text{[substituting in 5.1]} \\ & = & pr_c; g & \text{[by definition of product]} \\ & = & pr_c; Id_{\mathcal{IF}}(g) & \text{[by definition of identity functor]} \end{array}$$

This equation proves that the diagram of figure 5.2 commutes, therefore the projection arrows form the components of a natural transformation to $Id_{\mathcal{IF}}$. □

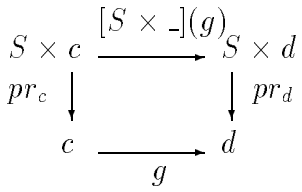


Figure 5.2: Commuting diagram for natural transformation

Some notation is now introduced to make the theory more concise.

Notation 6 *The observations of a system P through windows W is denoted P_W. Restriction of a functor, F, to a subcategory of $\mathcal{IF}$ such as P_W is denoted*

$$P_W \lhd F$$

Proposition 3 showed that **INFER** $W\ P$ is a confidentiality statement. The ordering over functors, given in chapter 3, can be used to define when **INFER** $W\ P$ satisfies a confidentiality specification, given in terms of a confidentiality statement.

Definition 42 *A system P is said to satisfy a confidentiality statement $\mathcal{T}$ over observations of P (through windows in W), denoted*

$$P \quad \textbf{s-sat}^{W} \quad \mathcal{T}$$

if and only if **INFER** $W\ P \quad \trianglerighteq (P_W \lhd \mathcal{T})$. ◇

This definition is inevitable given the fact that **INFER** $W\ P$ is a confidentiality statement and the reader has accepted the definition of when one confidentiality statement is better than another.

Example 17 Suppose it is necessary to limit the knowledge of an observer through the window $\{a\}$. If

$$\mathcal{T} \triangleq \textbf{From } l : \{a\}^* \textbf{ MAY\_INFER } (tr \restriction \{a\} = l) \wedge \#(tr \restriction \{b\}) \leq \#l \textbf{ Over } tr : \{a, b\}^*$$

then an observer through the window $\{a\}$ is limited to knowing that the number of b events which could have occurred is bounded above by the number of a events observed. No observer can deduce anything more than this; therefore an observer would have no idea how many b events had actually occurred and would have no idea when in the sequence of a events any b event might occur.
If

$$Q \triangleq (a \rightarrow b \rightarrow STOP) [] (b \rightarrow a \rightarrow STOP)$$

then

$$Q = (\{a, b\}, \{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle a, b\rangle, \langle b, a\rangle\})$$

and the global observations of Q are:

$$\begin{array}{l} (\{a, b\}, \{\langle\rangle\}), \quad (\{a, b\}, \{\langle a\rangle\}), \\ (\{a, b\}, \{\langle b\rangle\}), \quad (\{a, b\}, \{\langle a, b\rangle\}) \\ (\{a, b\}, \{\langle b, a\rangle\}) \end{array}$$

the observations of Q through the window $\{a\}$ are:

$$Q_{\{\{a\}\}} = \{(\{a\}, \{\langle\rangle\})\ (\{a\}, \{\langle a\rangle\})\}$$

The inference over these observations through $\{a\}$ is

$$\begin{array}{l} \textbf{INFER } \{\{a\}\}\ Q\ ((\{a\}, \{\langle\rangle\})) = (\{a, b\}, \{\langle\rangle, \langle b\rangle\}) \\ \textbf{INFER } \{\{a\}\}\ Q\ ((\{a\}, \{\langle a\rangle\})) = (\{a, b\}, \{\langle a\rangle, \langle a, b\rangle, \langle b, a\rangle\}) \end{array}$$

The predicate

$$tr \in \{a, b\}^* \wedge tr \restriction \{a\} = t \wedge \#(tr \restriction \{b\}) \leq \#t$$

which characterizes $\mathcal{T}$ when viewed as a set of traces is

$$\{\langle\rangle\}$$

when $t = \langle\rangle$ and

$$\{\langle a\rangle, \langle a, b\rangle, \langle b, a\rangle\}$$

when $t = \langle a\rangle$, therefore

$$\textbf{INFER } \{\{a\}\}\ Q \quad \trianglerighteq\ (Q_{\{\{a\}\}} \lhd \mathcal{T})$$

$\triangle$

In the next few propositions some properties of satisfaction between systems and confidentiality statements are shown.

Corollary 1 *If P is a system and W a set of windows then*

$$P \ \textbf{s-sat}^{\mathrm{W}} \ \textbf{INFER} \ W \ P$$

and

$$P \ \textbf{s-sat}^{\mathrm{W}} \ (P_W \lhd \mathcal{T}) \implies \textbf{INFER} \ W \ P \ \trianglerighteq \ (P_W \lhd \mathcal{T})$$

Proof

By proposition 3 and definition 42. □

Corollary 1 shows that **INFER** $W \ P$ is the strongest confidentiality statement which P will satisfy through windows W. The next proposition shows that any functor which has a natural transformation to **INFER** $W \ P$ is a confidentiality statement.

Proposition 4 (**INFER** $W \ P \ \trianglerighteq \ P_W \lhd F) \implies (P_W \lhd F)$ *is a confidentiality statement over the observations of P through windows in W.*

Proof If (**INFER** $W \ P \ \trianglerighteq \ P_W \lhd F$) then by definition of $\trianglerighteq$ (definition 34) there is a commuting diagram with a natural transformation from $(P_W \lhd F)$ to the identity, therefore $(P_W \lhd F)$ is a confidentiality statement. □

The next proposition shows that if a system, P, satisfies a specification through some windows, W, then in the calculus CS and its concrete instances, such as CST, the specification implies **INFER** $W \ P$.

Proposition 5 *If P is a system, W a set of windows and $\mathcal{S}$ a confidentiality specification then*

$$P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$$

if and only if over the subcategory P_W

$$\mathcal{S} \Longrightarrow_{\mathrm{IF}} \textbf{INFER} \ W \ P$$

Proof If $P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$ then

$$\textbf{INFER} \ W \ P \ \trianglerighteq \ (P_W \lhd \mathcal{S})$$

in a Heyting algebra

$$x \geq y \iff (y \implies x)$$

therefore over the subcategory denoted by P_W

$$\mathcal{S} \Longrightarrow_{\mathrm{IF}} \textbf{INFER} \ W \ P$$

□

5.1.1 Preserving satisfaction

The natural way to structure a complex specification is as the conjunction of many individual requirements. To develop "secure" systems of a realistic size it is necessary to develop components of the system which satisfy individual requirements; and then, by some means, assemble together the components to give a single system which satisfies all the requirements simultaneously.

In chapter 3 a calculus was presented with operators of conjunction and disjunction. Laws are now developed which show that components can be assembled together using product and coproduct to satisfy a complex specification in terms of conjunction and disjunction in CS. Each law is presented as an inference rule of the form:

$$\frac{\textit{antecedent} \qquad \textit{antecedent}}{\textit{consequent}}$$

If the truth of each *antecedent* is established then the truth of the consequent can be assured. The next proposition is about how the strongest confidentiality statement, satisfied by the product of two objects, relates to the strongest confidentiality statements of the individual objects.

Proposition 6 *If P and Q are systems with windows W then*

$$\textbf{INFER}\ W\ (P \times Q) \quad \trianglerighteq\ ((P \times Q)_W \lhd \textbf{INFER}\ W\ P\ \otimes \textbf{INFER}\ W\ Q\ )$$

Proof Let $f : c \to d$ be an arrow in the subcategory of $\mathcal{IF}$ denoted by $(P \times Q)_W$.

$(\textbf{INFER}\ W\ P\ \otimes \textbf{INFER}\ W\ Q\ )(f); pr_{P \times d \times Q}$
$= ([P \times \_] \otimes [Q \times \_])(f); pr_{P \times d \times Q}$
$= ([P \times \_](f) \times [Q \times \_](f); pr_{P \times d \times Q}$
[By defn. of $\otimes$]
$= (id_P \times f \times id_Q \times f); pr_{P \times d \times Q})$
[By defn. of $[P \times \_]$]
$= ((id_P \times f \times id_Q) \times f); pr_{P \times d \times Q})$
[associativity of product]
$= \langle pr_{P \times c \times Q}; (id_P \times f \times id_Q), pr_c; f \rangle; pr_{P \times d \times Q}$
[By defn. of product arrow]

$= pr_{P\times c\times Q}; (id_P \times f \times id_Q)$
[By defn. of product]
$= pr_{P\times c\times Q}; (id_P \times id_Q \times f)$
[By isomorphism of products]
$= pr_{P\times c\times Q}; (id_{P\times Q} \times f)$
[By lemma 4 in chapter 2]
$= pr_{P\times c\times Q}; [P \times Q \times \_](f)$
[By defn. of $[P \times \_]$]
$= pr_{P\times c\times Q}; \mathbf{INFER}\ W\ (P \times Q)\ (f)$

This equational reasoning shows that the projection arrows give rise to natural transformations which ensure that the commuting diagram of definition 34 is satisfied. □

This proposition is interesting because in Safety-Iflow the product of two systems is their parallel composition.

Definition 43 (Parallel composition) *If P and Q are non-empty, prefix closed sets of traces (i.e. processes) then their parallel composition is defined as*

$$P \| Q \ \triangleq\ (\alpha P \cup \alpha Q, \{\, t : (\alpha P \cup \alpha Q)^* \mid t \upharpoonright \alpha P \in P \ \wedge\ t \upharpoonright \alpha Q \in Q \,\})$$

◇

Example 18 If

$$P \ \triangleq\ a \rightarrow b \rightarrow STOP$$

and

$$Q \ \triangleq\ (a \rightarrow STOP) [\!] (b \rightarrow STOP)$$

then

$$P = (\{a, b\}, \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}) \text{ and } Q = (\{a, b\}, \{\langle\rangle, \langle a\rangle, \langle b\rangle\})$$

and

$$P \| Q = (\{a, b\}, \{\langle\rangle, \langle a\rangle\})$$

or

$$P \| Q = a \rightarrow STOP$$

△

Theorem 3 *For any two systems P and Q in Safety-Iflow, $P \times Q = P \| Q$.*

Proof Direct from lemma 13 in chapter 4, which gives the result of the product of two objects, and definition 43 of parallel composition. □

Example 19 If P and Q are as defined in example 18 then

$$P \times Q = (\{a\}, \{\langle\rangle, \langle a\rangle\})$$

and through the window $\{a\}$

$$(P \times Q)_{\{\{a\}\}} = \{(\{a\}, \{\langle\rangle\}),\ (\{a\}, \{\langle a\rangle\})\}$$

Examining the observation $(\{a\}, \{\langle a\rangle\})$ reveals that

$$\textbf{INFER}\ \{\{a\}\}\ P\ (\{a\}, \{\langle a\rangle\}) = (\{a, b\}, \{\langle a\rangle, \langle a, b\rangle\})$$

and

$$\textbf{INFER}\ \{\{a\}\}\ Q\ (\{a\}, \{\langle a\rangle\}) = (\{a\}, \{\langle a\rangle\})$$

therefore

$$\begin{array}{l}\textbf{INFER}\ \{\{a\}\}\ P\ \otimes\ \textbf{INFER}\ \{\{a\}\}\ Q\ (\{a\}, \{\langle a\rangle\}) \\ = (\{a, b\}, \{\langle a\rangle, \langle a, b\rangle\}) \times (\{a\}, \{\langle a\rangle\}) \\ = (\{a, b\}, \{\langle a\rangle\}) \\ = \textbf{INFER}\ \{\{a\}\}\ (P \times Q)\ (\{a\}, \{\langle a\rangle\}) \\ [\text{Since } P \times Q = Q]\end{array}$$

The reader is welcome to check the other observation!

$\triangle$

Using the previous proposition the inference rule in the following theorem is easy to prove.

Theorem 4 *If P and Q be systems with windows W and let $\mathcal{S}$ and $\mathcal{T}$ be confidentiality statements over the subcategories P_W and Q_W respectively, then the following inference rule is valid.*

$$\frac{P\ \textbf{s-sat}^{\text{W}}\ \mathcal{S} \qquad Q\ \textbf{s-sat}^{\text{W}}\ \mathcal{T}}{P \times Q\ \textbf{s-sat}^{\text{W}}\ \mathcal{S} \otimes \mathcal{T}}$$

Proof

$P\ \textbf{s-sat}^{\text{W}}\ \mathcal{S} \wedge Q\ \textbf{s-sat}^{\text{W}}\ \mathcal{T}$

$\Longrightarrow \textbf{INFER}\ W\ P\ \ \trianglerighteq\ \mathcal{S} \wedge \textbf{INFER}\ W\ Q\ \ \trianglerighteq\ \mathcal{T}$
[By definition 42]
$\Longrightarrow (P \times Q)_W \lhd \textbf{INFER}\ W\ P\ \ \otimes\ \textbf{INFER}\ W\ Q\ )\ \trianglerighteq\ (P \times Q)_W \lhd \mathcal{S}\ \otimes\ \mathcal{T}$
[By defn. of $\otimes$]
$\Longrightarrow$

$(\ (P \times Q)_W \lhd \textbf{INFER}\ W\ P \times Q\ \ )$
$\trianglerighteq$
$(\ (P \times Q)_W \lhd \textbf{INFER}\ W\ P\ \ \otimes\ \textbf{INFER}\ W\ Q\ \ )$

[By proposition 6]

$\Longrightarrow P \times Q\ \ \textbf{s-sat}^{\text{W}}\ \ \mathcal{S}\ \otimes\ \mathcal{T}$

[Transitivity of $\trianglerighteq$ & defn. 42]

□

The result of theorem 4 can be used immediately.

Corollary 2 *If P, Q; W; $\mathcal{S}$ and $\mathcal{T}$ are the same as in theorem 4 then over the subcategory $(P \times Q)_W$*

$$\frac{P \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S} \qquad Q \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{T}}{P \times Q \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T}}$$

Proof

By theorem 4 and definition of $\wedge_{\mathrm{IF}}$. □

This corollary allows a designer to split the task of finding a system which will satisfy the conjunction of two confidentiality statements. If the designer can find two subsystems which satisfy the conjuncts of the overall confidentiality statement then they can be combined 'securely' using parallel composition. This is analogous to the CSP law

$$\frac{P \,\textbf{sat}\, S(tr) \qquad Q \,\textbf{sat}\, T(tr)}{P \| Q \,\textbf{sat}\, S(tr \upharpoonright \alpha P) \wedge T(tr \upharpoonright \alpha Q)}$$

which allows a designer to split the proof task of verifying the system $P \| Q$ into verifying the individual processes P and Q.

Example 20 Suppose a limit on the inference of an observer through the window $\{a\}$ is required. If

$$\mathcal{S} \ \triangleq \ \textbf{From } l : \{a\}^* \ \textbf{MAY\_INFER } tr \upharpoonright \{a\} = l \wedge tr \upharpoonright \{c\} \in \{c\}^* \ \textbf{Over } tr : \{a, c\}^*$$

then an observer through the window $\{a\}$ is not allowed to infer anything about how many c events may have occurred or when they might occur in any sequence of a events observed.

If

$$P \ \triangleq \ (a \rightarrow STOP_{\{a\}}) \| (c \rightarrow P)$$

then

$$P = (\{a, c\}, \{\, s \mid s \upharpoonright \{a\} \in \{\langle\rangle, \langle a \rangle\} \wedge s \upharpoonright \{c\} \in \{c\}^* \,\})$$

by definition 43 of parallel composition. Clearly

$$P \ \textbf{s-sat}^{\{\{a\}\}} \ \mathcal{S}$$

By corollary 2 P can be composed in parallel with the system Q defined in example 17 and know that they will satisfy $\mathcal{S}$ conjoined with the specification $\mathcal{T}$ defined in example 17, that is:

$$P \| Q \ \textbf{s-sat}^{\{\{a\}\}} \ \mathcal{S} \wedge_{\text{IF}} \mathcal{T}$$

$\triangle$

In category theory the natural extension to the above theorem is to look for its dual, therefore the next step is to look for the duals of theorem 4 and corollary 2. First the following auxiliary lemma is needed.

Lemma 21 *There exists a natural transformation*
$[P \times \_] \stackrel{\bullet}{\rightarrow} [(P + Q) \times \_]$

Proof

If $f : c \rightarrow d$ is an arrow from an object c to an object d in $\mathcal{IF}$ then

$$\begin{array}{rcll} [P \times \_](f); i_P \times id_d & = & (id_P \times f); (i_P \times id_d) & \text{[Defn. of } [P \times \_]] \\ & = & (id_P; i_P) \times (f; id_P) & \text{[By lemma 3]} \\ & = & (i_P; id_{P+Q}) \times (id_c; f) & \text{[By the identity axiom]} \\ & = & (i_P \times id_c); (id_{P+Q} \times f) & \text{[By lemma 3]} \\ & = & (i_P \times id_c); [(P + Q) \times \_](f) & \end{array}$$

$\square$

Proposition 7 *If P and Q are systems; W is a set of windows; and $\mathcal{S}$ and $\mathcal{T}$ are confidentiality specifications defined over the subcategory $(P + Q)_W$ then*

$$\begin{array}{l} \textbf{INFER} \ W \ P \quad \trianglerighteq \ \mathcal{S} \wedge \textbf{INFER} \ W \ Q \quad \trianglerighteq \ \mathcal{T} \\ \Longrightarrow \\ \textbf{INFER} \ W \ (P + Q) \quad \trianglerighteq \ (\mathcal{S} \oplus \mathcal{T}) \end{array}$$

Proof By lemma 21 and definition of coproduct

$$[P \times \_] \stackrel{\bullet}{\rightarrow} [(P + Q) \times \_] \text{ and } [Q \times \_] \stackrel{\bullet}{\rightarrow} [(P + Q) \times \_]$$

and the diagram of definition 34 holds (since the commuting diagram of the coproduct arrows gives rise to the commuting diagram of natural transformations). This means that

INFER $W \ (P + Q) \ \trianglerighteq$ **INFER** $W \ P$ *and* **INFER** $W \ (P + Q) \ \trianglerighteq$ **INFER** $W \ Q$

By transitivity of $\trianglerighteq$

$$\textbf{INFER}\ W\,(P+Q) \quad \trianglerighteq\ \mathcal{S} \quad \text{and} \quad \textbf{INFER}\ W\,(P+Q) \quad \trianglerighteq\ \mathcal{T}$$

Therefore

$$\mathcal{S} \oplus \mathcal{T} \xrightarrow{\bullet} [(P+Q) \times \_]$$

by definition of least upper bound. □

This proposition is useful because in Safety-Iflow coproduct corresponds to environmental choice.

Definition 44 (Choice) *If P and Q are non-empty, prefix closed sets of traces (i.e. processes) with identical alphabets then their choice composition is defined as*

$$P[]Q \ \triangleq\ (\alpha P, \{\, t : (\alpha P)^* \mid t \in P \ \vee\ t \in Q \,\})$$

◇

Theorem 5 *For any two systems P, Q in Safety-Iflow with identical alphabets,*

$$P + Q = P[]Q$$

Proof Direct from lemma 13 of chapter 4, which gave the construction for coproduct, and definition 44 of choice composition. □

From proposition 7 the following theorem is easy to prove.

Theorem 6 *If P and Q are systems; W is a set of windows; and the confidentiality specifications $\mathcal{S}$ and $\mathcal{T}$ are defined over $(P+Q)_W$ then the following law is valid*

$$\frac{P\ \textbf{s-sat}^{\,\text{W}}\ \mathcal{S} \qquad Q\ \textbf{s-sat}^{\,\text{W}}\ \mathcal{T}}{P+Q\ \textbf{s-sat}^{\,\text{W}}\ \mathcal{S}\ \oplus\ \mathcal{T}}$$

Proof

$P\ \textbf{s-sat}\ \mathcal{S}\ \wedge\ Q\ \textbf{s-sat}\ \mathcal{T}$

$\Longrightarrow \textbf{INFER}\ W\,P \quad \trianglerighteq\ (P_W \lhd \mathcal{S})\ \wedge\ \textbf{INFER}\ W\,Q \quad \trianglerighteq\ (Q_W \lhd \mathcal{T})$
[By definition 42]
$\Longrightarrow \textbf{INFER}\ W\,(P+Q) \quad \trianglerighteq\ (\mathcal{S} \oplus \mathcal{T})$
[By proposition 7]

$\Longrightarrow (P+Q)\ \textbf{s-sat}^{\,\text{W}}\ (\mathcal{S} \oplus \mathcal{T})$

[By definition 42]

□

Corollary 3 *If P, Q; W; $\mathcal{S}$ and $\mathcal{T}$ are the same as in theorem 6 then over the subcategory $(P+Q)_W$*

$$\frac{P \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S} \qquad Q \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{T}}{P + Q \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S} \vee_{\mathrm{IF}} \mathcal{T}}$$

Proof

By theorem 6 and definition of $\vee_{\mathrm{IF}}$. □

Corollary 3 enables a designer of a secure system to split the task of finding a system which will satisfy the disjunction of two confidentiality statements. If the designer can find two sub-systems which satisfy the disjuncts of the overall specification then they can be combined 'securely' using choice composition.

Example 21 If the alphabet of P is extended, defined in example 20, to include the event b, using the alphabet extension operator $\_ +_{\{b\}}$ defined in [Hoare 85], and similarly extend the alphabet of Q, defined in example 17, to include the event c, and take the confidentiality specifications which they satisfy then by corollary 3

$$P_{+\{b\}} \ [\!] \ Q_{+\{c\}} \ \ \textbf{s-sat}^{\,\{\{a\}\}} \ \ \mathcal{S} \vee_{\mathrm{IF}} \mathcal{T}$$

△

Other laws can be derived immediately from corollaries 2 and 3.

Corollary 4 *If P; W; $\mathcal{S}$ and $\mathcal{T}$ are the same as in theorem 4 then over the subcategory $(P \times Q)_W$*

$$\frac{P \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S}}{P \ \textbf{s-sat}^{\,\mathrm{W}} \ \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T}}$$

Proof

By the properties of $\wedge_{\mathrm{IF}}$. □

Corollary 5 *If P, Q; W; and $\mathcal{S}$ are the same as in theorem 6 then over the subcategory $(P+Q)_W$*

$$\frac{P \ \textbf{s-sat}^{\text{W}} \ \mathcal{S}}{P + Q \ \textbf{s-sat}^{\text{W}} \ \mathcal{S}}$$

Proof

By the properties of coproduct and $\vee_{\text{IF}}$. □

The final theorem of the subsection is a very useful property of systems in CSP when any behaviour of the system can be observed.

Theorem 7 *If P is a system, W is a set of windows and*

$$\exists w : W \cdot \alpha P = w$$

then for any confidentiality specification $\mathcal{S}$

$$P \ \textbf{s-sat}^{\text{W}} \ \mathcal{S}$$

Proof If for some $w : W$ $\alpha P = w$ then

$$\forall l : P_W \cdot P \times l = l$$

which is the identity function.

$$P_W \lhd \mathcal{S} \stackrel{\bullet}{\rightarrow} [P \times \_]$$

since by definition of a confidentiality statement there is always a natural transformation to the identity and the identity arrows from l to l make the diagram of definition 34 commute. This means that

$$\textbf{INFER} \ W \ P \quad \trianglerighteq \ (P_W \lhd \mathcal{S})$$

which in turn means that by definition 42

$$P \ \textbf{s-sat}^{\text{W}} \ \mathcal{S}$$

□

5.2 A Galois correspondence

The theorems 2 and 3 of section 5.1 give rules for combining components, which satisfy individual requirements, together to give a system which satisfies an overall specification. In this section the concept of a Galois correspondence is explored to give a necessary and sufficient condition for when components of a specification can be conjoined to give a specification at least as good as the original. This technique can in principle be used for specifications which are not constructed from conjunctions and disjunctions. The technique can also be used to derive guards which when placed in parallel with an "unsecure" system will give a "secure" system.

By lemma 6 the relation $\trianglerighteq$ over confidentiality statements and systems is a preorder. Any preorder is a category with the objects of the preorder forming the objects of the category. There is an arrow from an object c to an object d if and only if $c \leq d$. Such categories are known as preorder categories. A functor between two preorder categories is just a monotonic function.

Definition 45 *Let S and T be two preorders and $L : S \to T$ and $R : T \to S$ be two monotonic functions. A Galois correspondence exists between S and T if and only if:*

$$\forall x \in S; y \in T \cdot L(x) \leq_T y \iff x \leq_S R(y)$$

The function L is said to be a left adjoint of R, and R is a right adjoint of L. $\diamond$

The utility of the Galois correspondence is that by choosing L and R carefully a necessary and sufficient condition for breaking systems and specifications into "simpler" components is obtained. It shall be shown how any confidentiality statement can be approximated from two conjoined confidentiality statements using the Galois correspondence.

A necessary and sufficient condition for the existence of a right adjoint is given by the following definition taken from [Lambek & Scott 86].

Definition 46 *Let $R : S \to T$ be a functor between two partial orders such that for any collection of elements of T the greatest lower bound exists in T, and such that R preserves greatest lower bounds, then R has a left adjoint L if and only if*

$$\forall x \in T \cdot L(x) = \bigotimes \{ y \in S \mid x \leq_T R(y) \}$$

that is $L(x)$ is defined to be the greatest lower bound of the set of objects y such that $x \leq R(y)$.

$\diamond$

As well as defining when R has a left adjoint the definition gives a way of constructing it. The relative pseudo complement which provided the formal meaning of implication

in CS is the right adjoint of $\otimes$ in the Galois correspondence which arises from the definition of relative pseudo complement.

A very useful theorem can now be stated, it shows how in principle confidentiality is preserved by system operators such as prefixing as well as parallel and choice composition.

Theorem 8 *If R is a functor from Iflow-order to itself such that R preserves greatest lower bounds and*

$$\bigotimes\{\, y \in \mathit{Iflow\text{-}order} \mid R(y) \trianglerighteq x \,\}$$

exists in Iflow-order then

$$\forall \mathcal{S}, \mathcal{Z} : \text{Iflow-order} \cdot \mathcal{S} \trianglerighteq \bigotimes\{\, y \in \mathit{Iflow\text{-}order} \mid R(y) \trianglerighteq Z \,\} \Longleftrightarrow R(\mathcal{S}) \trianglerighteq \mathcal{Z}$$

Proof By definition of a Galois correspondence. □

Note that R is a functor which takes a functor as an argument and delivers a functor as a result. The theorem gives a necessary and sufficient condition for when a confidentiality statement given by R meets a specification $\mathcal{Z}$. An example of how this theorem can be used is when R corresponds to placing a system in parallel with a fixed system.

Definition 47 *Let $R_{\mathcal{T}}$ be a functor from Iflow-order to itself such that*

$$\forall \mathcal{S} \in \mathit{Iflow\text{-}order} \cdot R_{\mathcal{T}}(\mathcal{S}) = \mathcal{S} \otimes \mathcal{T}$$

◇

The left adjoint of $R_{\mathcal{T}}$ can be used to give a theorem on how to derive sub-components of any confidentiality statement.

Corollary 6

$$\forall \mathcal{S}, \mathcal{T}, \mathcal{Z} : \text{Iflow-order} \cdot \mathcal{S} \trianglerighteq \bigotimes\{\, d \in \mathit{Iflow\text{-}order} \mid d \otimes \mathcal{T} \trianglerighteq \mathcal{Z} \,\} \Longleftrightarrow \mathcal{S} \otimes \mathcal{T} \trianglerighteq \mathcal{Z}$$

Proof

By definition 47 and theorem 8. □

The utility of such a theorem is that if there is a confidentiality specification, $\mathcal{Z}$, of the form

$$\textbf{From } l : A^* \textbf{ MAY\_INFER } P(tr) \textbf{ Over } tr : B^*$$

and another confidentiality specification, $\mathcal{T}$, for which there is a system, S, which satisfies $\mathcal{T}$, then S and $\mathcal{T}$ can be re-used. Corollary 6 gives a necessary and sufficient condition for finding a simpler specification $\mathcal{S}$ which when conjoined with $\mathcal{T}$ will be no worse than the original specification $\mathcal{Z}$. A more direct but weaker result for systems follows almost immediately.

Theorem 9

$$\forall P, Q : \mathcal{IF}; \mathcal{Z} : \text{ Iflow-order} \cdot$$
$$P \ \textbf{s-sat}^{\text{w}} \ \bigotimes \{ \, d \in \textit{Iflow-order} \mid d \otimes \textbf{INFER} \ W \ Q \ \ \trianglerighteq \ \mathcal{Z} \, \}$$
$$\Longrightarrow$$
$$P \times Q \ \textbf{s-sat}^{\text{w}} \ \ \mathcal{Z}$$

Proof

$$\begin{array}{lll}
\textbf{INFER} \ W \ P & \trianglerighteq & \bigotimes \{ \, d \in \text{ Iflow-order} \mid d \otimes \textbf{INFER} \ W \ Q \ \ \trianglerighteq \ \mathcal{Z} \, \} \\
& & \text{[By definition 42]} \\
& \Longleftrightarrow & \textbf{INFER} \ W \ P \ \otimes \textbf{INFER} \ W \ Q \ \ \trianglerighteq \ \mathcal{Z} \\
& & \text{[By Galois correspondence]} \\
& \Longrightarrow & \textbf{INFER} \ W \ (P \times Q) \ \ \trianglerighteq \ \mathcal{Z} \\
& & \text{[By theorem 4]} \\
& \Longrightarrow & (P \times Q) \ \textbf{s-sat}^{\text{w}} \ \ \mathcal{Z} \\
& & \text{[By definition 42]}
\end{array}$$

□

Chapter 6

Interfaces and Refinement

In this chapter two aspects of designing secure systems are examined. The first involves expressing confidentiality properties in a theory of typed machines. These typed machines are just labelled graphs but they enable a confidentiality specification to be related to a confidentiality statement in a calculus, such as CST, and to a system description such as a process in CSP. Typed machines provide a useful intermediate description in the task of verifying systems.

The second aspect of designing a secure system involves refinement. In section 6.2 the properties of functional refinement are examined with respect to confidentiality. A number of results concerning preservation of confidentiality and satisfaction of confidentiality properties are shown. One particular result gives a criteria for deciding whether a system exists which can satisfy a functionality and a confidentiality specification simultaneously.

6.1 Lifting information flow

All the categorical infrastructure built in this section is to prove theorems which are general results for the category $\mathcal{IF}$. The theorems allows a specification of confidentiality to be stated and checked in the static terms of possible states and operations rather than the dynamic terms of possible behaviours; this makes it easier to show that a system satisfies a confidentiality specification. The method for lifting statically expressed confidentiality specifications is that of adjunctions (see chapter 2 for the technical definition).

There is a well known adjunction between state machines and behaviours, [Goguen 72], which provides a link between information flow expressed over behaviours and its realization as a state machine. Other adjunctions exist between behaviours and linear and stochastic machines, these appeared first in [Goguen 72, Goguen 73a] and later in [Arib & Manes 74]. The following simple extended example illustrates the strategy explored in this section.

Example 22 Suppose X is the interface to some machine. There are only two possible outputs which can be observed at X, o_1 and o_2. The outputs at X are

determined by two users U_1 and U_2 only. The information which must be protected is 'which user determines which output'.

Suppose from an observer's knowledge of the construction of the machine it has a rule, *ded*, which from an output at X places limits on the observers' certainty about which user caused an output. A designer of a "secure" system could use the rule *ded* to thwart such an observer by constructing the machine in such a way so that *ded* satisfies a minimum amount of uncertainty associated with an output. Formally,

$$X = \{o_1, o_2\} \text{ and } Users = \{U_1, U_2\}$$

and

$$ded : X \rightarrow X \times Users$$

such that

$$\begin{aligned} ded(o_1) &= \{(o_1, U_1), (o_1, U_2)\} \\ ded(o_2) &= \{(o_2, U_1), (o_2, U_2)\}. \end{aligned}$$

The specification of *ded* asserts that if an output is observed then either user U_1 or U_2 could be responsible for that output which is what the observer knew anyway. The specification of *ded* has nothing to say about what an observer is allowed to deduce from a sequence of outputs. It is obvious how to extend *ded* to a sequence of outputs but it also provides a good illustration of how the information flow property of *ded* can be lifted to such a sequence.

There is a well known adjunction between the category of monoids, **Mon**, and the category of sets, **Set** (see pages 271 - 274 of [Barr & Wells 90]). The underlying functor U forgets the monoidal structure and gives the set underlying the monoid. The free functor F takes a set X to the free monoid (or Kleene closure of X), X^*, with sequence concatenation as the binary operator and the empty sequence, $\langle\rangle$, as the identity. Recall that by definition of an adjunction in chapter 2, an adjunction between **Mon** and **Set** means that for a set X and a function

$$u : X \rightarrow U(M)$$

(where M is any monoid) there is a unique monoid homomorphism

$$g : F(X) \rightarrow M$$

such that

$$u = \eta_X ; U(g)$$

where η_X is the unit of the adjunction.

To use this adjunction to lift the property *ded* it is noted that there is a monoid consisting of non-empty sets of sequences with the binary operator

$$\diamond : \mathbb{P}Y^* \times \mathbb{P}Y^* \rightarrow \mathbb{P}Y^*$$

such that

$$\forall A, B : \mathbb{P}Y^* \cdot A \diamond B = \{ x^\frown y \mid x \in A \wedge y \in B \}$$

and an identity

$$\{\langle\rangle\}.$$

The function

$$u : X \to \mathbb{P}(X \times Users)^*$$

is such that

$$\forall x : X \cdot u(x) = \{\, \langle y \rangle \mid y \in ded(x) \,\}$$

and the unit of the adjunction, η_X, is the simple mapping

$$\eta_X(x) = \langle x \rangle.$$

The definition of an adjunction gives an algebraic characterization of the unique monoid homomorphism, called $ded^\natural$, which lifts the property captured by ded to that property over sequences of outputs. Specifically

$$ded^\natural : X^* \to \mathbb{P}(X \times Users)^*$$

is the unique monoid homomorphism such that

$$u = \eta_X ; U(ded^\natural).$$

The monoid homomorphism $ded^\natural$ which satisfies this equation is unique by definition of the adjunction. The consequence of this is that for any sequence of outputs an observer cannot deduce which user was responsible for issuing the outputs. For example with the projection functions, pr_X and pr_{Users}

$$ded^\natural(\langle o_1, o_2, o_1 \rangle) = \left\{ t : (X \times \mathrm{Users})^* \;\middle|\; \begin{array}{l} \#t = 3 \\ \wedge \\ pr_X(t) = \langle o_1, o_2, o_3 \rangle \\ \wedge \\ pr_{\mathrm{Users}}(t) = \{U_1, U_2, U_3\}^* \end{array} \right\}$$

Thus the property captured by ded has been lifted to a property over output behaviours captured by $ded^\natural$. △

The previous example illustrates the way in which static confidentiality properties can be lifted to confidentiality statements. In the category $\mathcal{IF}$ a function like ded cannot be extended using structural induction, which is the alternative way of defining $ded^\natural$, since a structural induction will depend upon the model of computation used. Instead a category of typed machines, which has free typed machines, is constructed. The freeness property allows static confidentiality property over an interface to be lifted to behaviours observed via that interface. The strategy for the rest of this section is to define what a typed machine is and then the category of typed machines, **TMach**; a functor from a subcategory of **TMach** into $\mathcal{IF}$ is assumed which allows lifted static confidentiality properties over typed machines to be expressed as confidentiality properties over $\mathcal{IF}$. Finally it is shown that if a system satisfies a static confidentiality property in **TMach** then it will satisfy the lifted property also.

6.1.1 A category of typed machines

Definition 48 (Typed Machines) *A typed machine consists of a triple*

$$(G, I, E)$$

where G is a labelled directed graph, I is a set of initial nodes of the graph G and E is the set of events the machine is prepared to engage in. The graph G of a typed machine can have loops on each node representing a null computation, this is the null arc in the graph. ◇

Example 23 Syntactic processes in the CSP process algebra can be viewed as Typed Machines, for example the syntactic description

$$P \triangleq a \to b \to P$$

has the graph illustrated by figure 6.1.

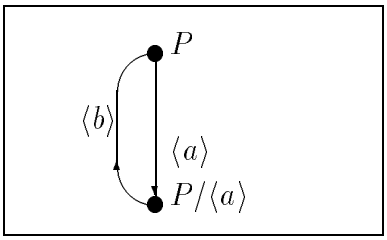


Figure 6.1: Syntactic graph of the CSP process $P \triangleq a \to b \to P$.

The syntactic graph consists of two nodes, P and $P/\langle a\rangle$, $P/\langle a\rangle$ is the new process $b \to P$ which comes from the process P after performing the event a. There is a directed arc from P to $P/\langle a\rangle$, labelled with the event a, and a directed arc from $P/\langle a\rangle$ to P, labelled with the event b. The null arc of P is the transition labelled $\langle\rangle$; $P/\langle\rangle$ is equal to P which is correct for a null computation. There is a similar null arc at $P/\langle a\rangle$. There is only one initial node, P and the set of events the typed machine is prepared to engage in is $\{a, b\}$.

The nodes correspond to syntactic processes, the arcs to following the syntactic arrow $\to$, "then".

△

Definition 49 (TMach) *The category* **TMach** *has as its objects typed machines and there is an arrow from a typed machine (G_1, I_1, E_1) to a typed machine (G_2, I_2, E_2) if and only if there exists a graph homomorphism*

$$h : G_1 \to G_2$$

such that the initial nodes I_1 are mapped to I_2. ◇

The category **TMach** has a rich structure, to map typed machines to an information flow category a simpler subcategory will be needed in general. For example for the concrete information flow category Safety-Iflow the subcategory of **TMach** is defined as follows.

Definition 50 (TMach$_{Safe}$) *The category* **TMach**$_{Safe}$ *has as its objects typed machines* (G, I, E) *such that the labels are singleton sets containing a trace, over the set of events* E*, or the empty set. A null arc is labelled by the empty trace,* $\{\langle\rangle\}$.

There is an arrow from a typed machine (G_1, I_1, E_1) *to a typed machine* (G_2, I_2, E_2) *if and only if there is a homomorphism from* G_1 *to* G_2 *such that* $E_2 \subseteq E_1$ *and each label of* G_1 *when restricted through* E_2 *is a subset of a label of* G_2. $\diamond$

Example 24 The syntactic graph of $P \triangleq a \rightarrow b \rightarrow P$ has an arrow to the syntactic graph of $Q \triangleq a \rightarrow Q$ in **TMach**$_{Safe}$ using the CSP restriction operator $\restriction$ over the alphabet a. The alphabet of the second graph of figure 6.2 is $\{a\}$ which is a subset of the alphabet of the first graph, which is $\{a, b\}$.

The arc labelled by $\{\langle a\rangle\}$ in the first graph of figure 6.2 maps to the arc labelled by $\{\langle a\rangle\}$ in the second graph of figure 6.2. The label remains the same under the restriction $\restriction \{a\}$. The arc labelled by $\{\langle b\rangle\}$ in the first graph of figure 6.2 is mapped to the null arc of the single node of the second graph. Each syntactic CSP process is assumed to have null arcs on each node, but for clarity they are suppressed in figure 6.2, except for the graph of Q. The label $\{\langle b\rangle\}$ when restricted to $\{a\}$ becomes $\{\langle\rangle\}$ which agrees with the label on the null arc. Under this mapping the two nodes P and $P/\langle a\rangle$ are mapped to the single node Q. The null arcs on each node of the first graph map to the null arc of the second graph.

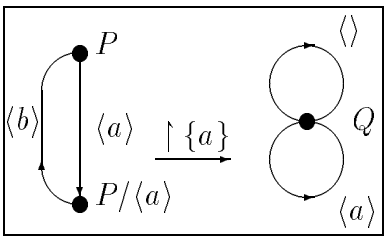


Figure 6.2: Restriction from $P \triangleq a \rightarrow b \rightarrow P$ to $Q \triangleq a \rightarrow Q$.

$\triangle$

It is assumed that there is a functor, B, from a subcategory of **TMach** to an information flow category $\mathcal{IF}$. For the category Safety-Iflow a subcategory of **TMach** has been defined. The functor B maps a typed machine in **TMach**$_{Safe}$ to an object in Safety-Iflow by concatenating all the labels on a path through the graph and taking the union of all the paths. The arrows of **TMach**$_{Safe}$ are mapped to trace restrictions over sets of traces with alphabet inclusion. First a path through a graph must be defined.

Definition 51 *A path through a graph is a traversal of the arcs of the graph from an initial node such that each arc in the path is traversed only once.* ◇

Example 25 Including the null arcs of the graph of the process P in example 23 the graph has two paths of length one: the null arc from P to P; and the arc labelled by $\langle a \rangle$ from P to $P/\langle a \rangle$. There are three paths of length two: the null arc from P to itself followed by the arc from P to $P/\langle a \rangle$; the arc from P to $P/\langle a \rangle$ followed by the arc from $P/\langle a \rangle$ to P; and the arc from P to $P/\langle a \rangle$ followed by the null arc from $P/\langle a \rangle$ to itself. Similarly there are three paths of length three and one of length four making nine paths altogether.

△

To define B over $\mathbf{TMach}_{Safe}$ a function over the arcs which comprise the paths is needed.

Definition 52 *In* $\mathbf{TMach}_{Safe}$ *if* $[n \to m]$ *is an arc from node* n *to node* m, $\hat{}$ *concatenates arcs and* **label** *is a function which gives the trace label associated with an arc then*

$$CAT([n \to m]) = label([n \to m])$$

and

$$CAT([n \to m] \hat{} s) = label([n \to m]) \frown CAT(s)$$

◇

Example 26 Applying the function CAT to a path of length three from example 25 gives

$$\begin{aligned} & CAT([P \to P] \hat{} [P \to P/\langle a \rangle] \hat{} [P/\langle a \rangle \to P/\langle a \rangle]) \\ &= \langle\rangle \frown CAT([P \to P/\langle a \rangle] \hat{} [P/\langle a \rangle \to P/\langle a \rangle]) \\ &= \langle\rangle \frown \langle a \rangle \frown CAT([P/\langle a \rangle \to P/\langle a \rangle]) \\ &= \langle\rangle \frown \langle a \rangle \frown \langle\rangle \\ &= \langle a \rangle \end{aligned}$$

△

With these auxiliary definitions the functor B over $\mathbf{TMach}_{Safe}$ can be defined.

Definition 53 *B_{safe} is defined over the paths of the graph of a typed machine. To avoid being bogged down in detail a function PATHS is assumed which gives all the paths of a graph G. The functor B_{safe} maps objects in* $\mathbf{TMach}_{Safe}$ *to objects in Safety-Iflow like so:*

$$B_{safe}(G, I, E) = (E, \bigcup_{t:PATHS(G)} \{CAT(t)\})$$

The arrows map over as defined by their source and target. ◇

Example 27 If P is the typed machine of example 23 then applying B_{safe} to P gives $\{\langle\rangle, \langle a \rangle, \langle a, b \rangle\}$. Note that there is no trace $\langle a, b, a, b \rangle$ because there are only two arcs, excluding the null arcs, and a path traverses an arc only once.

△

6.1.2 Static confidentiality properties

An observer of a system usually only sees part of the total system; the part of the system which is observable looks like a system in its own right. The observable part of a system is the interface between the total system and an observer. In this chapter the interface is a typed machine, paths through the graph of the typed machine correspond to observations. By mapping an interface to another typed machine confidentiality properties can be expressed. The expression of these properties using typed machines makes them easier to compare with a design but more difficult to understand in terms of high level requirements. Before showing how high level confidentiality statements can be related to typed machines, the expression of confidentiality properties over typed machines is defined.

Definition 54 *If $O = (G_o, I_o, E_o)$ is a typed machine representing an interface to a system then a typed machine $K = (G_k, I_k, E_k)$ is an inference object for O if and only if there is a surjective homomorphism from G_k to G_o and a surjection from E_k to E_o.* ◇

The mapping from I_k to I_o is completely determined by the mappings from G_k to G_o and E_k to E_o.

Example 28 In example 24 the syntactic graph of the process

$$P \triangleq a \rightarrow b \rightarrow P$$

has a surjective homomorphism to the syntactic graph of the process

$$Q \triangleq a \rightarrow Q$$

and

$$f \triangleq \{a \mapsto a, b \mapsto a\}$$

is a surjection from E_P to E_Q. If Q represented the interface of a system where only an a operation could be performed then P could be an inference object for Q.

△

Example 29 If the interface is again

$$Q \triangleq a \rightarrow Q$$

and the inference object is

$$R \triangleq (b \rightarrow R) [] (a \rightarrow R)$$

then R is an inference object of Q and the b event non-interferes with the a event in R. Informally the interface Q is a projection of the inference object R so that when an a event occurs there is no way of inferring whether a b event has occurred or if it has, whether it occurred before or after the a event. This is a static confidentiality property because the graph of the interface Q has no arcs denoting more than one a operation, similarly the inference object only represents single operations. Q has one state and one transition. R has one state and two possible transitions.

△

To formally establish the dynamic property of non-interference, observations of the interface and their image in the inference object must be mapped to Safety-Iflow and compared with the appropriate confidentiality statement. From these static representations dynamic behaviours can be generated, this is common in computer science where a finite state transition diagram can generate an arbitrary number of transitions. The next proposition is important in constructing confidentiality statements from typed machines.

Proposition 8 *If there is a surjective homomorphism from a graph G_1 to a graph G_2 then there is a graph homomorphism from G_2 to G_1.*

Proof If h is the surjective homomorphism then for each arc, s, in the graph G_2 the inverse image of h can be taken. The arc s can then be mapped to an arc selected from $h^{-1}(s)$. In this way s is embedded into $h^{-1}(s)$ a subgraph of G_2. □

The subgraph $h^{-1}(s)$ of G_2 will represent the inference associated with an observation of an interface.

6.1.3 Lifting static properties

To lift static properties to dynamic behaviours an adjunction between the category of graphs and the category of small categories is used.

Definition 55 (Free typed machines) *If U is the forgetful functor from the category of small categories,* **Cat**, *to the category of small graphs,* **Grph**, *then let*

$$F : \mathbf{Grph} \rightarrow \mathbf{Cat}$$

be the left adjoint of U.

If T is a typed machine then $(UF(G), I, E)$ is the free typed machine generated by T and is denoted by $T^\natural$. ◇

Each graph in **Grph** can be viewed as a diagram of objects (nodes) and arrows (arcs) like a diagram for a category. In a graph composite arrows and identity arrows are not present. A graph G is a precategory and $F(G)$ is the closure of G with respect to arrow composition and identities. The forgetful functor U maps $F(G)$ back to a graph which has the same nodes (objects) and arcs (arrows) as $F(G)$, but which arcs are composites or identities are forgotten. An arc a_{12} which is the composite of the arcs a_1 and a_2 is present in $UF(G)$ but the information that

$$a_{12} = a_1 \hat{} a_2$$

(where $\hat{}$ is arc concatenation) is forgotten. Similarly arcs which were identity arrows are present, but when concatenated with other arcs cannot be eliminated like identity arrows. A free typed machine $T^\natural$ is simply the typed machine obtained by performing the closure on the graph of the typed machine T. The labels on the arcs of a typed machine preserve the information about which arcs were composites and which were identities. In $\mathbf{TMach}_{Safe}$ it is assumed that the label for an identity arrow is the empty trace, $\langle\rangle$, which is the label for the null arc.

Example 30 If the graph of figure 6.3 is the graph of a typed machine T then the graph of figure 6.4 represents the infinite graph of the free typed machine $T^\natural$. Unfortunately the graph is to large to fit on a page and only the new arcs labelled by $\langle\rangle$ and $\langle a, a, a, a\rangle$ are shown in figure 6.4.

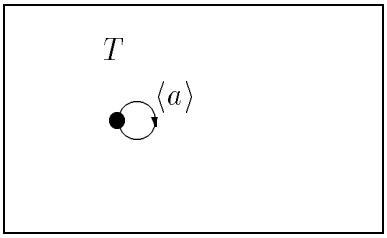


Figure 6.3: The graph of the typed machine T.

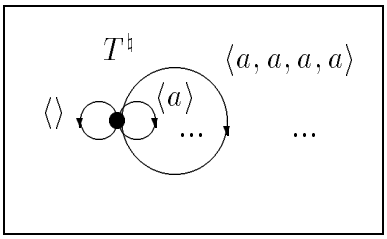


Figure 6.4: The graph of the typed machine $T^\natural$.

$\triangle$

Definition 56 *An observation of an interface, O, is a typed machine with a graph consisting of a single non-null arc of the typed machine $O^\natural$.* $\diamond$

Proposition 9 *If O is an interface and K is an inference object and*

$$h : K \to O$$

is the projection of the inference object K onto O then $h^\natural$ is the lifted arrow from $K^\natural$ to $O^\natural$.

Proof By the property of an adjunction. $\square$

Lemma 22 *If t is a path through the graph of an interface $O^\natural = (G_o^\natural, I_o, E_o)$, $K^\natural = (G_k^\natural, I_k, E_k)$ is an inference object, $h^\natural$ is the lifted projection of K onto O which has a surjective graph homomorphism*

$$h_1^\natural : G_k^\natural \to G_o^\natural$$

then there is an arrow

$$k : O^\natural \to K^\natural$$

such that t *is mapped to the graph* $h_1^{\natural -1}(t)$

Proof

The graph homomorphism part of the projection $h^\natural$ gives rise to a graph homomorphism which embeds t into the graph $h_1^{\natural -1}(t)$ by proposition 8. There may be many embeddings into $h_1^{\natural -1}(t)$ but which is used is of no interest. The initial nodes and events of O can similarly be embedded in the inverse image of $h^\natural$. □

Lemma 23 *If* l *is an observation of* O *and* $i(l)$ *is the inclusion of* l *into* O *then* $B_{safe}(i(l)) = B_{safe}(l)$ *in* $\mathbf{TMach}_{Safe}$.

Proof

l has a single arc with a trace label x say, there are no empty arcs. The inclusion of l into O leaves the arc and label unchanged, therefore by the definition of B_{safe} over $\mathbf{TMach}_{Safe}$

$$B_{safe}(i(l)) = \{x\} = B_{safe}(l)$$

□

An observation of an interface can be included in the free typed machine generated by the interface and then mapped to the subgraph of the inference object which projects onto that embedded observation.

Theorem 10 *If* O *and* K *are an interface and its inference object in* **TMach** *and* i *is an inclusion arrow and*

$$k : O^\natural \to K^\natural$$

is the arrow of lemma 22 in **TMach** *and there is an arrow from* $B(i(l))$ *to* $B(l)$ *then the functor* $\mathcal{S}$ *such that for all observations* l *of* O

$$\mathcal{S}(B(l)) = B(k(i(l)))$$

is a confidentiality statement in Iflow-order.

Proof $i(l)$ is a path of O, k maps $i(l)$ to the subgraph g of the graph of K, such that g was projected onto $i(l)$. The initial node of $i(l)$ maps to the single initial node of g and the alphabet of O is mapped to the alphabet of K by inclusion. The subgraph g has a graph homomorphism to $i(l)$ by restricting the graph homomorphism (from G_k to G_o) to g. Therefore there is an arrow from $B(k(i(l)))$ to $B(i(l))$ in Iflow-order.

By hypothesis there is an arrow from $B(i(l))$ to $B(l)$, therefore there is also an arrow from $B(k(i(l)))$ to $B(l)$ in Safety-Iflow. These arrows form the components of a natural transformation from $\mathcal{S}$ to the identity in Iflow-order which by definition 33 means that $\mathcal{S}$ is a confidentiality statement. □

Corollary 7 *If O and K are an interface and its inference object in* **TMach**$_{Safe}$ *and i is an inclusion function and*

$$k : O^\natural \to K^\natural$$

is the arrow of lemma 22 in **TMach** *then the function $\mathcal{S}$ such that for all observations l of O*

$$\mathcal{S}(B_{safe}(l)) = B_{safe}(k(i(l)))$$

is a confidentiality statement in Safety-Iflow.

Proof

By lemma 23 $B_{safe}(i(l)) = B_{safe}(l)$, therefore the identity arrow goes from $B_{safe}(i(l))$ to $B_{safe}(l)$ and by theorem 10 $\mathcal{S}$ is a confidentiality statement over Safety-Iflow. □

Example 31 Using the interface

$$Q \triangleq a \to Q$$

and inference object

$$R \triangleq (b \to R) [\!] (a \to R)$$

of example 29 then there is an arrow

$$h : R \to Q$$

in **TMach**$_{Safe}$ which is uniquely determined by the alphabet restriction over the trace labels. This arrow becomes the unique arrow

$$h^\natural : R^\natural \to Q^\natural$$

which is depicted in figure 6.5 along with the inclusion of the observation of $\langle a \rangle$. B_{safe} gives the traces of the typed machines, in terms of these traces $h^\natural$ is the restriction $\restriction \{a\}$ and k is the inverse relation.

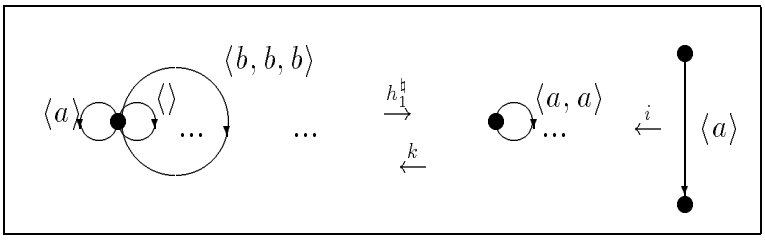


Figure 6.5: The subgraph of $R^\natural$ consistent with an a event.

The mapping depicted in figure 6.5 is expressed by

$$\{\, tr : \{a, b\}^* \mid tr \restriction \{a\} = \langle a \rangle \,\} \overset{\restriction \{a\}}{\to} \{\langle a \rangle\}$$

and the function k maps $\{\langle a \rangle\}$ to $\{\, tr : \{a,b\}^* \mid tr \upharpoonright \{a\} = \langle a \rangle \,\}$. In general if $l : \{a\}^*$ then $\{l\}$ is mapped to

$$\{\, tr : \{a,b\}^* \mid tr \upharpoonright \{a\} = l \,\}$$

This gives the semantic confidentiality function

$$\forall l : \{a\}^* \cdot \mathcal{S}(\{a\}, \{l\}) = \{\, tr : \{a,b\}^* \mid tr \upharpoonright \{a\} = l \,\}$$

in terms of the syntax of confidentiality statements this is represented by

$$\textbf{From } l : \{a\}^* \textbf{ MAY\_INFER } tr \upharpoonright \{a\} = l \textbf{ Over } tr : \{a,b\}^*$$

△

6.1.4 Verifying systems

Proposition 10 *If O and K are an interface and its inference object in* **TMach***; i is an inclusion arrow from an observation to O; and there is an arrow from $B(i(l))$ to $B(l)$ then for all observations l of O*

$$\mathcal{S}(B(l)) = B(c^\natural(i(l)))$$

is a confidentiality statement in Iflow-order, where c is an arrow induced by O and K.

Proof If K is an inference object of O then there is a projection, h, of K onto O. By lemma 22 there is a graph homomorphism

$$c_1^\natural = h_1^{\natural -1}$$

which defines the graph homomorphism component of c. A function can always be found from the alphabet of O to the alphabet of K which will be the function over alphabets component of the arrow c, therefore by theorem 10 $\mathcal{S}$ is a confidentiality statement in Safety-Iflow. □

Corollary 8 *If O and K are an interface and its inference object in* $\textbf{TMach}_{Safe}$*, c is an arrow induced by O and K and i is an inclusion arrow from an observation to O then for all observations l of O*

$$\mathcal{S}(B_{safe}(l)) = B_{safe}(c^\natural(i(l)))$$

is a confidentiality statement in Safety-Iflow, where c is a function derived from O and K.

Proof

By corollary 7 and proposition 10. □

Example 32 If

$$O \triangleq a \rightarrow O$$

and

$$K \triangleq a \rightarrow K [\!] b \rightarrow K$$

then K is an inference object of O since

$$K \upharpoonright \{a\} = O$$

and c is the arrow from O to K where $\{a\}$ is included into $\{a, b\}$ and the graph of O is mapped to the isomorphic subgraph of K. $\triangle$

Definition 57 *P is a system in* **TMach** *if and only if $B(P^\natural)$ is a system in $\mathcal{IF}$.* $\diamond$

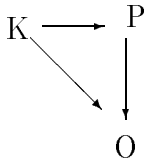
Example 33 If $P \triangleq a \rightarrow P$ then

$$B_{Safe}(P^\natural) = (\{a\}, \{a\}^*)$$

which is non-empty and prefix closed, due to the closure $P^\natural$, and therefore P is a system in Safety-Iflow and hence in $\mathbf{TMach}_{Safe}$.

The typed machine Q with the same graph as P but labelled by the trace $\langle a, b \rangle$ is not a system in $\mathbf{TMach}_{Safe}$ since $B_{Safe}(Q^\natural)$ is not prefix closed. $\triangle$

Theorem 11 *If O is an interface object, K is an inference object, c is the arrow induced by O and K, there is an arrow from $B(i(l))$ to $B(l)$ and the following diagram commutes*



then

$$P \ \textbf{s-sat}^{W} \ \mathcal{S}$$

where W is the set of set of observations of O and for all observations l in W

$$\mathcal{S}(B(l)) = B(c^\natural(i(l)))$$

Proof If there is an arrow from K to P then by the property of an adjunction there is an arrow from $K^\natural$ to $P^\natural$ and arrows from both to $O^\natural$ which makes the obvious diagram commute. In particular for each observation l in W there is an arrow from

$c^\natural(i(l))$ to $P^\natural$ via its inclusion in $K^\natural$, where i is the inclusion of l into $O^\natural$. There is also an arrow from $B(c^\natural(i(l)))$ to $B(P^\natural)$ because B is a functor and functors preserve arrows. Similarly there are arrows to $B(i(l))$ which commute because the diagram above commutes.

By proposition 10 $\mathcal{S}$ is a confidentiality statement and has an arrow from $\mathcal{S}(B(l))$ to $B(l)$ and by the argument above has an arrow from $\mathcal{S}(B(l))$ to $B(P^\natural)$; therefore by the definition of product there is an arrow from $\mathcal{S}(B(l))$ to $B(P^\natural) \times B(l)$ and arrows to $B(l)$ which makes the obvious diagram commute for every observation l. By definition 42 in chapter 5 this means that

$$P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$$

□

Corollary 9 *If O is an interface object, K an inference object, c the arrow induced by O and K and there is an arrow from K to a system P then*

$$P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$$

where W is a set of set of observations of O and for all observations l in W

$$\mathcal{S}(B_{Safe}(l)) = B_{Safe}(c^\natural(i(l)))$$

Proof

By lemma 23 there is an arrow from $B_{Safe}(i(l))$ to $B_{safe}(l)$, Safety_Iflow is a preorder which means that the arrows are unique consequently the diagram of theorem 11 commutes because. This means that by theorem 11

$$P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$$

□

Example 34 In this example conditional non-interference between a set of interactions $\{a\}^*$ and the set of interactions $\{b\}^*$. The interface is described by the CSP process

$$O \ \triangleq \ (a \rightarrow O)[\!](left \rightarrow O_s)$$
$$O_s \ \triangleq \ (a \rightarrow O_s)[\!](right \rightarrow O)$$

This interface has a secure session which starts with the event $left$ and ends with the event $right$. The inference object associated with this interface is described by the CSP process

$$K \ \triangleq \ (\phi \rightarrow K)[\!](left \rightarrow K_s)$$
$$K_s \ \triangleq \ (a \rightarrow O_s)[\!](b \rightarrow K_s)[\!](right \rightarrow O)$$

It is not possible to construct a graph with an arc labelled by the empty set in CSP so a special label ϕ is introduced to denote this. The inference object K legislates that during a secure session it must be possible for there to be any number of b events occurring before or after an a event. Outside a secure session it does not require anything of a system. The system

$$P \triangleq (a \rightarrow P) [\!] (b \rightarrow a \rightarrow P) [\!] (left \rightarrow P_s)$$
$$P_s \triangleq (a \rightarrow P_s) [\!] (b \rightarrow P_s) [\!] (right \rightarrow P)$$

satisfies the confidentiality statement which arises from the confidentiality function from O to K.

$\triangle$

6.2 Refinement and confidentiality

At the moment refinement is a costly process in time and expertise, because of this it seems to be most suited to the production of safety-critical or secure systems where the cost can be justified. It is in the production of secure systems however where the use of functional refinement appears to fail. It is possible to specify a design of a system that exhibits some confidentiality properties and to refine that specification, using standard laws of refinement, into one that violates the confidentiality properties.

Example 35 If

$$P \triangleq a \rightarrow P \text{ and } Q \triangleq b \rightarrow Q$$

then for the system $P \| Q$ an agent who can only see b events cannot tell what a events might have occurred. If

$$S \triangleq a \rightarrow b \rightarrow S$$

then S is a refinement of $P \| Q$ in CSP, but the same agent seeing a b event knows that the event a must have occurred. The refinement has transformed the system $P \| Q$ into a less "secure" one. $\triangle$

The other standard forms of refinement, for example [Morgan 90], can also lead to "insecure" systems, this has been examined in [Jacob 89b]. One of the goals of specification is to liberate a designer of a system from considering **how** a system goes about achieving **what** is required. Unfortunately one of the most popular ways of specifying a system's confidentiality is to state **how** it functions and then assert that this means the system is secure. If refinement is used to produce the implementation then problems can arise. If the specification is behaviourally equivalent to the implementation then one of the main goals of specification, abstraction, may have been lost.

Fortunately refinement and confidentiality can be reconciled, for example in [G-C & Sanders 91] a sufficient condition for a refinement to preserve the confidentiality property of non-interference [Goguen & Meseguer 82] is proved. In some situations non-interference is too restrictive, see chapter 8, and in others it is too weak,

therefore a way of specifying general confidentiality properties and their behaviour under refinement is required. In this section a framework for refinement relative to this constraint is defined and some initial results on valid refinements and their composition are established. The abstract categorical approach is applicable to many models of computation such as timed traces, refusals and Petri Nets.

In subsection 6.2.1 some simple conditions for when a system satisfies a confidentiality specification are proved. In addition various results are proved about how a system, which satisfies a functional and confidentiality specification, can be composed with other such systems. These results indicate that the construction of a large system with confidentiality properties can be broken down into a number of simpler components which can then be assembled together.

6.2.1 Refinement

In this section the composition of systems, functional and confidentiality specifications is considered with respect to refinement. A proposition is proved which gives a necessary condition for a system to satisfy functionality and confidentiality. The term functionality is used loosely, functionality can be a safety specification or include liveness, and timeliness constraints.

Assumption It is assumed for the rest of this chapter that functional specifications as well as systems and observations are objects in an Information flow category.

A safety specification in Safety-Iflow is just a non-empty set of traces, in the categories for timed traces and refusals a "functional" specification is a set of timed traces and set of trace refusal pairs respectively; these functional specifications reflect timeliness and liveness constraints respectively. In the rest of the section results are given about how systems respect confidentiality and functionality under product and coproduct. The refinement discussed in this section concerns functional rather than information flow refinement. To state the proposition concerning satisfaction of functionality and confidentiality the following auxiliary definition is made.

Notation 7 *If $\mathcal{IF}$ has a preorder subcategory, denoted $\mathcal{IF}_{\sqsubseteq}$, which represents functional refinement then $P \sqsubseteq Q$ denotes the unique arrow from the object Q to the object P.*

$P \not\sqsubseteq Q$ denotes the fact that there are no arrows from the object Q to the object P.

If P and Q are systems and S is a functional specification then

$$P \sqsubseteq Q$$

means that Q is better than P, or Q is a refinement of P.

$$P \textbf{ sat } S$$

means that $S \sqsubseteq P$. This follows the convention in CSP where if $traces(Q) \subseteq traces(P)$ then $P \sqsubseteq Q$ or if the failures of Q are a subset of the failures of P then again

$P \sqsubseteq Q$. For Petri nets the existence of an implementation morphism (as defined in [Mess. & Mont. 88]) from N to N' reflects a refinement. The following proposition generalises a result due to He Jifeng, [He 89].

Proposition 11 *If* Func *is a functional specification,* $\mathcal{S}$ *a confidentiality statement and* P **sat** Func*, that is* Func $\sqsubseteq P$ *then*

$$\begin{array}{l} \exists l : \mathrm{Func}_W \cdot \mathrm{Func} \not\sqsupseteq \mathcal{S}(l) \\ \Longrightarrow \\ \nexists P \cdot l \in P_W \ \wedge \ P \ \mathbf{sat} \ \mathrm{Func} \wedge P \ \ \mathbf{s\text{-}sat}^{\mathrm{W}} \ \ \mathcal{S} \end{array}$$

that is if there is an observation, l, for which there are no arrows from $\mathcal{S}(l)$ to Func *then there is no system P, with the observation l, which can satisfy both the functional and confidentiality specifications.*

Proof Suppose for a contradiction that $\mathrm{Func} \not\sqsupseteq \mathcal{S}(l)$ but

$$\exists P \cdot l \in P_W \wedge \mathrm{Func} \sqsubseteq P \wedge P \ \ \mathbf{s\text{-}sat}^{\mathrm{W}} \ \ \mathcal{S}.$$

Now

$$\mathrm{Func} \sqsubseteq P \Longrightarrow \mathrm{Func} \times l \sqsubseteq P \times l \tag{6.1}$$

by definition of product, but

$$\begin{array}{l} P \ \ \mathbf{s\text{-}sat}^{\mathrm{W}} \ \ \mathcal{S} \ \ \Longrightarrow (P_W \ \triangleright \mathcal{S}) \overset{\bullet}{\rightarrow} (P_W \ \triangleright \lceil P \times \_ \rceil) \\ \qquad \text{[By definition of } \ \mathbf{s\text{-}sat}^{\mathrm{W}} \ .] \\ \qquad \Longrightarrow \forall l : P_W \cdot P \times l \sqsubseteq \mathcal{S}(l) \\ \qquad \text{[By definition of } \overset{\bullet}{\rightarrow}.] \\ \qquad \Longrightarrow \forall l : P_W \cdot \mathrm{Func} \times l \sqsubseteq \mathcal{S}(l) \\ \qquad \text{[By 6.1 above and transitivity of } \sqsubseteq.] \\ \qquad \Longrightarrow \forall l : P_W \cdot \mathrm{Func} \sqsubseteq \mathcal{S}(l) \\ \qquad \text{[By definition of product.]} \end{array}$$

that is there is always and arrow from $\mathcal{S}(l)$ to Func, this contradicts the hypothesis, therefore proving the proposition. □

This proposition tells the designer that if they find an observation, such that the allowable inference associated with that observation has no arrow to the object representing the desired functionality of a system, then that observation must be removed by the refinement.

Example 36 Consider the following simple confidentiality specification

From $\langle\rangle : \{x, y\}^*$ **MAY_INFER** $tr \in \{\langle\rangle, \langle b\rangle\}$ **Over** $tr : \{x, y, b\}^*$
From $\langle x\rangle : \{x, y\}^*$ **MAY_INFER** $tr \in \{\langle x\rangle, \langle b, x\rangle\}$ **Over** $tr : \{x, y, b\}^*$
From $\langle y\rangle : \{x, y\}^*$ **MAY_INFER** $tr \in \{\langle y\rangle, \langle y, b\rangle\}$ **Over** $tr : \{x, y, b\}^*$

If the system

$$P \triangleq (b \to x \to STOP \mid y \to b \to STOP)$$

is taken as the functional specification then this is represented by the object

$$P' = (\{x, y, b\}, \{\langle\rangle, \langle b\rangle, \langle b, x\rangle, \langle y\rangle, \langle y, b\rangle\}).$$

The observation $(\{x, y\}, \{\langle x\rangle\})$ is mapped, by the above confidentiality statement, to the object $(\{x, y, b\}, \{\langle x\rangle, \langle b, x\rangle\})$, but there is no arrow from this to P' since $\langle x\rangle \notin P'$. Any refinement of P cannot allow the observation $(\{x, y\}, \{\langle x\rangle\})$ if it is to satisfy the above confidentiality specification. The system

$$Q \triangleq (b \to STOP \mid y \to b \to STOP)$$

is a refinement of P which does not have the offending observation and by inspection satisfies the above confidentiality specification when it is restricted to the observations $Q_{\{\{x,y\}\}}$, that is $(\{x, y, b\}, \{\langle\rangle\})$ and $(\{x, y, b\}, \{\langle y\rangle\})$. $\triangle$

A functional refinement relation generally reflects the fact that one system is more constrained or predictable. This means that one is more certain about what a system will do, the opposite of confidentiality. The conflict between confidentiality and functional refinement has been described in [Jacob 89b]. Restricting our attention to preorder categories leads to some simple but very useful results.

Theorem 12 *In $\mathcal{IF}_{\sqsubseteq}$ if $S \sqsubseteq P$ and $T \sqsubseteq Q$ are refinements of S by P and T by Q respectively then the usual refinement properties hold:*

i) $S \sqsubseteq P \wedge T \sqsubseteq Q \Longrightarrow S \times T \sqsubseteq P \times Q$

ii) $S \sqsubseteq P \wedge T \sqsubseteq Q \Longrightarrow S + T \sqsubseteq P + Q$

Proof **i)** and **ii)** follow by definition of categorical product and coproduct. $\square$

This is important because it allows the construction of large functional specifications and systems from simpler components via the categorical operators of product and coproduct. In concrete categories product and coproduct turn out to be familiar operators. The construction of large systems from simpler components applies to confidentiality as well as functionality.

Corollary 10

$$S \sqsubseteq P \ \wedge \ T \sqsubseteq Q \ \wedge \ P \ \textbf{s-sat}^{\text{W}} \ \mathcal{S} \ \wedge \ Q \ \textbf{s-sat}^{\text{W}} \ \mathcal{T}$$

$$\Longrightarrow$$

$$S \times T \sqsubseteq P \times Q \ \wedge \ P \times Q \ \textbf{s-sat}^{\text{W}} \ \mathcal{S} \wedge_{\text{IF}} \mathcal{T}$$

Proof

By theorem 12 and 4. $\square$

As indicated the importance of product and coproduct in the results given lies in their instantiation in concrete categories. The construction of concrete categories should be done in such a way that coproduct and especially product correspond to some meaningful construction. In the category of timed traces categorical product corresponds to parallel composition; in the refusals category product is not quite parallel composition, but from the point of view of reasoning about confidentiality properties it is equivalent; to be precise there is a natural transformation between both product and parallel composition, when they are used as functors for determining the inference from an observation. In [Mess. & Mont. 88] the categorical product of two Petri nets corresponds to their parallel composition and coproduct to non-deterministic choice. This category is of particular theoretical interest because, by applying appropriate constraints, the semantics of a range of process algebras can, in principle, be modelled by Petri nets. An example of a system satisfying a confidentiality specification is the following.

Example 37 The confidentiality specification of subsection 4.3.1 is repeated for clarity. If *Input* denotes messages on a input channel and *Conf* and *Unclass* denote messages on confidential and unclassified channels then a functional specification for a message filter is that it behaves like a buffer.

$$BUFFER\_SPEC \triangleq tr \upharpoonright (Conf \cup Unclass) \leq tr \upharpoonright Input$$

The messages input are output, via the channels *Conf* and *Unclass*, in the same order and without anything being added or removed. For convenience the set of messages on the confidential and unclassified channel are grouped together.

$$CU \triangleq Conf \cup Unclass \text{ and } CUI \triangleq Conf \cup Unclass \cup Input$$

An object of Safety-Iflow which corresponds to the buffer specification is

$$S \triangleq (CUI, \{ tr : CUI^* \mid BUFFER\_SPEC \})$$

S establishes what the functional behaviour of a message filter is but says nothing about its confidentiality properties. If

$$CONSISTENCY \triangleq tr \upharpoonright Unclass = l$$

is the predicate which links possible inferences back to the trace of an observation l; and

$$CONFIDENTIALITY \triangleq tr \upharpoonright Conf \in Conf^*$$

is the predicate which states that the most that can be inferred is that some trace of confidential communication has taken place (the predicate is equivalent to *true*, but is written as above to indicate that it is confidential messages that are of interest); and

$$FILT\_INFERENCE \triangleq CONSISTENCY \wedge CONFIDENTIALITY$$

then a confidentiality specification for a message filter is

$$\mathcal{S} \triangleq \textbf{From } l : Unclass^* \textbf{ MAY\_INFER } FILT\_INFERENCE \textbf{ Over } tr : CU^*$$

This confidentiality specification says that the most that can be inferred from an observation of events visible to an unclassified user is that any number of confidential events might have occurred. Given the safety and confidentiality specifications, candidate systems can be checked against them. If

$$SECURE\_FILTER \triangleq input?x \rightarrow \left(\begin{array}{l} y \rightarrow conf!x \rightarrow SECURE\_FILTER \\ [\!] \\ n \rightarrow unclass!x \rightarrow SECURE\_FILTER \end{array} \right)$$

then *SECURE_FILTER* receives a message and depending on some external agent, which decides whether a message is confidential or not, and outputs the message on the confidential or unclassified channel. Note that if desired the external agent's actions can be specified separately and conjoined with the buffer specification.

Hiding the events $\{y, n\}$ which determine whether a message is confidential or not gives a system which satisfies the buffer specification.

$$SECURE\_FILTER \backslash \{y, n\} \textbf{ sat } S$$

Because the confidentiality specification is in terms of only confidential and unclassified events it is necessary to restrict *SECURE_FILTER* appropriately. It can be shown that the restricted system satisfies the confidentiality specification. Another way of showing that this satisfaction holds will be demonstrated later in this section.

$$SECURE\_FILTER \upharpoonright CU \textbf{ s-sat}^{\{\text{Unclass}\}} \mathcal{S}$$

$\triangle$

Example 38 Suppose a buffer is required which may store no more than a hundred messages. A specification for such a buffer is

$$FIN\_BUFFER\_SPEC \triangleq tr \upharpoonright (Rest \cup Unclass) \leq^{100} tr \upharpoonright In$$

where *Rest* is the set of messages on a restricted channel and *Unclass* is the set of messages on the unclassified channel of example 37. For convenience the following abbreviation is made.

$$RUI \triangleq Rest \cup Unclass \cup In$$

The object denoted by the specification *FIN_BUFFER_SPEC* is

$$T \triangleq (RUI, \{\, tr : RUI^* \mid FIN\_BUFFER\_SPEC \,\})$$

To define the confidentiality specification, in this case, it is necessary to define some auxiliary functions on traces. The first function selects the odd numbered elements from a sequence of events.

$$odd(\langle\rangle) = \langle\rangle$$
$$odd(l^\frown\langle u\rangle) = \begin{cases} odd(l)^\frown\langle u\rangle & If\ \#l\ is\ even \\ odd(l) & else \end{cases}$$

For example

$$odd(\langle u, v, w\rangle) = \langle u, w\rangle.$$

The counterpart of the function odd is the function $even$.

$$even(\langle\rangle) = \langle\rangle$$
$$even(l^\frown\langle u\rangle) = \begin{cases} even(l)^\frown\langle u\rangle & If\ \#l\ is\ odd \\ even(l) & else \end{cases}$$

For example

$$even(\langle u, v, w\rangle) = \langle v\rangle.$$

The final function on traces which is used in the confidentiality specification is $\downarrow$ which is defined and used in [Hoare 85]. The function $\downarrow$ *channelname* restricts a trace to messages on the named channel. For example

$$\langle rest.x, rest.y, unclass.x\rangle \downarrow rest = \langle x, y\rangle.$$

These functions are composed together in the confidentiality specification to act on traces, for example

$$\begin{aligned} odd(\langle rest.x, rest.y, unclass.x\rangle \downarrow rest) &= \langle x\rangle \\ &= odd(\langle unclass.x\rangle \downarrow unclass) \end{aligned}$$

and

$$even(\langle rest.x, rest.y, unclass.x\rangle \downarrow rest) = \langle y\rangle$$

The confidentiality specification consists of three requirements.

$$CONSISTENT \triangleq tr \upharpoonright Unclass = l$$

The first requirement insists that any inferred behaviour must be consistent with an observation, although l is free in this predicate it will be bound in the actual specification.

$$KNOWN \triangleq odd(tr \downarrow Rest) = l \downarrow Unclass$$

The second requirement insists that the first message passed along the restricted channel, and then every odd numbered communication, must be consistent with the message passed down the unclassified channel.

$$RESTRICT \triangleq even(tr \downarrow Rest) \in Mess^*$$

The final requirement is that any message can be passed down the restricted channel and that it cannot be influenced by the communications on the unclassified channel. Putting these together gives the following confidentiality specification.

$$\mathcal{T} \triangleq \left\{ \begin{array}{l} \textbf{From } l : \mathit{Unclass}^* \\ \textbf{MAY\_INFER } \mathit{CONSISTENT} \wedge \mathit{KNOWN} \wedge \mathit{RESTRICT} \\ \textbf{Over } tr : \mathit{RU}^* \end{array} \right.$$

where $RU \triangleq Rest \cup Unclass$

The confidentiality specification states that when a message is output on the unclassified channel then one can infer that two messages may have been input, but one cannot infer the value of the first message received on the input channel. All that can be inferred about it is that it can be output along the restricted channel. This specification allows some inference to be made about messages output on the restricted channel but places limits on it. Note also that the restricted channel interferes with the unclassified channel. If

$$\mathit{R\_FILTER} \triangleq (in?x \rightarrow in?y \rightarrow rest!x \rightarrow rest!y \rightarrow unclass!x \rightarrow \mathit{R\_FILTER})$$

then

$$\mathit{R\_FILTER} \textbf{ sat } T \wedge (\mathit{R\_FILTER} \upharpoonright RU) \textbf{ s-sat}^{\{\mathrm{Unclass}\}} \mathcal{T}$$

R_FILTER will be proved to satisfy $\mathcal{T}$ later in example 39. By corollary 10 and by definition of parallel composition the system *SECURE_FILTER*, defined in example 37, can be taken and placed in parallel with *R_FILTER* and know that $\mathit{SECURE\_FILTER} \| \mathit{R\_FILTER}$ satisfies the specifications S and T and the confidentiality specifications $\mathcal{S}$ and $\mathcal{T}$. That is

$$\begin{array}{l} (\mathit{SECURE\_FILTER} \backslash \{y, n\}) \| \mathit{R\_FILTER} \\ \quad \textbf{sat} \\ S(tr \upharpoonright \alpha \mathit{SECURE\_FILTER}) \wedge T(tr \upharpoonright \alpha \mathit{R\_FILTER}) \end{array}$$

and

$$(\mathit{SECURE\_FILTER} \upharpoonright CU) \| (\mathit{R\_FILTER} \upharpoonright RU) \textbf{ s-sat}^{\{\mathrm{Unclass}\}} \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T}.$$

In Safety-Iflow $\mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T}$ can be combined to give one confidentiality statement.

$$\mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T} = \left\{ \begin{array}{l} \textbf{From } l : \mathit{Unclass}^* \\ \textbf{MAY\_INFER } \mathit{FILT\_INFERENCE} \wedge \mathit{REST\_INFERENCE} \\ \textbf{Over } tr : (RU \cup CU)^* \end{array} \right.$$

where

$$\mathit{REST\_INFERENCE} \triangleq \mathit{CONSISTENT} \wedge \mathit{KNOWN} \wedge \mathit{RESTRICT}$$

$\triangle$

Expressing a confidentiality requirement as a function over a set of observations is a natural way of specifying confidentiality, especially if the confidentiality property is discretionary. To compare a system or functional specification with such a confidentiality specification means that the system (or functional specification) has to be used to define a function for determining actual inference. Although this can be done, and leads to tractable proof techniques, it is also possible to collapse a confidentiality specification down into a single object, which (although an unintuitive representation of confidentiality) is simpler to compare with systems and functional specifications. The following theorem shows that confidentiality can be reasoned about on the same level as systems, observations and functional specifications.

Theorem 13 *In $\mathcal{IF}_{\sqsubseteq}$ if W is a set of subsets of the alphabet of P and $\mathcal{S}$ is a confidentiality statement then*

$$P \sqsubseteq \bigvee \mathcal{S}(\!|\, P_W \,|\!) \Longrightarrow P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S}$$

where $\bigvee$ denotes least upper bound (as introduced in chapter 3).

Proof By definition of least upper bound (or coproduct)

$$\forall o : O \cdot P \sqsubseteq \mathcal{S}(o).$$

By definition of product and the fact that by its definition $\mathcal{S}$ has a natural transformation to the identity it must be the case that

$$\forall o : O \cdot P \times o \sqsubseteq \mathcal{S}(o).$$

Now

$$\begin{aligned} \forall o : O \cdot P \times o \sqsubseteq \mathcal{S}(o) &\iff \mathcal{S} \stackrel{\bullet}{\rightarrow} \lceil \bigvee \mathcal{S}(\!|\, P_W \,|\!) \times \_ \rceil \\ &\iff \lceil \bigvee \mathcal{S}(\!|\, P_W \,|\!) \times \_ \rceil \stackrel{\bullet}{\leftarrow} \mathcal{S} \end{aligned} \tag{6.2}$$

since in a preorder category the natural transformation diagram commutes because of uniqueness of arrows. Finally

$$\begin{aligned} P \sqsubseteq \bigvee \mathcal{S}(\!|\, P_W \,|\!) &\iff \lceil P \times \_ \rceil \stackrel{\bullet}{\leftarrow} \lceil \bigvee \mathcal{S}(\!|\, P_W \,|\!) \times \_ \rceil \\ &\quad \text{[By definition of product and uniqueness of arrows in a preorder.]} \\ &\Longrightarrow \lceil P \times \_ \rceil \stackrel{\bullet}{\leftarrow} \mathcal{S} \\ &\quad \text{[By 6.2 above and transitivity of } \stackrel{\bullet}{\leftarrow}\text{.]} \\ &\iff P \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S} \end{aligned}$$

□

This theorem states that, in an information flow category in which there exists a wide preorder sub-category reflecting refinement, a sufficient condition for "secure" functional refinement can be expressed as being in an interval. The upper bound of

the interval in the preorder is the functional specification and the lower bound is the least upper bound of the image of the confidentiality statement. The theorem allows a designer to take a confidentiality statement, calculate the least upper bound and thus generate a proof obligation for secure refinement. If this condition is not met the system may still be secure since it is only a sufficient condition.

Example 39 If $\mathcal{T}$ and R_FILTER are as defined in example 38 with $W = \{Unclass\}$ then

$$\begin{aligned}
&\bigvee \mathcal{S}(\!| R_FILTER_W |\!) \\
&= (RU, \bigcup_{l \in Unclass^*} \{ tr : RU^* \mid CONSISTENT \wedge KNOWN \wedge RESTRICT \}) \\
&= (RU, \{ tr : RU^* \mid \exists l \in Unclass^* \cdot CONSISTENT \wedge KNOWN \wedge RESTRICT \}) \\
&= \left(RU, \left\{ tr : RU^* \;\middle|\; \begin{array}{l} tr \upharpoonright Unclass \in Unclass^* \\ \wedge \\ odd(tr \downarrow Rest) = tr \upharpoonright Unclass \\ \wedge \\ even(tr \downarrow Rest) \in Mess^* \end{array} \right\} \right)
\end{aligned}$$

From this point it is simple, but tedious, to show

$$tr \in \bigvee \mathcal{S}(\!| R\_FILTER_W |\!) \Longrightarrow tr \in traces(R\_FILTER \upharpoonright RU).$$

Establishing this means that by theorem 13

$$R\_FILTER \ \textbf{s-sat}^{\{\text{Unclass}\}} \ \mathcal{T}$$

This proves the claim made in example 38. △

The next theorem shows that with an additional assumption a necessary and sufficient condition can be used for refinement.

Corollary 11 *In $\mathcal{IF}_{\sqsubseteq}$, if P is a system, $\mathcal{S}$ is a confidentiality statement over observations P_W through a set of alphabets W, and*

$$\lceil \bigvee \mathcal{S}(\!| P_W |\!) \times \_ \rceil = (P_W \rhd \mathcal{S})$$

then

$$P \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \Longleftrightarrow P \ \textbf{s-sat}^{W} \ \mathcal{S}$$

Proof

By the same proof as theorem 13 except

$$\lceil P \times \_ \rceil \overset{\bullet}{\leftarrow} \lceil \bigvee \mathcal{S}(\!| P_W |\!) \times \_ \rceil \Longrightarrow \lceil P \times \_ \rceil \overset{\bullet}{\leftarrow} \mathcal{S}$$

can be strengthened to

$$\lceil P \times \_ \rceil \stackrel{\bullet}{\leftarrow} \lceil \bigvee \mathcal{S}(\!| P_W |\!) \times \_ \rceil \Longleftrightarrow \lceil P \times \_ \rceil \stackrel{\bullet}{\leftarrow} \mathcal{S}$$

since $\lceil \bigvee \mathcal{S}(\!| P_W |\!) \times \_ \rceil = (P_W \rhd \mathcal{S})$. □

This very useful result can be used in the practical development of systems. An example of when this holds comes from example 37.

Example 40 If $\mathcal{S}$ and *SECURE_FILTER* are as defined in example 37 and $W = \{Unclass\}$ then

$$\bigvee \mathcal{S}(\!| SECURE\_FILTER_W |\!) = (CU, CU^*)$$

therefore for any observation $(Unclass, \{l\})$ in $SECURE\_FILTER_W$

$$\begin{array}{l} \lceil \bigvee \mathcal{S}(\!| SECURE\_FILTER_W |\!) \times \_ \rceil \ (Unclass, \{l\}) \\ = \lceil (CU, CU^*) \times \_ \rceil \ (Unclass, \{l\}) \\ = (CU, CU^*) \times (Unclass, \{l\}) \\ = (CU, \{\, tr : CU^* \mid tr \upharpoonright Unclass = l \,\}) \\ = (CU, \{\, tr : CU^* \mid tr \upharpoonright Unclass = l \wedge tr \upharpoonright Conf \in Conf^* \,\} \end{array}$$

therefore

$$\lceil \bigvee \mathcal{S}(\!| SECURE\_FILTER_W |\!) \times \_ \rceil = (SECURE\_FILTER_W \rhd \mathcal{S})$$

Now $(CU, traces(SECURE\_FILTER) \upharpoonright CU) = (CU, CU^*)$ and

$$\bigvee \mathcal{S}(\!| SECURE\_FILTER_W |\!) = (CU, CU^*)$$

therefore by corollary 11 it must be the case that

$$(SECURE\_FILTER \upharpoonright CU) \ \textbf{s-sat}^{\{\text{Unclass}\}} \ \mathcal{S}.$$

This proves the claim made at the end of example 37. △

Some other simple but useful corollaries follow from theorems 12 and 13.

Corollary 12 *In $\mathcal{IF}_{\sqsubseteq}$, if $\mathcal{S}$, $\mathcal{T}$ are confidentiality statements over observations P_W and Q_W respectively through windows in W, and*

$$P \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \wedge Q \sqsubseteq \bigvee \mathcal{T}(\!| Q_W |\!)$$

then

$$\text{i)} \ P \times Q \ \textbf{s-sat}^{\text{W}} \ \mathcal{S} \wedge_{\text{IF}} \mathcal{T}$$

and

$$\text{ii)} \ P + Q \ \textbf{s-sat}^{\text{W}} \ \mathcal{S} \vee_{\text{IF}} \mathcal{T}$$

Proof

$$P \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \wedge Q \sqsubseteq \bigvee \mathcal{T}(\!| Q_W |\!) \implies P \times Q \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \times \bigvee \mathcal{T}(\!| Q_W |\!)$$

[by theorem 12]

$$\implies P \times Q \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S} \wedge_{\mathrm{IF}} \mathcal{T}$$

[by theorem 13]

This proves i).

$$P \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \wedge Q \sqsubseteq \bigvee \mathcal{T}(\!| Q_W |\!) \implies P + Q \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) + \bigvee \mathcal{T}(\!| Q_W |\!)$$

[by theorem 12]

$$\implies P + Q \ \textbf{s-sat}^{\mathrm{W}} \ \mathcal{S} \vee_{\mathrm{IF}} \mathcal{T}$$

[by theorem 13]

This proves ii). □

This corollary states that if the systems P and Q have the minimal amount of behaviour required for the confidentiality statements $\mathcal{S}$ and $\mathcal{T}$, then their product and coproduct will satisfy the conjunction and disjunction respectively of $\mathcal{S}$ and $\mathcal{T}$. This allows a designer to produce components which meet separate requirements, expressed as a minimal set of behaviours, and still be able to compose them together to meet all the requirements simultaneously.

Example 41 Extending examples 39 and 40, corollary 12 gives

$$(R\_FILTER \upharpoonright RU) \| (SECURE\_FILTER \upharpoonright CU) \ \textbf{s-sat}^{\{\mathrm{Unclass}\}} \ \mathcal{T} \wedge_{\mathrm{IF}} \mathcal{S}$$

and

$$(R\_FILTER \upharpoonright RU) [\!] (SECURE\_FILTER \upharpoonright CU) \ \textbf{s-sat}^{\{\mathrm{Unclass}\}} \ \mathcal{T} \vee_{\mathrm{IF}} \mathcal{S}$$

△

The next corollary extends corollary 12 to include functionality.

Corollary 13 *In $\mathcal{IF}_{\sqsubseteq}$, if $\mathcal{S}, \mathcal{T}$ are confidentiality statements, S and T are functional specifications such that*

$$S \sqsubseteq P \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \wedge T \sqsubseteq Q \sqsubseteq \bigvee \mathcal{T}(\!| Q_W |\!)$$

then

$$\text{i)} \ S \times T \sqsubseteq P \times Q \sqsubseteq \bigvee \mathcal{S}(\!| P_W |\!) \times \bigvee \mathcal{T}(\!| Q_W |\!)$$

and

$$\text{ii)}\ S + T \sqsubseteq P + Q \sqsubseteq \bigvee \mathcal{S}(\!|\ P_W\ |\!) + \bigvee \mathcal{T}(\!|\ Q_W\ |\!)$$

Proof

By theorem 12 and corollary 12. □

This theorem guarantees that if a designer has managed to find components which meet desired functional and security requirements then they can be combined to meet the combined functional and security requirements.

6.2.2 Conclusions

Specifications for general confidentiality properties can be given for systems separately from functional specifications. In general a confidentiality specification will conflict with a functional specification, however with care the conflict can be reconciled. Informally a functional specification provides an "upper" bound for a system and a confidentiality specification provides a "lower" bound. The act of "classical" refinement moves a system away from the "upper" bound towards the "lower" bound. A system will satisfy both the functional and confidentiality specification if refinement keeps the new system within this interval.

The results given show that if systems satisfy their individual functional and confidentiality specifications then their composition via categorical product and coproduct satisfy the product and coproduct of their specifications. In concrete instances, such as Safety-Iflow, this means that CSP parallel and choice composition preserve functionality and confidentiality. In the concrete categories for timed traces, refusals and Petri nets the product and coproduct operators take on similar interpretations. In a category designed to reason about inference in a relational database product and coproduct would correspond to relational operations.

Chapter 7

Information flow across Petri nets and time

In this chapter two concrete categories are constructed. These categories satisfy the postulates for an Information flow category stated in chapter 3.

7.1 How to build your own calculus!

In this section guidelines are given on how to build a semantic model for a calculus of information flow for the reader's favourite model of computation. Because of the categorical approach taken in this thesis the reader of this section must have some familiarity with category theory to be able to build their version of a model of information flow. The guidelines are illustrated by outlining the construction of a model of information flow for Petri nets. In subsection 7.1.1 a brief review of an algebraic treatment of Petri nets is given. This forms the theoretical basis for the construction of a suitable model for information flow, which illustrates how one might go about constructing other models of information flow, in subsection 7.1.2.

7.1.1 Petri Nets are Monoids

Petri nets are widely used to model concurrent systems, a description of Petri nets can be found in [Reisig 85]. The algebraic model for Petri nets used in this chapter is largely based on the elegant formulation given in [Mess. & Mont. 88].

Definition 58 (Petri Nets) *A place/transition net is a triple* (S, T, F)*, where*

- S *is a set of places*
- T *is a set of transitions*
- $F : (S \times T) + (T \times S) \to \mathbb{N}$ *is a bag called the causal dependency relation. (*$\mathbb{N}$ *denotes the natural numbers and* $+$ *denotes the disjoint union of sets.)*

$\diamond$

In this thesis the term Petri Net will mean the general case of a place/transition net. The key point of [Mess. & Mont. 88] is that a Petri net is simply an ordinary directed graph with two algebraic operations representing concurrent and sequential composition.

Definition 59 (Graphs) *A graph G is a set T of arcs, a set V of nodes and two functions ∂_0 and ∂_1 called source and target, respectively.*

$$\partial_0, \partial_1 : T \to V$$

A morphism h from G to G' is a pair of functions (f, g) such that

$$f : T \to T' \text{ and } g : V \to V'$$

such that

$$g \circ \partial_0 = \partial'_0 \circ f \quad \wedge \quad g \circ \partial'_1 \circ f$$

$\diamond$

This defines a category **Graph** with objects graphs and morphisms graph homomorphisms, with the obvious pointwise composition of graph homomorphisms. With the previous definition, the definition of a Petri net can be re-formulated.

Definition 60 *A Petri Net is a graph where the arcs are called transitions and the set of nodes is the free commutative monoid $S^{\oplus}$ over a set of places S:*

$$\partial_0, \partial_1 : T \to S^{\oplus}$$

A Petri net morphism is a graph morphism (f, g), where g (the function from nodes to nodes) is a monoid homomorphism. $\diamond$

Petri nets and their morphisms define a category **Petri**. It is easy to see that this new definition coincides with the standard definition if for each $t \in T$ the set

$$\{ s : S \mid F(s,t) \neq 0 \vee F(t,s) \neq 0 \}$$

is finite.

In [Mess. & Mont. 88] it is shown that the categorical product of two Petri nets is the *synchronous product* of the nets. This means that the synchronous product of two Petri nets is the result of a composition operator with synchronization. The places of the result are the union of the places of the factors, while the transitions in the synchronous product are pairs, that is synchronizations, of the given transitions.

The category **Petri** also has coproducts. The coproduct of two Petri nets is the result of a composition operation without synchronization: the two nets are just laid side by side without interaction. With nets that have initial state the meaning of coproduct is nondeterministic choice composition.

The synchronous product of two Petri nets is a little too strong for the purpose of determining inference from an observation. For example the product of the two Petri nets of figure 7.1 gives a Petri net which does not capture the case when a transition can occur from place b to place c. To capture this behaviour the notion of a pointed Petri net is required.

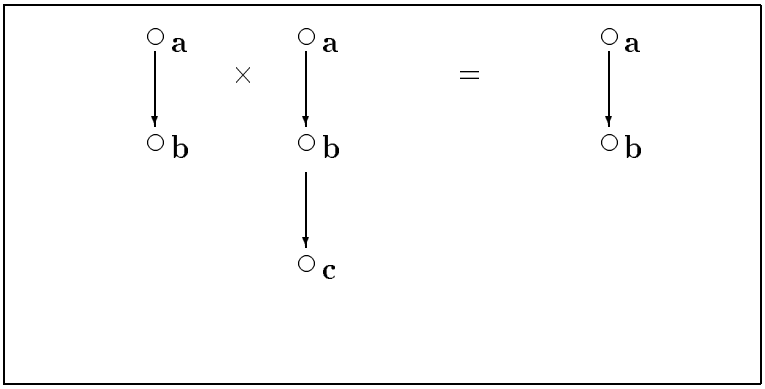


Figure 7.1: Synchronous product of Petri nets

Definition 61 (Pointed Petri Nets) *A pointed Petri net consists of a Petri net where the set of transitions is a pointed set* $(T, 0)$, $0 \in T$, *the commutative monoid* $S^{\oplus}$ *is viewed as a pointed set, and*

$$\partial_0, \partial_1 : (T, 0) \to S^{\oplus}$$

are pointed set maps. ◇

The 0 element is a special element added to a set T to make it pointed. This is a standard mathematical trick to represent partial functions by total functions. The 0 element can be thought of as undefined, so any elements of T which are not mapped by a partial function are mapped to 0. Maps between pointed sets are required to leave 0 fixed and directly correspond to partial functions between the original sets. A pointed Petri net morphism is a Petri net morphism (f, g), where f is a map of pointed sets, this defines a category $\mathbf{Petri}_0$.

As in **Petri**, the product of two pointed Petri nets as graphs has an obvious pointed net structure and yields the categorical product in $\mathbf{Petri}_0$. The difference is illustrated by the product of the two Petri nets in figure 7.2. Now the transition from place b to place c can occur. Coproducts also exist in $\mathbf{Petri}_0$.

7.1.2 Information flow for Petri nets

Using the given model of computation presented in subsection 7.1.1 a semantic model for a calculus of information flow is constructed in this section. To build a semantic model for information flow five steps are necessary. These steps are not independent but the fundamental step which influences the rest is

Step 1: Decide what composition operator, for "systems", categorical product represents and make the category rich enough to support all small products and coproducts.

The composition operator must have the properties of categorical product, but this

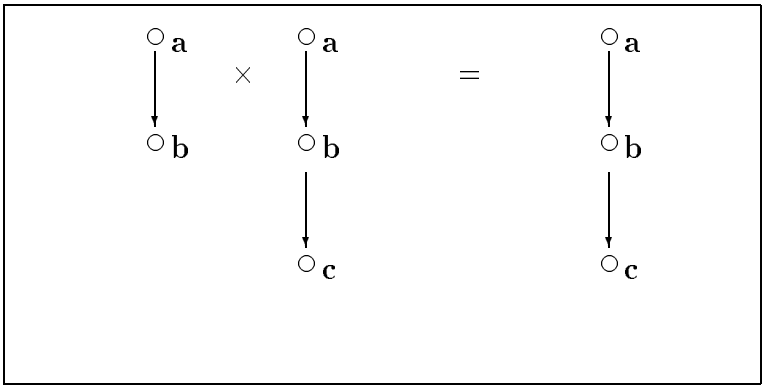


Figure 7.2: Product of pointed Petri nets

can be achieved by defining the objects and arrows of the category appropriately. For Petri nets the composition operator of interest is the parallel composition of two nets, which from subsection 7.1.1 can be represented as categorical product. Other composition operators might be inter-module communication as, for example, described in [Meadows 92]. The idea of a system is quite general in the case of inter-module communication a "system" is a module, for the example at hand it is a Petri net. With the definition of categorical product comes its dual, coproduct, for free. As indicated in subsection 7.1.1 the dual of product for Petri nets corresponds to non-deterministic composition of Petri nets.

The first step is an approximation to the information flow category and defines categorical product to correspond to the composition operator of interest, the next step is to find suitable observations.

Step 2: Define what the observations of systems are such that the categorical product of a system and an observation gives a reasonable notion of inference.

This step may well involve adding objects and arrows to the category in order to introduce suitable observations or indeed modifying the existing objects and arrows. The third step involves an initial object for the information flow category.

Step 3. Check the existence of an initial object or introduce it and make it satisfy the appropriate properties for an information flow category.

An initial object is required to give a clean mathematical treatment of 'false' in a calculus of information flow. It seems to be relatively easy to introduce, in a way that satisfies the appropriate postulate for an information flow category, without modifying the other objects and arrows.

The final step is to check the distributive property of postulate 2 stated in the chapter 3.

Step 4. Check postulate 2 for an information flow category and modify the category if necessary and possible.

If this final step is successful then one has constructed a suitable semantic model for a calculus of information flow and a semantic mapping, from a suitable abstract syntax, can be made in the way illustrated in chapter 4. If the distributive property does not hold then it might be possible to modify the category in order to allow it. If this is not possible then the semantic model will still support a conjunction and disjunction calculus with 'false' and 'true'. A calculus such as this is still very expressive, all that is really lost is a way of comparing specifications in the calculus.

To illustrate these steps a completed definition of an information flow category is defined and then how this definition was arrived at is discussed. First an auxiliary definition is given.

Definition 62 *If $A^{\oplus}$ and $B^{\oplus}$ are commutative monoids freely generated by A and B respectively and Σa_i denotes a element of the monoid $A^{\oplus}$ then*

$$(\_ \restriction \_) : (A^{\oplus} \times \mathbb{P}B) \rightarrow B^{\oplus}$$

is an infix operator which denotes a monoid homomorphism from $A^{\oplus}$ to $B^{\oplus}$ such that

$$\Sigma a_i \restriction B = \Sigma f(a_i)$$

where

$$\begin{array}{ll} f(a_i) = a_i & \text{if } a_i \in B \\ f(a_i) = 0_B & \text{else} \end{array}$$

The operator $\restriction$ is also used to denote restriction for a set of elements of a monoid, its definition is the obvious extension to a set. $\diamond$

Definition 63 (Petri_Iflow) *The objects of Petri-Iflow are all triples (with a single exception) of the form:*

$$(N, A, M)$$

where $N = (\partial_0, \partial_1 : (T, 0) \rightarrow S^{\oplus})$ is a pointed Petri net as defined in definition 61; and M is a set of elements of the commutative monoid $A^{\oplus}$. The single exception to this is the empty set which is added to give a suitable initial object for the category. A morphism exists from (N, A, M) to (N', A', M') if and only if

$A' \subseteq A$;

there exists a Petri net morphism from N to N';

and $M \restriction A' \subseteq M'$.

There is also a morphism from the empty set to any other object in Petri_Iflow where the morphisms are given by the unique functions from the empty set, in the underlying category of sets. $\diamond$

Note that to define inference over standard Petri nets A will be a subset of S, but this constraint is not enforced generally in order to make life easier when showing that Petri_Iflow is an information flow category. To understand how this definition

came about the four steps are followed.

Step 1:

For Petri nets most of the hard work has already been done in [Mess. & Mont. 88]. Petri nets are the systems of interest and categorical product in Petri_0 is the parallel composition of two Petri nets with synchronization over common transitions. The coproduct is the non-deterministic choice composition of two Petri nets. The product and coproduct of two Petri nets generalizes to arbitrary sets of nets, hence postulate 1 holds (the initial net has no places and no transitions). The first bit of work that needs to be done concerns observations.

Step 2:

To give a reasonable notion of an observation, such that the product of a system and an observation gives a realistic inference, means that the objects of the category have to be more than just Petri nets. An observation of a system consists of a Petri net and a marking, that is a collection of tokens on the places of the Petri net. This pair represents the partial view of the system with its marking corresponding to a partial view of the system's marking. The marking can be represented by an element of the monoid which makes up the places of the Petri net. An inference differs from an observation in that more than one marking is possible, each marking represents a possible configuration which is consistent with the system and the observation.

For this reason the objects of the category are modified to objects of the form (N, A, M), where the set of monoid elements M denotes a set of possible markings of the pointed Petri net N. Unfortunately there is now no initial object, therefore the empty set with its morphisms is added to the category to act as an artificial initial object. The product and coproduct of objects in the augmented category is the product and coproduct of the pointed petri nets and the product of the set of monoid elements under restriction.

Lemma 24 *If*

$$N \triangleq (\partial_0, \partial_1 : (T, 0) \to S^{\oplus})$$

and

$$N' \triangleq (\partial'_0, \partial'_1 : (T', 0) \to S'^{\oplus})$$

are pointed Petri nets and both (N, A, M) and (N', A', M') are objects in Petri_Iflow then

$$(N \times N', A \cup A', \{\, m : (A \cup A')^{\oplus} \mid m \restriction A \in M \wedge m \restriction A' \in M' \,\})$$

is their product and

$$(N + N', A \cap A', \{\, m : (A \cap A')^{\oplus} \mid m \restriction A \in M \vee m \restriction A' \in M' \,\})$$

is their coproduct.

Proof

The product and coproduct of the objects of Petri_Iflow are simply the products and coproducts of the components of the objects. The product and coproduct of the

pointed Petri nets is product and coproduct in $\mathbf{Petri}_0$. The product of the second components, M and M', follows from the definition of $\uparrow$. An easy way to see this is to note that traces form a monoid over an alphabet S, the empty sequence is the unit and concatenation is the monoidal operator. The definition of $\uparrow$ for monoids coincides with the definition of $\uparrow$ for traces, hence the product of two sets of traces coincides with the product of two sets of monoid elements which is how product has been defined in the lemma. Coproduct of two sets of monoid elements follows in the same way. □

This lemma shows that products and coproducts still exist; they exist for any set of objects in Petri_Iflow since in [Mess. & Mont. 88] it is stated that any set of Petri nets has a product and coproduct and the other components of the triple have products and coproducts by analogy with the trace model. Thus the first step in designing the information flow category is still valid. Note that the marking is **not** being used in the normal way. The markings on concurrent Petri nets would be present on each net and fire concurrently. The markings here represent an observation of a system not a stage in an actual computation, therefore a Petri net system will have no markings itself just these associated observations; thus the same observation of two concurrent nets becomes merged into one rather than two firing together.

Step 3:

The initial object was artificially introduced into the category and it was defined so that step 3 would follow easily. The product of the initial object with any other object is the initial object because there is no monoid homomorphism from an object of Petri_Iflow to the empty set, but there is always a unique arrow, the added set functions, from the empty set to any other object.

Step 4

To achieve step 4 subcategories of Petri_Iflow need to be defined.

Definition 64 *$Petri\_Iflow_A$ is a subcategory of Petri_Iflow where the set of generators, A, for the markings, M, is fixed. The initial object is added to make the subcategories conform to steps* 1 *and* 3. ◇

Note that the generators for the monoid in the pointed petri net will not be fixed because the product of two Petri nets will give the disjoint union of their respective monoids of nodes.

Lemma 25 *If (N, A, M) is an object of $Petri\_Iflow_A$ then the functors*

$$[(N, A, M) \times \_](N', A', M') = (N, A, M) \times (N', A, M')$$

preserve small colimits over $Petri\_Iflow_A$.

Proof

A sketch of the proof is that the functors preserve colimits because the components of the objects (N, A, M) do. Pointed petri nets preserve colimits under product because

finite product and coproduct, of free commutative monoids, coincide. The markings preserve colimits under product by the same argument given in lemma 16 in chapter 3. That is, if the monoid is represented by traces, with concatenation as the monoid operator and the empty trace as the unit, then generators A preserve colimits under product because set intersection distributes over union. □

This lemma indicates that like CST the full calculus is defined only when the set of generators, A, is known and fixed. When the set of generators has not been fixed, the calculus only has conjunction, disjunction, top (the value representing true) and bottom (the value representing false).

It has been shown using these four steps that a suitable semantic space can be constructed to define an information flow calculus for Petri nets. The semantic function for an atomic confidentiality statement still needs to be defined to provide the link between a language and the semantic space for an information flow calculus. A confidentiality statement for Petri nets would take a representation of an observation to a set of possible markings over some general net. The set of possible markings could be denoted by some predicate. The explicit construction of an abstract syntax and the semantic function are left as a challenge to the reader with chapter 4 as a guide.

7.2 A Category of Timed Traces

One model of computation which is suitable for reasoning about information flow is the timed trace model of Timed CSP, TCSP [Reed 88]. TCSP is a process algebra with a hierarchy of semantic models, the timed trace model is one of the simplest. The associated theory of TCSP allows systems to be specified with time critical requirements, as well as untimed requirements, and systems can be shown to correctly implement such specifications, [Schneider 89]. The reason why the timed trace model has been chosen, is to base a theory of information flow within a theory designed to support the construction of functional systems with limited timing channels.

In the next part of this section an example is given of how information can flow across an apparently secure system when timing information is taken into consideration; a partial order over timed traces is then introduced. The partial order is shown to have the properties required to be an information flow category and certain subsets of it are shown to be adequate semantic domains for a calculus of information flow; the results proved in this thesis are therefore valid in this new category.

7.2.1 Covert communication over timed traces

In the trace model of CSP no timing information is present, however information can flow using events which can represent either an input or an output to a system. Given a system it is necessary to identify external operations of a system where users or processes can interact with the system. The interactions with a system can be passive, in the sense of accepting an output, or active, when making an input. An example of a "secure" system is given.

Example 42

$$P \triangleq (a \rightarrow b \rightarrow STOP) [] (b \rightarrow a \rightarrow STOP)$$

In the trace model of CSP

$$\tau(P) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle b\rangle, \langle b, a\rangle\}.$$

With respect to the users interacting through the alphabets $\{a\}$ and $\{b\}$ not only does $\{a\}$ non-interfere with $\{b\}$ but a process at $\{b\}$ cannot deduce anything about a process at $\{a\}$ and of course vice-versa.

$\triangle$

The next example illustrates how taking time into account can show that an apparently secure system can leak information. First an auxiliary definition is made.

Definition 65 *If T is a set of timed traces then $\tau(T)$ is the set of traces with the time of occurrence of each event removed.* $\diamond$

Example 43

$$Q \triangleq (a \rightarrow b \rightarrow STOP) [] WAIT\ 1; (b \rightarrow a \rightarrow STOP)$$

In the trace model of CSP there is no record of when an event occurs therefore the process $WAIT\ 1$ is indistinguishable from $SKIP$, hence

$$\tau(Q) = \tau(P)$$

and if an observer cannot see the time an event occurred, Q is as secure as P.

In the timed trace model of CSP

$$\text{Hstrip}(\mathcal{T}_T(Q)) = \left\{ \begin{array}{l} \langle\rangle, \langle(t_1, a)\rangle, \langle(t_1, a), (t_1 + t_2, b)\rangle, \\ \langle(1 + t_3, b)\rangle, \langle(1 + t_3, b), (1 + t_3 + t_4, a)\rangle \mid \\ t_1 \geq 0 \wedge t_2 \geq 1 + \delta \wedge t_3 \geq 0 \wedge t_4 \geq \delta \end{array} \right\}$$

(where $\mathcal{T}_T(Q)$ is the semantic function from the process algebra of TCSP to its semantics model.) In TCSP a process interacting through $\{a\}$ interferes with a process interacting through $\{b\}$ because time allows previously indistinguishable traces to be distinguished. For example consider the timed trace $\langle(0, a), (0.5, b)\rangle$ of Q; purging the a events gives $\langle(0.5, b)\rangle$ which is not a behaviour of Q. In other words $\{a\}$ can interfere with $\{b\}$ because the delay of 1 second means that $\{b\}$ will only see $\langle(0.5, b)\rangle$ when an a event has occurred. $\triangle$

Note that the timing information used to distinguish traces is just additional information. The events which a process refused to participate in after a certain trace could equally be used to distinguish traces. Probabilities associated with a trace could also distinguish traces and be used leak information. Time in this respect is just another kind of observation of a system.

7.2.2 A partial order for Timed traces

The following constraints for timed traces are taken directly from [Reed 88].

Definition 66

$$TM_A \triangleq \mathbb{P}(TA)_{\leq}$$

such that if $S \in TM_A$ *then*

1. $s \in S \wedge s \cong w \Longrightarrow w \in S$
2. $\forall t \in [0, \infty) \cdot \exists n(t) \in \mathbb{N} \cdot \forall s \in S \cdot (end(s) \leq t \Longrightarrow \#s \leq n(t))$

$\diamond$

Condition 1 states that the order of events which occur at the same time is irrelevant. Condition 2 captures the reality that infinitely many events cannot occur in a finite period of time. Note that the conditions for a set of timed-traces to be non-empty and prefix closed have been omitted as well as conditions on hatted traces.

Definition 67 *If* V *is a set of events then*

$$TV_M \triangleq \{ s : TV_{\leq} \mid \forall t \in [0, \infty) \cdot \exists n(t) \in \mathbb{N} \cdot (end(s) \leq t \Longrightarrow \#s \leq n(t)) \}$$

$\diamond$

This definition means that $TV_M \in TM_A$ since condition 1 of definition 66 is vacuously satisfied by TV_M because by definition of the set TV_M, in chapter 2, it is the set of all timed traces which preserve causality.

Definition 68 (Tsafe-Iflow) *The objects of Iflow are all pairs of the form:*

$$(A, T)$$

where A *is any subset of an alphabet* Σ *and* $T \in TM_A$, $A = \{\}$ *only when* $T = \{\langle\rangle\}$.

$$(A, T) \preceq (B, S)$$

if and only if

$$B \subseteq A \ \wedge \ T \upharpoonright B \subseteq S$$

$\diamond$

Lemma 26 *Tsafe-Iflow is a partial order.*

Proof

Follows from the properties of $\upharpoonright$ and $\subseteq$. $\square$

Considering a partial order as a preorder category means that least upper bound of a set of objects is a colimit and greatest lower bound corresponds to limit. The next lemma shows that Tsafe-Iflow is a complete lattice.

Lemma 27 *If X is a collection of objects in Tsafe-Iflow then the greatest lower bound is*

$$\bigwedge X \triangleq (V, \bigcap_{(A,T):X} \{\, t : TV_M \mid t \restriction A \in T \,\})$$

and the least upper bound is

$$\bigvee X \triangleq (Z, \bigcup_{(A,T):X} T \restriction Z)))$$

where

$$V \triangleq \bigcup_{(A,T):X} A \quad \text{and} \quad Z \triangleq \bigcap_{(A,T):X} A.$$

Proof

Follows by a similar induction proof to that used in lemma 13. □

Example 44 The greatest lower bound of

$$W \triangleq (\{a\}, \{\, \langle\rangle, \langle(t,a)\rangle \mid t \geq 0 \,\})$$

and

$$V \triangleq (\{a\}, \{\, \langle\rangle, \langle(t,a)\rangle \mid t \geq 1+\delta \,\})$$

is

$$(\{a\}, \{\, \langle\rangle, \langle(t,a)\rangle \mid t \geq 1+\delta \,\})$$

and their least upper bound is

$$(\{a\}, \{\, \langle\rangle, \langle(t,a)\rangle \mid t \geq 0 \,\}$$

△

In the treatment of CSP used in [Hoare 85], each process P is associated with a unique set of events, the alphabet of the process. The association of an alphabet with a process is mirrored by the elements of the partial order Tsafe-Iflow which consist of a pair of an alphabet and a set of timed traces. In the theory of timed CSP the need for process alphabets is removed by the introduction of an alphabeticised parallel operator. It is convenient to retain the alphabet in Tsafe-Iflow but the connection between the alphabeticised parallel operator and elements of Tsafe-Iflow is maintained by taking the greatest lower bound of projections through appropriate alphabets, as the following proposition shows.

Proposition 12 *If P and Q are systems and $A = \sigma(P)$, $B = \sigma(Q)$ and $X \subseteq A$, $Y \subseteq B$ then*

$$((A, \mathcal{T}_T(P)) \restriction X) \times ((B, \mathcal{T}_T(Q)) \restriction Y) = (X \cup Y, \mathrm{Hstrip}(\mathcal{T}_T(P_X \|_Y Q)))$$

Proof

$$((A, \mathcal{T}_T(P)) \restriction X) \times ((B, \mathcal{T}_T(Q)) \restriction Y) = (X, \mathcal{T}_T(P) \restriction X) \times (Y, \mathcal{T}_T(Q) \restriction Y)$$

Since $X \subseteq A$ and $Y \subseteq B$. The alphabet component of the proposition now follows directly from lemma 27. Considering the second component only

$$\{ t : T(X \cup Y)_M \mid t \restriction X \in \mathcal{T}_T(P) \restriction X \wedge t \restriction Y \in \mathcal{T}_T(Q) \restriction Y \}$$

$$= \{ t : T(X \cup Y)_M \mid \exists s_p : \mathcal{T}_T(P) \restriction X; s_q : \mathcal{T}_T(Q) \restriction Y \cdot t \restriction X = s_p \wedge t \restriction Y = s_q \}$$

$$= \{ t : T\Sigma_M \mid \exists s_p : \mathcal{T}_T(P) \restriction X; s_q : \mathcal{T}_T(Q) \restriction Y \cdot t \restriction (X \cup Y) = t \wedge t \restriction X = s_p \wedge t \restriction Y = s_q \}$$

[Since $t \restriction (X \cup Y) = t$ keeps only traces from $T(X \cup Y)_M$]

$$= \left\{ \begin{array}{l} t : T\Sigma_M | \exists s_p : \mathcal{T}_T(P) \restriction X; s_q : \mathcal{T}_T(Q) \restriction Y \cdot \\ t \restriction (X \cup Y) = t \wedge \mathit{hstrip}(t) \restriction X = \mathit{hstrip}(s_p) \wedge \mathit{hstrip}(t) \restriction Y = \mathit{hstrip}(s_q) \end{array} \right\}$$

[Since $\mathit{hstrip}(t) = t$ when t is unhatted.]

$$= \mathrm{Hstrip}(\mathcal{T}_T(P_X \|_Y Q))$$

[Since hatted information is discarded.]

□

The previous lemma shows that if one is ignorant of when an event becomes available then greatest lower bound and parallel composition are the same. In the next few lemmas and propositions it is shown that Tsafe-Iflow is an Iflow category.

Lemma 28 *Every set of objects in Tsafe-Iflow has a product and coproduct.*

Proof

Product and coproduct correspond to the greatest lower and least upper bound respectively in a partial order. Any collection of objects is guaranteed a product and a coproduct by Tsafe-Iflow being a lattice. □

Lemma 28 shows that postulate 1 of chapter 3 is satisfied by Tsafe-Iflow.

Lemma 29 *The initial and terminal objects of Tsafe-Iflow are $(\Sigma, \{\})$ and $(\{\}, \{\langle\rangle\})$ respectively.*

Proof

Direct from definition 68. □

The sub-lattices of Tsafe-Iflow which form cartesian closed subcategories of Tsafe-Iflow are now examined.

Definition 69

$$\textit{Tsafe-Iflow}_A \triangleq \{ (A,T) \mid (A,T) \in \text{Tsafe-Iflow} \}$$

$\diamond$

Tsafe-Iflow$_A$ is the set of elements of Tsafe-Iflow with a fixed alphabet, A. The following lemma shows that the monotonic function $[(A,S) \times \_]$ preserves colimits in the subcategory Tsafe-Iflow$_A$.

Lemma 30 *For each $A, B \subseteq \Sigma$, if $(B,S) \in$ Tsafe-Iflow$_A$ then $[(B,S) \times \_]$ preserves least upper bounds*

Proof

Since the alphabet of each object in Tsafe-Iflow is fixed the trace component of the least upper bound is simply a union of timed traces. If X is any subset of Tsafe-Iflow$_A$ then

$$\begin{aligned}
&[(B,S) \times _](A, \bigcup_{(A,T):X} T) \\
&= (B \cup A, \{ t : T(B \cup A)_M \mid t \in S \wedge t \in \bigcup_{(A,T):X} T \}) \\
&= (B \cup A, \bigcup_{(A,T):X} \{ t : T(B \cup A)_M \mid t \in S \wedge t \in T \}) \\
&= \bigcup_{(A,T):X} [(B,S) \times _](A,T)
\end{aligned}$$

$\square$

The concrete information flow categories can now be characterized.

Proposition 13 *For each $A \subseteq \Sigma$, Tsafe-Iflow$_A$ is an Iflow category.*

Proof By lemma 28, postulate 1 stated in chapter 3 holds. By definition of bottom in a partial order, the greatest lower bound (or product) of any other element with bottom is bottom. By lemma 30 postulate 2 holds. $\square$

This proposition shows that by fixing the alphabet for a set of sets of traces all the operators for a Heyting algebra can be given a meaning. This concrete model can be used to give a semantic space for a semantic space for a particular calculus of information flow over timed traces.

7.3 Discussion

A category of Petri nets has been shown to be another model of computation which satisfies the postulates of chapter 3. This is important for two reasons: it provides evidence that the approach taken in this thesis is general; and the Petri net model

presented in [Mess. & Mont. 88] has been used to give a semantic model to CCS. The model of computation is not a preorder which justifies the use of categorical product and coproduct rather than the simpler concepts of greatest lower bound and least upper bound. The recipe like presentation of the Petri net category will hopefully encourage the reader to try and construct an information flow category using their favourite model of computation!

The timed trace model describes interactions with systems in **real** time. It has been shown to have enough extra information in order to show information flow problems in systems which for untimed interactions would be considered secure.

The examples used to illustrate information flow also illustrate that a timing channel is nothing special, only extra information about an interaction which can be observed. Other models such as refusals (what events a system refuses to participate in) also have extra information about interactions which can be exploited to leak.

Using a slightly simpler model of timed traces, that is ignoring when an event *becomes* available, a partial order has been defined which is shown to be a complete lattice. Certain sub-lattices have been shown to be suitable models for a Heyting algebra itself a model for intuitionistic logic. These Heyting models provide a semantics for equivocation propositions described in the thesis. They provide a flexible way of specifying arbitrary information flow properties of systems involving time.

It is generally much more important that a system does what it was procured for rather than for it to be absolutely secure. For this reason theories of security should be based on theories of computation created to support the development of functional systems. CSP and TCSP are examples of such theories, they have been used in applications from specifying the data link level of ethernet to providing the semantics for the programming language of the transputer. A body of knowledge is accumulating on using TCSP, [Davies et al 90] & [Jackson 90], and proving correctness of systems in TCSP, [Schneider 89].

Using a simple model of TCSP a partial order over sets of timed traces has been constructed and shown to be another model of computation which satisfies the postulates of chapter 3. Hatted events in the partial order Tsafe-Iflow have been ignored, this means that hiding cannot be defined in Tsafe-Iflow. Another problem is that the information about when an event becomes available can also be used to leak information. To include hatted events and address these problems a slightly more complicated category is needed. This does not seem worth the effort for timed traces. A timed failures category would remove the need for hats and also address the problem of leaking information via refusing to perform certain events at certain times. The timed trace model can be embedded in the timed trace-stability model of TCSP. The extra information given by stabilities, the maximum time which a process is tied up performing internal events, is of no use in reasoning about information flow since each timed trace has a unique stability value.

Chapter 8

A Secure CCIS

The most difficult problems in the procurement of computer systems arise in evaluating how well a system solves some real world problem such as preventing the theft of information, detecting enemy missiles or ensuring that the shelves of a supermarket are stocked correctly. The problem is that the system must perform in an environment that is complex and imperfectly understood. The requirement for confidentiality is to prevent sensitive information from reaching inappropriate parts of an organisation or indeed reaching parties outside an organisation.

The purpose of this chapter is to present a case study to support the assertion that the theory developed in the previous chapters is useful. The case study of a Command and Control Information System, CCIS, is based on realistic although fictitious requirements. A CCIS is a system which will support the control and command of resources (external to the system) using available information. Resources are controlled by commands issued through the system. A command in such a system is simply an order or directive based on known and externally received information. The control aspect denotes the collation of known and external information in order to reach a decision. A CCIS can support control by the collation and collection of information, and by distributing control. The nature of a CCIS is that of large amounts of data moving round a distributed system. There are no sophisticated algorithms necessary for a CCIS, but the sheer volume of information brings its own problems, especially from the viewpoint of security. An everyday example of a CCIS is a system for a Hypermarket which controls the stocks of goods sold and the means of transporting goods from a warehouse or supplier.

Issues of confidentiality in an information system are more complicated than one might think at first sight. For example individuals responsible for carrying out an order can be at a lower clearance than the originator of the order; this means that a classified order entered into a system will cause data to flow to individuals with a lower clearance. Whenever data flows from a user with a higher clearance than the recipient, there is a risk that confidentiality might be violated.

The confidentiality requirement for a system has to be balanced against the functionality of the system. For example a system which prevents the release of information which is necessary to the conduct of a battle might be secure, but is useless to a commander. To support the expression and analysis of confidentiality require-

ments a notation is needed to: clarify the issues of what parts of an organisation use the system directly; what information is allowed to reach those parts; and what impact do separate confidentiality requirements have on each other and the functional requirements.

The standard approach to describing system confidentiality is to label objects and subjects in a system and restrict the access of subjects to objects. By viewing a system as a collection of objects (files) and subjects (users or programs accessing files) it is easy to see how to implement access controls. There are three major problems with this approach:

- it obscures the real issues;
- it does not address the use of commercial off the shelf software with adequate granularity;
- and it does not address covert channels of communication.

At the requirements level it is harmful to describe the confidentiality of a system in terms of how the system should store information and how it should be accessed. The reason for this is that because confidentiality is a non-functional property, specifying the mechanisms used to achieve confidentiality obscures: the degree of confidentiality; and what should be kept confidential from whom.

An example of inadequate granularity is that of a commercial database management system. Such a system could be used with files which contain tables. A table will usually have columns at different levels of sensitivity; but by classifying at the file level the relationships between different columns is not addressed and either sensitive information is not protected, or the functionality of the database is impaired.

The problem of covert channels arises because the very mechanisms used to protect data can be used to communicate that data to unauthorized individuals. The covert channel problem however has been given undue prominence in recent years leading to overly restrictive expressions of confidentiality requirements. For example submitting a highly classified order to a system which causes data, necessary to perform the order, to flow to unclassified users is secure but would not be allowed. It will frequently be the case that covert channels will present no practical risk and therefore need not be taken into account. At present it seems to be the case that secure systems are designed to perform some real world task with respect to a requirement which demands an absolute ban on information flow between certain users; this is a case of "the tail wagging the dog".

None of the current approaches for describing confidentiality are adequate for specifying and analysing confidentiality requirements. A sensible and well balanced set of measures is needed when specifying the confidentiality requirements of a system. This means that data must be allowed to flow from one user to another but the meaning of that data must be taken into account. The prevention of exploitation of covert channels can be addressed separately by, for example, using a controlled user interface and development system.

8.1 A natural language requirement

Suppose an organization requires automated support for the movement of its assets, in this case boots, to and from various locations. Such a system should not keep the movement of boots a secret from the parts of the organization which have to organise and move the boots around. Such a restriction would conflict with the system's purpose, to support the command and control of boots. There are security problems if, for example, the reason for the movement of boots is a secret, but that reason must be entered into the system to allow other users to do their job.

Assume there are eight participating groups using the CCIS, namely: *HQ*; *Stock-cell*; *Transport*; *Movement-cell*; *Warehouse*; *Exercise*; *Operator* and *Security officer*. Each group has a different function in the operation of Boots.

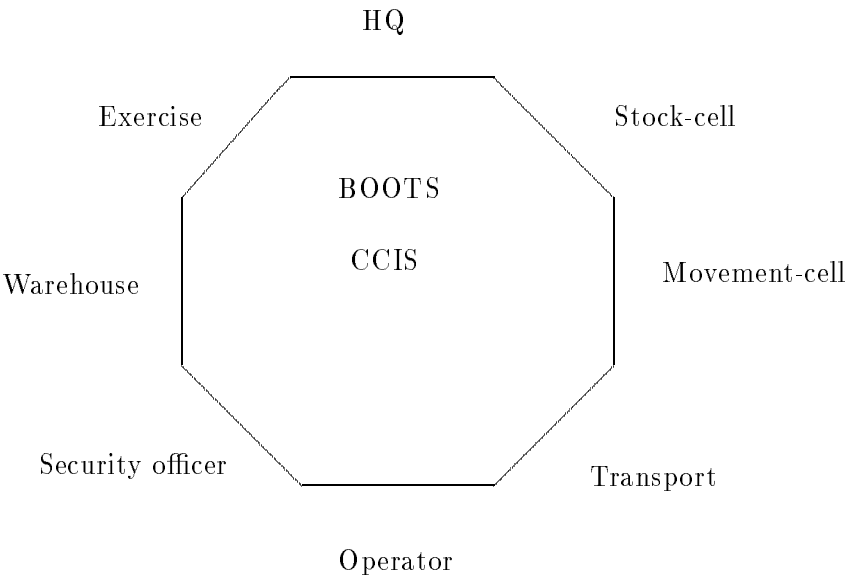


Figure 8.1: Black box representation of Boots.

HQ *HQ* is the source of orders in the system. An aide to a general informs the system that N boots are needed at a destination for a purpose, P. For simplicity the purpose, P, is assumed to be a code name for some pre-planned operation. Any group receiving the code name can look up the purpose in a locally held document.

The type of boot is not the concern of HQ, that is the responsibility of the *Stock-cell*. HQ only concerns itself with the strategic movement of boots, deciding where these boots should come from is not their concern.

Stock-cell The *Stock-cell* decides what kind of boot needs to be transported to a destination and from where the boots should be taken. The type of boot is determined by the purpose code name and the source of the boots is determined

by the stock records. For example a purpose code name such as "operation snowman", when found in the code word document, means the type of boot should be suitable for arctic conditions.

Some orders are more sensitive than others, due to the associated purpose. Sensitive messages are restricted to a smaller trusted group. The same order can be sensitive at one time and unclassified at another due to changing circumstances in the world. Only the stock control cell is allowed to know the purpose of an order. How the boots are transported from A to B is not the concern of the stock control cell.

Movement-cell The *Movement-cell* decides how many lorries are needed to move N boots from A to B. No other information is required.

Warehouse Managers at various warehouses receive orders to prepare N boots of a certain type for collection. The managers also inform the stock control cell of the arrival of new boots.

Transport The *Transport* depot needs to know how many lorries are needed to transport boots between their source and destination. Information must flow from *HQ* to *Transport* for *Transport* to do its job, but *Transport* must remain ignorant of the purpose of the order submitted by *HQ*.

Operator The *Operator* archives the system audit trail. The audit trail is a record of messages passed between different users of the system. The archived audit trail allows misuse of the system to be detected. In a real system an operator would also backup all information held on a system but the Boots system is assumed simpler to illustrate only the principle.

Security officer The *Security officer* can inspect the source and destination of messages in the audit trail, in order to monitor unusual traffic, but should not be able to deduce the content of messages passed between users. The audit trail must not be changed except by adding further interactions to it. The security officer can also downgrade prepared messages on request. For example an aide might mistakenly submit a classified order with a purpose which is not really classified. The security officer can inspect that order with its associated purpose and downgrade the order if requested.

Exercise *Exercise* is a window which allows an exercise directing officer to simulate the behaviour of *HQ* and *Warehouse*. Exercise information can cause actual lorries to move around but must not corrupt real data such as the distribution of boots.

For simplicity there is only one kind of sensitive information which is held on the CCIS, this is the purpose of an order. Users of the system which can access this information are trusted not to compromise security. For example a user who is cleared to access the purpose of an order is trusted not to copy it to users who are not cleared to know it. The CCIS is not trusted to prevent unauthorized disclosure

of information. The absence of any illegal disclosures of information is a property of a system which must be established. In general the absence of illegal disclosures is achieved by showing that the interfaces of a system comply with the security requirements and that there are no undesirable side effects when an interface is used.

Each confidentiality requirement is stated in terms of what an agent in a user group is allowed to know about the actions of an agent in another user group. For brevity statements such as "an agent in user group A is not allowed to know about the actions of an agent in user group B", will be paraphrased by "user group A is not allowed to know about the actions of user group B".

Recall that in the description of *Stock-cell* there was a requirement for a smaller trusted group of the stock control cell to handle sensitive orders. To describe this the user group *Stock-cell* is partitioned into "high" and "low" to denote where sensitive orders can be processed. Each user subgroup is identified with separate sub-sections of stock control. Similarly the *Exercise* user group is partitioned into *HQ (exercise)* and *Warehouse (exercise)*. The user subgroup *HQ (exercise)* is not partitioned by high and low exercise orders. To participate in exercises, without corrupting real data, the user group *Stock-cell* is partitioned into *Stock-cell (real)* and *Stock-cell (exercise)*. For the same reason *Movement-cell* is partitioned into *Movement-cell (real)* and *Movement-cell (exercise)*.

Statements of the security requirement are listed below and include requirements for confidentiality and for integrity. The meaning of each statement is given afterwards.

1. *Stock-cell (low)* **must not know the purpose of a high order** received, from *HQ*, at *Stock-cell (high)*.

2. The user groups: *Movement-cell*; *Transport*; *Exercise*; *Operator* and *Warehouse* **must not know the purpose of an order** from *HQ*.

3. The audit trail **must not be altered** by anyone.

4. The user group *Operator* **must not know** the audit trail.

5. The *Security officer* **must not know the content of messages** passed directly between users.

6. The user groups: *Stock-cell (real)*; *Movement-cell (real)* and *Warehouse* **must not be altered by** the user groups: *Exercise*; *Stock-cell (exercise)*; and *Movement-cell (exercise)*.

The first security requirement is that sensitive orders, that is those with a purpose with a classification "high", are restricted to a smaller trusted group, that is those with a clearance "high". The second security requirement is that only the stock control cell, *HQ* (and sometimes the *Security officer*) is allowed to know the purpose of an order. The meaning of the third requirement is that the audit trail can only be modified by adding records of new interactions to it. The fourth security requirement is that an *Operator* cannot read the audit trail. The fifth requirement captures the

desire that the security officer only knows enough to do his job. The final requirement is that exercise data must not corrupt real data, although *Transport* can receive orders to really move lorries as part of an exercise.

The words in each security requirement will be formalized in section 8.2 and appendix A.1. Note that the security requirements contain integrity constraints as well as confidentiality constraints.

Each user group and its associated functionality defines the potential observations which can be made by that group.

HQ At the user group *HQ* an order is prepared consisting of three parts: the number of boots needed; their destination and a purpose code name. If the order has mistakenly been over classified then it is sent to *Security officer* with a request for downgrading, if not then it is sent to *Stock-cell*.

Stock-cell On receiving an order from *HQ* two kinds of message are sent, one to *Movement-cell* and one to *Warehouse*. To determine the message content for the message to *Movement-cell* an inquiry is made on the disposition of boots.

A stock control officer can monitor the arrival of new boots at warehouses and decide to move them. This might be done for various reasons such as preventing a warehouse from becoming full or so that not all boots of a certain type can be destroyed in a warehouse fire.

The requirement for the exercise part of the Stock-cell is the same as the requirement for its real world operation except that orders from *HQ* are simulated by *Exercise* (HQ). Similarly two types of message are sent, one to *Movement-cell (exercise)* and one to *Exercise (Warehouse)*. Exercise data is interrogated to determine what boots are available. Messages to *Movement-cell (exercise)* consist of the same parts as messages to *Movement-cell*. Messages to *Exercise (Warehouse)* consist of two parts, the number of boots and the kind of boots required.

Movement-cell On receiving a message from *Stock-cell*, or *Stock-cell (exercise)*, an inquiry is made of the system concerning the disposition of lorries, then a message is sent to *Transport* consisting of three parts: the pick up point; the destination; and which lorries are to be used.

Warehouse On receiving a message from *Stock-cell* the warehouse manager prepares the necessary boots for collection. When new boots are received the system is informed of this by a message consisting of two parts, the number of new boots and their type.

Transport On receiving a message from *Movement-cell*, *Transport* takes the appropriate action. When a lorry reports that a delivery has been completed the system is informed of this by a message consisting of the identity of the lorry and its position.

Operator At the user group *Operator* a request for a backup of the audit trail can be made at any time. When the system is ready an operator is informed and when an operator loads a tape (or whatever medium is used to backup the audit trail) the system is informed. An operator is informed when the backup is complete.

Security officer *Security officer* receives an audit record of security relevant events when an archive of the system is made. The set of security relevant events are records consisting of which user groups have communicated with each other. When a request from *HQ* is received to downgrade an order the security officer can downgrade it and send it on to *Stock-cell*.

Exercise The user group *Exercise* can prepare an order like *HQ*, or simulate a Warehouse manager. Orders from *HQ* consist of three parts: the number of boots needed; their destination and a purpose code name. When simulating a Warehouse manager *Exercise* can receive messages from *Stock-cell (exercise)* about boots needed for collection. A message can be sent to that part of the system supporting the exercise consisting of two parts, a number (denoting new boots received) and their type.

In figure 8.2 the communication channels between the user groups are shown. For example the orders from *HQ* to *Stock-cell* means that there is a connection between *HQ* and *Stock-cell*. The *Operator* is not directly linked to any user group since the function of the *Operator* is to only archive audit trails and an audit trail is recorded by the system autonomously.

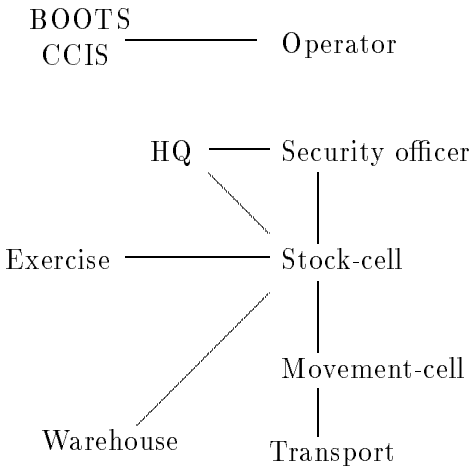


Figure 8.2: Communication channels between user groups.

8.2 Formalizing a confidentiality requirement

In this section a single natural language requirement for a CCIS is formalized. The final part of section 8.2 presents the conjunction of the remaining confidentiality requirements which have been formalized in the appendix. The conjoined confidentiality specification is then compared with the individual confidentiality specifications and any conflicts are examined.

8.2.1 A confidentiality specification

At this stage there is no information on what operations will make up the potential observations of the CCIS. It is convenient to give a name now to the set of operations which a user group can do, these will be called windows. Each user group must login with a password which authenticates that user group. This means that a user group can be identified by its window, therefore the name given to each window will be the name of the associated user group.

The alphabet of the CCIS, that is the complete interface to the environment, is the union of all the windows.

$$Boots \triangleq \begin{array}{l} HQ \cup Stock\text{-}cell \cup Security\ officer \cup \\ Movement\text{-}cell \cup Transport \cup Exercise \cup \\ Operator \cup Warehouse \end{array}$$

The set of windows over which the confidentiality specification will be defined is denoted

$$Boot\ Windows \triangleq \{Low, The\ rest, Operator, Security\ officer, Real, Boots\ (exercise)\}$$

for convenience. The windows *The rest*, *Boots (exercise)* and *Real* in the set *Boot Windows* are the union of other windows in *Boots* and their definitions given in the appendix.

To write a confidentiality statement describing the confidentiality requirement given in the previous section some details about the window *Stock-cell* must be given.

In **Stock-cell** there are only two classifications associated with an order received from *HQ*.

$$Class \triangleq \{high, low\}$$

An order consists of three parts: a request to move some boots; the purpose of the request; and the classification of the order.

$$Order \triangleq \{ (R, P, C) \mid R : REQUEST \wedge P : PURPOSE \wedge C : Class \}$$

Note that the three parts do not have to be sent together. They could be separate messages sent to *Stock-cell* which together constitute an order. It seems simpler however to explicitly associate these components together. The information flow techniques illustrated in this paper work independently of these implementation issues.

Orders to *Stock-cell* received from the CCIS which are classified *high* are denoted *High* and handled by a sub-section of the *Stock-cell*.

$$High \triangleq \{ (R, P, high) \mid R : REQUEST;\ P : PURPOSE \}$$

Similarly orders to *Stock-cell* which are classified *low* are denoted *Low* and handled by a sub-section of the *Stock-cell* distinct from *High*.

$$Low \triangleq \{ (R, P, low) \mid R : REQUEST;\ P : PURPOSE \}$$

The window *Low* along with *High* partition the set of orders from *HQ*.

$$Order = High \cup Low$$

Stock control operations are denoted by *Stock op*. More details about the composition of *Stock op* are deferred until later in the development process.

To formalize the security requirement it is necessary to give a formal meaning to "not knowing the purpose of a high order". Another way of saying this is that for any potential observation through the window *Low*, which is identified with the less trusted stock-cell group, it must not be possible to infer which *PURPOSE* value is associated with the classification *high*.

$$\mathcal{SR}_1 \triangleq \textbf{From } l : Low^* \textbf{ MAY\_INFER } tr \upharpoonright Low = l \textbf{ Over } tr : (High \cup Low)^*$$

At the window *Low* the confidentiality specification forbids the inference of what *High* events might have occurred. This confidentiality specification is similar to the one given in example 37. The confidentiality specification limits an observer, at the windows other than *Low*, to only being able to infer what has already been observed and nothing more. This is the strongest possible constraint which is weakened when other confidentiality specifications are conjoined with this specification.

This sub-section has shown how confidentiality statements can be produced from natural language requirements. The next sub-section shows how these can be combined together and analysed.

8.3 An analysis of the confidentiality requirements

In this section the confidentiality specification for the stock control cell is conjoined with the confidentiality specifications for the other user groups, given in the appendix, to give the overall confidentiality requirement. The third security requirement is ignored because it is captured as a specification of allowable sequences of operations and can be shown to be satisfied by the techniques described in [Hoare 85]. The confidentiality specification $\mathcal{SR}_1$, defined in sub-section 8.2.1, is conjoined with the confidentiality specifications $\mathcal{SR}_2$, $\mathcal{SR}_4$, $\mathcal{SR}_5$ and $\mathcal{SR}_6$ defined in appendix A. The conjunction is not a simple expansion of all the confidentiality statements but is an evaluation involving conjunction of predicates and alphabet expansions. The

evaluated expression covers more than a page and the reader should not attempt to understand it. The evaluated expression is the sum of the simpler parts and is an intermediate step which will be simplified. It is worth noting that the conjunction is still simpler than many formalisms of security.

$$\mathcal{SR}_1 \wedge_{\mathrm{IF}} \mathcal{SR}_2 \wedge_{\mathrm{IF}} \mathcal{SR}_4 \wedge_{\mathrm{IF}} \mathcal{SR}_5 \wedge_{\mathrm{IF}} \mathcal{SR}_6$$

evaluates to

$$\begin{array}{l}
\textbf{From } l : Low^* \\
\textbf{MAY_INFER} \\
\left(\begin{array}{l} tr \restriction Low = l \\ \wedge \\ tr \restriction High \in High^* \end{array} \right) \\
\textbf{Over } tr : Order^*
\end{array}$$

$$\begin{array}{l}
\textbf{From } l : The\ rest^* \\
\textbf{MAY_INFER} \\
\left(\begin{array}{l} tr \restriction The\ rest = l \\ \wedge \\ Pse^*(tr \restriction Order) \in PURPOSE^* \\ \wedge \\ \#tr \restriction Order \leq \#l \\ \wedge \\ tr \restriction (The\ rest \cup Audits) \in ded_archive(l) \end{array} \right) \\
\textbf{Over } tr : (The\ rest \cup Order \cup Audits)^*
\end{array}$$

$$\begin{array}{l}
\textbf{From } l : Operator^* \\
\textbf{MAY_INFER} \\
\left(\begin{array}{l} tr \restriction Operator = l \\ \wedge \\ tr \restriction (Operator \cup Audits) \in infer_archive(l) \end{array} \right) \\
\textbf{Over } tr : (Operator \cup Audits)^*
\end{array}$$

$$\begin{array}{l}
\textbf{From } l : Security\ officer^* \\
\textbf{MAY_INFER} \\
\left(\begin{array}{l} tr \restriction Security\ officer = l \\ \wedge \\ tr \restriction (Security\ officer \cup Auditable) \in ded_trail(l) \end{array} \right) \\
\textbf{Over } tr : (Security\ officer \cup Auditable)^*
\end{array}$$

$$
\begin{array}{l}
\textbf{From } l : Real^* \\
\textbf{MAY\_INFER} \\
\left(\begin{array}{l}
tr \upharpoonright Real = l \\
\wedge \\
tr \upharpoonright Boots(exercise) \in Boots(exercise)^* \\
\wedge \\
Pse^*(tr \upharpoonright Order) \in PURPOSE^* \wedge \\
tr \upharpoonright (Real \cup Auditable) \in ded\_trail(l)
\end{array} \right) \\
\textbf{Over } tr : (Real \cup Order \cup Boots\ (exercise) \cup Auditable)^*
\end{array}
$$

This complex confidentiality specification describes the total requirement for confidentiality for the CCIS. The complexity is tamed by only specifying it in simple understandable parts. The total requirement is a weakening of the individual requirements, it allows more to be inferred than any of the individual requirements. The problem now is to make sure that it is strong enough to give a secure system. Only two parts of the total specification are analysed in this chapter because the rest follow the pattern of the first requirement and it is only the second one which is significantly weaker.

To analyse the total specification of the CCIS the inference over each window in the specification is considered separately. This means that the confidentiality statements over the other windows are ignored. For example the specification $\mathcal{SR}_1$ only allows inference over the window *Low*; to compare the total specification with $\mathcal{SR}_1$ the inference over *The rest*, *Operator*, *Security officer* and *Real* is ignored.

The inference over *Low* in the total specification is a predicate over $Order \cup Boots(exercise)$ whereas the inference over *Low* in $\mathcal{SR}_1$ is a predicate over *Order* only. This means that to compare the total specification with $\mathcal{SR}_1$ the total specification must be restricted to operations from *Order*. Recall that

$$Order = Low \cup High$$

therefore restricting the predicates in the overall confidentiality specification to *Order* gives

$$
\begin{array}{l}
\textbf{From } l : Low^* \textbf{ MAY\_INFER} \\
\left(\begin{array}{l}
tr \upharpoonright (Low \cap Order) = l \upharpoonright Order \\
\wedge \\
tr \upharpoonright (High \cap Order) \in (High \cap Order)^*
\end{array} \right) \\
\textbf{Over} \\
tr : (Order \cap Order)^*
\end{array}
$$

Using simple laws for alphabet restriction stated in [Hoare 85] this can be simplified to

$$
\textbf{From } l : Low^* \textbf{ MAY\_INFER } \left(\begin{array}{l}
tr \upharpoonright Low = l \\
\wedge \\
tr \upharpoonright High \in High^*
\end{array} \right) \textbf{ Over } tr : Order^*
$$

This part of the overall confidentiality specification is exactly the same as the corresponding part in the first confidentiality statement $\mathcal{SR}_1$. This means that the other

confidentiality specifications with which $\mathcal{SR}_1$ was conjoined have not weakened the confidentiality requirement over the user group which uses *Low* operations. A weakening in a confidentiality requirement is sometimes necessary for a system to do the job it was procured for, but a weakening is also a compromise in confidentiality which should be examined closely.

Analysing $\mathcal{SR}_6$

The remaining confidentiality requirements when analysed do not weaken the confidentiality requirement, except for $\mathcal{SR}_6$. The sixth confidentiality requirement is concerned with the real world operation of the CCIS not being altered by the exercise operations. Extracting the sixth part of the total confidentiality specification and restricting the result to $Real \cup Boots\ (exercise)$ gives

$$
\begin{array}{l}
\textbf{From } l : Real^* \\
\textbf{MAY\_INFER} \\
\left(\begin{array}{l}
tr \upharpoonright Real = l \\
\wedge \\
tr \upharpoonright (Real \cup Boots\ (exercise)) \in ded\_trail(l) \\
\wedge \\
tr \upharpoonright Boots\ (exercise) \in Boots\ (exercise)^*
\end{array} \right) \\
\textbf{Over } tr : (Real \cup Boots\ (exercise))^*
\end{array}
$$

which differs from the original, stated in the appendix, by the extra constraint of

$$tr \upharpoonright (Real \cup Boots\ (exercise)) \in ded\_trail(l)$$

This is because $Auditable \subset Real \cup Boots\ (exercise)$ and therefore *Real* can infer more information than the original confidentiality statement allowed.

A weakening

The weakening of $\mathcal{SR}_6$ is due to the fifth security requirement. This requirement is that the security officer can monitor the occurrence of message passing between different users but is not allowed to know the content of these messages. The security officer is part of the real world operation and therefore part of the window *Real*. The security officer monitors all interactions including exercise interactions. The security officer is affected by exercise operations contrary to the sixth security requirement. This is legitimate if the security officer is to do his job properly according to the functional description. This is a case where a weakening in the confidentiality requirements is justified in order for a system to perform properly. Catching conflicts in requirements like this is an important point. If a system was built which tried to satisfy all the security requirements then there would be a major problem when it was realised the security requirements were too strong. This would lead to delays and possibly a system with accidental security loopholes.

8.4 A static representation of the requirements

In this section the techniques of chapter 6 are used to describe the confidentiality statements of the previous sections in terms of static typed machines. The typed machine representations are used in the next section and appendix A to show that a collection of CSP components satisfy the confidentiality specifications.

The security requirement $\mathcal{SR}_1$

The main concern of the confidentiality specification $\mathcal{SR}_1$ was how much could be inferred from an observation of operations from Low^*. The first step in building a static description is therefore to describe what observations can be made.

$$SR1_{Low} \triangleq e : Low \rightarrow SR1_{Low}$$

The interface object $SR1_{Low}$ allows any operation classified as *low* to be performed. To express the desired inference in a static way an inference object for $SR1_{Low}$ is needed.

$$K\_SR1_{Low} \triangleq o : High \cup Low \rightarrow K\_SR1_{Low}$$

The inference object K_SR1_{Low} of $SR1_{Low}$ legislates that any order is possible. No operation classified *high* interferes with an operation classified *low*. K_SR1_{Low} can easily be shown to be an inference object of $SR1_{Low}$ by applying the CSP restriction operator, with the set Low, to the process K_SR1_{Low}.

To check that the interface object $SR1_{Low}$ and its inference object K_SR1_{Low} give at least as much confidentiality as the specification $\mathcal{SR}_1$ over observations drawn from Low^* they must be compared as functions. The confidentiality specification $\mathcal{SR}_1$ is

$$\textbf{From } l : Low^* \textbf{ MAY\_INFER } tr \upharpoonright Low = l \textbf{ Over } tr : (High \cup Low)^*$$

with the assumption that no inference is allowed through the other windows. As a function

$$\mathcal{SR}_1(l) = \{\, tr : (Order)^* \mid tr \upharpoonright Low = l \,\}$$

for every $l : Low^*$.

The traces of $SR1_{Low}$ can be calculated

$$traces(SR1_{Low}) = Low^*$$

therefore the domain of $\mathcal{SR}_1$ is the same as the interface $SR1_{Low}$. For an observation l of the interface the associated inference is

$$\{\, tr : Order^* \mid tr \upharpoonright Low = l \wedge tr \in traces(K\_SR1_{Low}) \,\}$$

but

$$traces(K\_SR1_{Low}) = Order^*$$

therefore the associated inference is

$$\{\, tr : Order^* \mid tr \upharpoonright Low = l \,\}$$

which agrees with $\mathcal{SR}_1$.

The other interfaces

The other interfaces and their inference objects are very simple to specify. For example

$$SR1_{The\_Rest} \triangleq e : The\_Rest \rightarrow SR1_{The\_Rest}$$

since like $SR1_{Low}$ there are no restrictions on what observations can be made at this interface. No more information is allowed to be inferred at the interface *The_Rest*, therefore the inference object is exactly the same as the interface object.

$$K\_SR1_{The\_Rest} \triangleq SR1_{The\_Rest}$$

The traces of these objects can be easily calculated and shown to give a function which is equivalent to the confidentiality specification over observations drawn from The_Rest^*. The static representation of confidentiality over the remaining interfaces: *Operator*; *Security officer*; and *Real* are expressed in a similar manner to $SR1_{The\_Rest}$ and its inference object $K\_SR1_{The\_Rest}$. The remaining static representations of confidentiality are described in appendix A.

8.5 Compliance of a design

To show compliance of a design for a secure information system against a confidentiality specification it is necessary to identify which components of the design must satisfy which parts of the confidentiality specification. In this case study this is done by taking the intersection of the alphabet of a component with each window in *Boot Windows* to see where the component "touches" the observable environment of the CCIS. For example consider the following component.

For a fixed number $n : \mathbb{N}$ the buffer of orders from *HQ* to *Stock-cell* is:

$$ORDER\_QUEUE \triangleq ORDER\_QUEUE_{\langle\rangle}$$

$$ORDER\_QUEUE_{\langle\rangle} \triangleq (stk, o) : \{stk\} \times Order \rightarrow ORDER\_QUEUE_{\langle o\rangle}$$

$$ORDER\_QUEUE_{s^\frown\langle o\rangle} \triangleq \begin{array}{l} \left(\begin{array}{l} (stk, o') : \{stk\} \times Order \rightarrow \\ ORDER\_QUEUE_{\langle o'\rangle^\frown s^\frown\langle o\rangle} \end{array} \right) \\ [] \\ (o \rightarrow ORDER\_QUEUE_s) \end{array}$$

such that $\#s < n - 1$

$$ORDER\_QUEUE_{s^\frown\langle o\rangle} \triangleq o \rightarrow ORDER\_QUEUE_s$$

such that $\#s = n - 1$

This process takes an order from *HQ* and puts it into a buffer ready to be processed by *Stock-cell*. When *Stock-cell* is ready the order o is taken from the queue. This

is a high level description of a queue between *HQ* and *Stock-cell*, it would be implemented using something like TCP/IP. To avoid introducing covert channels the implementation must be opaque, hiding the use TCP/IP. In other words to a user of Boots, *ORDER_QUEUE* looks like the implementation.

The intersection of the alphabet of *ORDER_QUEUE* with *Low* is *Low*. The intersection of the alphabet of *ORDER_QUEUE* with the other windows in *Boot Windows* is the empty set, therefore the only observable behaviour of *ORDER_QUEUE* is over the traces drawn from Low^*. This means that the only confidentiality specification *ORDER_QUEUE* must be shown to satisfy is $\mathcal{SR}_1$ because it only allows inference over *Low*. No inference is allowed by $\mathcal{SR}_1$ at the other windows but this is alright because *ORDER_QUEUE* does not interact with any other behaviour at any of the other windows in *Boot Windows*.

To show that *ORDER_QUEUE* satisfies the first security requirement it must be shown that

$$ORDER\_QUEUE \upharpoonright Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_1$$

The operations of *HQ* are hidden because none of the security requirements legislate over these operations. Using corollary 9 and the work done in section 8.4 it is only necessary to show that there is an arrow from K_SR1_{Low} to $ORDER\_QUEUE \upharpoonright Order$, in the category $\mathbf{TMach}_{Safe}$, in order to show that *ORDER_QUEUE* is compliant. Calculating $ORDER\_QUEUE \upharpoonright Order$ gives

$$ORDER\_QUEUE \upharpoonright Order = o : Order \rightarrow ORDER\_QUEUE \upharpoonright Order$$

this describes the interface of *ORDER_QUEUE* as *Stock-cell* sees it. As far as a stock control officer is concerned it can receive an order from *HQ* at any time. Comparing $ORDER\_QUEUE \upharpoonright Order$ with K_SR1_{Low} of section 8.4 shows that they are isomorphic since

$$Order = High \cup Low$$

This means that

$$ORDER\_QUEUE \upharpoonright Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_1$$

The other components of the CCIS Boots are defined and shown to satisfy the security requirements in appendix A.

8.6 Assembling the components

With the definition of all the components, in appendix A and the previous section, they can be composed together in parallel to form the total system, *Boots_CCIS*.

$$Boots\_CCIS \triangleq \left(\begin{array}{l} ORDER\_QUEUE \| INVOICE\_QUEUE \| \\ MOVE\_QUEUE \| BOOT\_DATABASE \| \\ LORRY\_DATABASE \| LORRY\_QUEUE \| \\ RE\text{-}GRADE\_QUEUE \| GRADED\_QUEUE \| \\ ORDER\_QUEUE\_EX \| INVOICE\_QUEUE\_EX \| \\ MOVE\_QUEUE\_EX \| BOOT\_DATABASE\_EX \| \\ AUDIT\_PROC \| ARCHIVE \end{array} \right)$$

By corollary 2 the parallel composition of the restricted components of the Boots CCIS satisfies the conjunction of the individual confidentiality specifications. For example

$$ORDER\_QUEUE \upharpoonright Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_1$$

and

$$INVOICE\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

means that

$$(ORDER\_QUEUE \upharpoonright Order \| INVOICE\_QUEUE) \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_1 \wedge_{\text{IF}} \mathcal{SR}_2$$

This can be repeated for every process to show that satisfies

$$Boots\_CCIS \upharpoonright Boots \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_1 \wedge_{\text{IF}} \mathcal{SR}_2 \wedge_{\text{IF}} \mathcal{SR}_4 \wedge_{\text{IF}} \mathcal{SR}_5 \wedge_{\text{IF}} \mathcal{SR}_6$$

The restrictions to individual components distribute over parallel composition because, except for *AUDIT_PROC* and *ARCHIVE*, the processes have no events in common. This reflects the fact that message passing in the CCIS is asynchronous. The alphabet restriction distributes over the components *AUDIT_PROC* and *ARCHIVE* because for the events in common they do not disallow any behaviour. These components are essentially recording components and will allow any behaviour in the rest of the CCIS to occur, this preserves the amount of inference established for the individual components when they are composed in parallel.

With theorem 14, which shows that *AUDIT_PROC* satisfies the integrity property for audit trails, *Boots_CCIS* is shown to be secure with respect to its security policy. The degree of parallelism in the Boots CCIS is not fixed by the above description. Using the laws of CSP the process *Boots_CCIS* can be transformed into a purely sequential machine or any number of communicating systems. The laws of CSP guarantee semantic equivalence, with respect to the trace model, so the transformed Boots CCIS will satisfy the same security policy as the original.

8.7 Discussion

This case study indicates that using confidentiality statements to capture confidentiality requirements is useful in the development of real systems. The approach of defining a set of legal operations (or events) which each user can access can be realistically implemented by enforcing a standard interface to a system. An example of a standard interface which is being adopted for open systems is the graphical user interface OSF/Motif™. By constraining such an interface, using a small amount of trusted software, the set of observable events can be fixed and security properties formalized in the way described in this case study. This use of open system standards promises the possibility of building *secure* open systems.

Chapter 9

Discussion

9.1 Conclusion

In this thesis a calculus for expressing confidentiality requirements has been presented. Primitive confidentiality statements can be combined using operators from the calculus, which for the abstract calculus CS is an intuitionistic logic. The semantics for the calculus is in terms of an abstract category.

The abstract categorical definitions are related to a concrete model, Safety-Iflow, consisting of sets of traces which gave a link to CSP. It was then shown how systems could satisfy confidentiality specifications. With the definition of satisfaction laws were proved which allowed the parallel composition of two systems to satisfy the conjunction of their confidentiality specifications. Similarly it has been shown that the choice composition of two systems satisfied the disjunction of their confidentiality specifications. With the calculus laws such as:

$$\frac{P \;\; \textbf{s-sat}^{\mathrm{W}} \;\; \mathcal{T} \qquad \mathcal{S} \;\; \Longrightarrow_{\mathrm{IF}} \;\; \mathcal{T}}{P \;\; \textbf{s-sat}^{\mathrm{W}} \;\; \mathcal{S}}$$

can be derived.

In the last section of chapter 5 the use of a Galois correspondence was investigated. A Galois correspondence is a simple form of the more general situation of an adjunction. The Galois correspondence was defined over the partial order of confidentiality statements. It was shown that conjunction is a monotonic function from the partial order to itself and is half of a Galois correspondence. The other half of the Galois correspondence gives a way breaking an atomic confidentiality statement into the conjunction of two simpler confidentiality statements. The Galois correspondence seems to offer a rich area for further work.

In chapter 6 a category of typed machines in which the category $\mathcal{IF}$ could be embedded was constructed. The category of typed machines is based on the category

of graphs and uses the free graph functor to lift properties of a generating graph to the free graph. This gave a way of lifting a static confidentiality property to a dynamic confidentiality specification. More importantly it was shown that if a system satisfied the static confidentiality property then it satisfied the dynamic confidentiality specification, a confidentiality statement. These results give a very useful way of specifying and verifying confidentiality which when specialized to a concrete category could be used by a designer and verifier of secure systems.

In section two of chapter 6 a preorder subcategory of $\mathcal{IF}$ was assumed. This reflects refinement in the concrete categories. Using the preorder structure it was shown how a confidentiality statement could be collapsed into a single object which could then be directly compared with systems and their functional specifications. A number of laws for refinement were proved. In chapter 7 two concrete categories which were models for $\mathcal{IF}$ were constructed. The categories were also semantic domains for Petri Nets and TCSP which allowed the results of chapters 5 and 6 to be applied to them.

In chapter 8 a sizeable case study of the security requirement of a CCIS was undertaken. The security requirement of the CCIS is realistic and does not require absolute bans on information flow between different windows. There has been some criticism that information flow places such stringent requirements on a system that it cannot be built or is operationally useless. The case study illustrated the fact that the information flow approach is not necessarily a "denial of service attack". The proof that the system satisfied its security policy depended on the functional correctness of the system. For example if a system labelled a message incorrectly then confidentiality would be compromised. The requirement for correctness came down to correct labelling and in general would involve correctness of types. This indicates that machines which enforce typing, such as [Terry 89b, Foster 89], would give more assurance that they satisfied their security policy.

9.2 Comparisons

Access control

One of the first attempts to capture the requirement for a secure system formally was by Bell and LaPadula at the MITRE corporation, [Bell73a, Bell73b, Bell74]. They constructed a state machine model which was an abstraction of the Multics system they wished to show secure. The state machine had what they termed Subjects applying operations that might require access to objects. It is by controlling the access of Subjects to Objects that Bell and LaPadula proposed to achieve security.

The Subjects of the Bell and LaPadula model correspond to untrusted processes. The Objects of the model correspond to files, devices and so on. A state of the machine was considered to be secure if and only if it satisfied two properties. The properties prevented reading of data a process was not cleared to read and writing

data to an object with a classification less than that of the source. The "Basic Security Theorem" of Bell and LaPadula stated that a system is secure if and only if its initial state is secure and every state transition is secure.

Since Bell and LaPadula proposed their model it has been adopted widely and criticised as well. The problems with the particular approach of the Bell-LaPadula model have been well documented in for example [McLean 87]. The success of the Bell-LaPadula model to constructing systems is due to its inductive nature, that is its constraints on states and state transitions, and its resemblance to real machines; these two properties of the model make it easy for an implementor to build a system and give a convincing argument that the system is secure.

The success of the Bell-LaPadula model also gave rise to more sophisticated security models, such as [Terry 89a], which addressed the problems associated with the Bell-LaPadula model. There are two problems with all of these state machine based approaches. The first problem is that it is not obvious that the model does capture what is claimed of it. The second problem is that the security properties which a model claims to have may not be those required by a customer. The first problem is not exclusively one of the security model approach, the information flow approach taken in this thesis is also difficult to grasp and appreciate. The advantage of the approach taken in this thesis is that once understood security policies can be put in terms understandable to a human and be similar in form to the original security requirement. The second problem with security models is also avoided by the approach taken in this thesis. An information flow property appropriate to a security requirement can be specified more easily than a security model can be altered and shown to capture that requirement.

Information flow

The research presented in this thesis owes its genesis to the work of Jacob presented in [Jacob 88a]. In this paper the trace model of CSP is used to define a theory of confidentiality for systems. The basic premise is that given complete knowledge of a system's behaviours an observer can deduce the possible behaviours of the system which are consistent with that observation. For any system, S, the inferences of S, written infer S, is the function of two parameters defined by:

$$\begin{aligned} &\text{infer } SBl \triangleq \{\, t : \tau S \mid t \upharpoonright B = l \,\} \\ &\text{if } B \subseteq \alpha S \wedge l \in S \upharpoonright B \end{aligned}$$

The application of infer in "infer SBl" can be read as "the inferences about S that B can make, having observed l". The notion of inference from an observation is the basic idea behind the categorical product of a system and observation in chapter 5.

The concept of a confidentiality statement is an abstraction of Jacob's security specification. A security specification over an alphabet A is a partial function

$$f : \mathbb{P}A \to A^* \to \mathbb{P}A^*$$

which for each $B, C \in dom f$ and $l \in dom(fB)$, satisfies the properties:

1. $\mathrm{dom}(fB)$ is a prefix closed subset of B^*
2. $\forall t : (fBl) \cdot t \restriction B = l$
3. $\forall t : (fBl) \cdot t \restriction C \in dom(fC)$

The first condition constrains observations through a window to be coherent, like the traces of a CSP process. The second condition ensures that the behaviours which can be deduced from an observation l are all consistent with it, with respect to the trace model of CSP. To explain the last condition it is useful to explain some of its components informally. The term "fC" is like the function "infer SC" in that for some observation, fC specifies the behaviours which may be deduced. The last condition therefore states that any behaviour s deduced through a window B from l which can also be observed through C is in $\mathrm{dom}(fC)$. The reason for this condition is that the specification f should constrain the information which can be deduced from s.

The definition of a confidentiality statement only constrains a function over observations through a window to be consistent with that observation. This is achieved by insisting on a natural transformation to the identity functor. The reason why the other two conditions are not used is because we are not interested in defining what constitutes a sensible security specification, we are only interested in how they can be combined within the calculus CS. This means that the results of chapter 3 are valid for functions which are illegal security specifications.

Security specifications can be ordered like confidentiality statements. A security specification f is better than a security specification g if and only if three conditions are satisfied. The first condition ensures that f has at least as many windows as g. The second condition constrains f to have a subset of the observations of g through a window B. The final condition states that the inference with f is more unsure than the inference on g. These three conditions are captured in this thesis by requiring a natural transformation from a confidentiality statement g to a confidentiality statement f when restricted to the domain of f.

In [Jacob 88a] a system, S, is said to satisfy a security specification, f, if the infer function applied to S, viewed as a security specification, is no worse than f. In other words a system can be said to satisfy a security specification if the inference is less than or equal to the specification. This definition of satisfaction is similar to that given in chapter 5.

The basic concepts of inference and satisfaction which are defined in this thesis are abstractions of inference and satisfaction defined in [Jacob 88a]. Category theory has provided an elegant framework for formulating these concepts and has led to relatively simple proofs. Once the basic concepts had been formulated categorically the idea of the calculus CS was suggested to the author by the categorical literature. Similarly the other results of this thesis arose from the categorical tools available. The research presented in this thesis would have been very difficult to carry out within the trace model of CSP and of course would not have been as general.

In [Foley 88] a theory of information flow is also developed within the theory of CSP. The basis of the approach taken in Foley's thesis is to relate information

flow directly with Shannon's information theory. To establish the connection with information theory Foley associated probabilities with traces; this gave a means of measuring the bandwidth of covert channels in a system. The approach of introducing Shannon's information theory into models of information flow has also been pursued in [Gray 90] and [Gray 91]. This approach is interesting but unlikely to be useful for anything but showing the security of a small computing device such as a one way regulator. The reason for this is that the sheer complexity of computer systems today and the requirements which are placed upon them, makes it unlikely that the threat to confidentiality will be great enough to justify the demonstration of confidentiality involving probabilities. The results of this thesis would however be applicable in an appropriate probabilistic category. Work on probabilistic CSP is proceeding at the programming research group at Oxford and its semantic domain might provide a suitable probabilistic category.

Another approach to defining confidentiality with traces is that taken by Sutherland, [Sutherland 86]. The set of possible worlds, W, of a system is its set of traces in the trace model of CSP. The definition of information flow due to Sutherland is stated next in terms as close as possible to the original, but within the trace model of CSP.

Definition 70 (Deducibility) *If W is the set of possible worlds for a system P and $\upharpoonright A$ and $\upharpoonright B$ are information functions (where $A, B \in \mathbb{P}(\alpha P)$) then* **information flows from $\upharpoonright A$ to $\upharpoonright B$** *if and only if there exists a possible world w and some element z such that for some $v : W$, $v \upharpoonright B = z$ but*

$$\forall w' : W \mid w' \upharpoonright A = w \upharpoonright A \cdot w' \upharpoonright B \neq z$$

$\diamond$

Example 45

$$\begin{array}{lcl} P & \triangleq & OUT!x \rightarrow OUT!y \rightarrow STOP \\ LOSSI & \triangleq & OUT?x \rightarrow IN!x \rightarrow LOSSI \mid OUT?x \rightarrow LOSSI \\ Q & \triangleq & IN?x \rightarrow Q \\ W & = & P \| LOSSI \| Q \end{array}$$

The process $LOSSI$ has the behaviour of a buffer which can lose input. This is also the behaviour of a one-way regulator such as an optical isolator. There is obviously information flowing from the process P to the process Q via $LOSSI$ when they are put in parallel. According to Sutherland, since information flow is symmetric, there is also information flowing from Q to P, this is far from obvious. To clarify the situation we need to look at what information is flowing from Sutherland's point of view.

$$W = \left\{ \begin{array}{l} \langle\rangle, \langle OUT.x\rangle, \langle OUT.x, IN.x\rangle, \langle OUT.x, OUT.x\rangle, \\ \langle OUT.x, IN.x, OUT.y\rangle, \\ \langle OUT.x, IN.x, OUT.y, IN.y\rangle \end{array} \right\}$$

$$W \upharpoonright \{IN.v\} = \{\langle\rangle, \langle IN.x\rangle\langle IN.x, IN.y\rangle\}$$

and

$$W \upharpoonright \{OUT.v\} = \{\langle\rangle, \langle OUT.x\rangle\langle OUT.x, OUT.y\rangle\}$$

If the possible world $w = \langle OUT.x, IN.x\rangle$ and $z = \langle\rangle \in W \upharpoonright \{OUT.v\}$ then information flows from $\upharpoonright \{IN.v\}$ to $\upharpoonright \{OUT.v\}$ because

$$w \upharpoonright \{IN.v\} = \langle IN.x\rangle = \langle OUT.x, IN.x, OUT.y\rangle \upharpoonright \{IN.v\}$$

but

$$\langle OUT.x, IN.x\rangle \upharpoonright \{OUT.v\} \neq \langle\rangle$$

and

$$\langle OUT.x, IN.x, OUT.y\rangle \upharpoonright \{OUT.v\} \neq \langle\rangle.$$

That is if P has not transmitted anything then an observer at P knows that Q has not received anything. It is also true that if P has not sent $\langle IN.x\rangle$ then an observer at P knows that Q has not received $\langle OUT.x\rangle$. This is information flow which cannot be exploited by Q to send meaningful information but it is information flow according to Sutherland.

$\triangle$

Another way of thinking about the information flowing backwards is that an observer at P knows something about the behaviour of Q. If P does not send the order "Sack Smith" then an observer at P knows that Q will never sack Smith. Sutherland is really defining that no information flow occurs between machines if and only if they are isolated with no communication. This is a variation on secure isolation as defined by John Rushby [Rushby 82]. In chapter 5 we defined non-deducibility in the abstract category $\mathcal{IF}$. An example was given in chapter 5 which demonstrated that non-deducibility is not in general preserved by parallel composition. By the above discussion it can be seen that if two systems do not interact with each other, that is they are separable, then their parallel composition will preserve non-deducibility.

The seminal papers for the information flow approach to computer security are [Goguen & Meseguer 82] and [Goguen & Meseguer 84]. Goguen and Meseguer developed a non-interference relation between the interfaces of systems modelled as deterministic state machines. The property of non-interference is intended to capture the idea that behaviour of a "high level" user should not affect the behaviour of a system as observed by a low level user.

In [Goguen & Meseguer 84] inference control for databases and the use of non-interference is discussed. The approach of this thesis is similar in spirit to that taken in [Goguen & Meseguer 84]. Using the notion of non-interference it discusses how complex security policies could be expressed as non-interference assertions and systems could be verified against such a specification. The research presented in this thesis is concerned with expressing complex security requirements as statements about inference and demonstrating how systems can be verified against them.

The research reported in [Goguen 92] is closest to the categorical approach taken in this thesis although it is principally concerned with the semantics of concurrently

interacting objects. The paper uses sheaves to give a model for concurrent objects, each having its own local state and methods of interaction. Objects inherit from each other by sheaf morphisms. In terms of the categorical model in this thesis an object is an confidentiality statement over a set of observations. The observations are required to be ordered which can be done in concrete categories such as Safety-Iflow. Natural transformations between equivocation propositions are sheaf morphisms. For example if we have two systems P and Q in Safety-Iflow and there is a natural transformation

$$\textbf{INFER}\ W\ P \stackrel{\bullet}{\rightarrow} \textbf{INFER}\ W\ Q$$

then this is in effect the restriction of P to the alphabet of Q. If the events in the alphabet of the system are thought of as methods then P inherits the methods of Q and may have some more methods of its own. The sheaf semantic approach to describing concurrent objects or systems is more general than the categorical approach described in this thesis. There is even a general definition of non-interference given in terms of sheafs. It would be interesting to see whether the results of this thesis would generalise to sheafs. There is some justification to think it would because of the connection between sheaf theory theory and intuitionistic logic.

Logics for security

In [Glasgow & McEwen 88] a logic of knowledge is defined, the model for the logic are the possible worlds given by operator nets. An operator net consists of a set of nodes, a set of arcs, and a set of equations which relate the output arcs of nodes to functions or operators applied to input arcs. The behavioural semantics of operator nets correspond to communication in a distributed system.

A similar approach to security is taken in [Cuppens 90]. In this paper modal operators for time, knowledge and right to know provide a framework to reason about security in an abstract manner. The model for this logic is the behaviours of valued systems, [Eizenberg 89]. A valued system is intended to represent a large variety of dynamic systems.

The knowledge calculus approach to formalizing security requirements must be the right way to go if formal security policies are to be made understandable and therefore convincing to a customer.

9.3 Future Work

The research presented in this thesis addresses only some of the problems in developing secure computer systems. It says nothing about how a security requirement should be arrived at. This problem is addressed by [Dobson & McDermid 89] using a technique called Enterprise modelling. With this technique the risks and vulnerabilities of a system can be examined. Only once such a risk assessment has been done can a sensible security requirement be stated. It would be useful to undertake a case study combining Enterprise modelling with the formalization of security policies. Once formalized the policies can be reasoned about and desirable properties shown.

It would be nice to extend the number of concrete categories. For example the relational model of a database seems a promising concrete category. Relational merge would correspond to categorical product and the projection of fields from a record would give arrows between different relations. Another possibility is to use the Galois correspondence of chapter 5 to prove laws for the composition of CSP processes and confidentiality specifications within the concrete category Safety-Iflow.

Appendix A

Case study components

A CCIS is required to manage the distribution of boots. There are eight participating groups in this CCIS, called Boots, namely: *HQ*; *Stock-cell*; *Transport*; *Movement-cell*; *Warehouse*; *Exercise*; *Operator* and *Security officer*. Each group has a different rôle in the operation of Boots.

The events which make up the potential observations which can be made of the CCIS are called windows. Each group is identified by its window, that is what it can observe, and therefore the name given to each window will be the name of the associated group. The groups *HQ* and *Stock-cell* have already been dealt with in chapter 8.

Movement-cell The *Movement-cell* window is used by a section whose job it is to decide how many trucks are needed to move N boots from A to B. No other information is required.

Warehouse The *Warehouse* window is for managers at various warehouses to receive orders to prepare N boots of a certain type for collection. The managers also inform the stock control cell of the arrival of new boots.

Transport *Transport* is a window to the transport depot. The depot needs to know how many lorries are needed to transport boots between their source and destination. *Transport* knows where boots are going so can guess that if they are destined for the North pole they are arctic boots. What *Transport* does not know is whether these boots are for "operation snowman" or "operation snowball", it is this ignorance which should be preserved.

Operator The *Operator* window allows an operator to archive the system audit trail. The audit trail is a record of messages passed between different users of the system. The archived audit trail allows blame to be attributed for clerical errors. In a real system an operator would backup all information held on a system but the Boots system is assumed simpler to illustrate only the principle.

Security officer The *Security officer* can inspect the source and destination of messages in the audit trail, in order to monitor unusual traffic, but must not be able to deduce the content of messages passed between users. The audit trail must

not be changed except by adding further interactions to it. The security officer can also downgrade prepared messages on request. For example an aide might submit a classified order with a purpose which is not classified, by mistake. The security officer can downgrade the order on request by the aide from *HQ*.

Exercise The *Exercise* window simulates the windows of *HQ* and *Warehouse*. Exercise information can move lorries around but must not corrupt real data such as the distribution of boots.

The security requirement is stated in terms of what an observer at a window is allowed to know about the actions of an agent at another window. For brevity statements such as "an agent at window A is not allowed to know about the actions of an agent at window B", will be paraphrased by "window A is not allowed to know about the actions of window B". It is important to remember that it is agents or observers at a window who pass and gain information, not windows themselves.

Each sub-window is identified with separate sub-sections of stock control. Similarly the *Exercise* window is partitioned into *HQ* (exercise) and *Warehouse (exercise)*. The sub-window *HQ* (exercise) is not partitioned by high and low exercise orders. To participate in exercises, without corrupting real data, the window *Stock-cell* is partitioned into *Stock-cell (real)* and *Stock-cell (exercise)*. For the same reason *Movement-cell* is partitioned into *Movement-cell (real)* and *Movement-cell (exercise)*.

Statements of the security requirement are listed below and include requirements for confidentiality and for integrity. The reason for each statement is discussed afterwards.

2. The windows: *Movement-cell*; *Transport*; *Exercise*; *Operator* and *Warehouse* **must not know the purpose of an order** from *HQ*.

3. The audit trail **must not be corrupted.**

4. The window *Operator* **must not know** the audit trail.

5. The *Security officer* **must not know the content of messages** passed between users.

6. The windows: *Stock-cell (real)*; *Movement-cell (real)* and *Warehouse* **must not be corrupted by** the windows: *Exercise*; *Stock-cell (exercise)*; and *Movement-cell (exercise)*.

The first security requirement has already been dealt with in chapter 8. The second security requirement reflects the desire that only the stock control cell is allowed to know the purpose of an order. The meaning of the third requirement is that the audit trail can only be modified by adding records of new interactions to it. The fourth security requirement reflects the desire that an *Operator* cannot read the audit trail. The fifth requirement captures the desire that the security officer only knows enough to do her job. The final requirement reflects the desire that exercise data must not

corrupt real data, although *Transport* can receive orders to really move lorries as part of an exercise.

The words in each security requirement will be formalized in section A.1. Note that the security requirements contain integrity constraints as well as confidentiality constraints. The integrity property of non-corruption of the audit trail is a safety property; whereas the integrity property of separation of exercise and real information is expressed as an information flow property.

A description of functionality

The functionality supported for each participating group is defined. Each group has a different rôle in the operation of Boots and this rôle and its associated functionality defines the potential observations which can be made by that group.

Stock-cell The requirement for the exercise part of the Stock-cell is the same as the requirement for its real world operation except that orders from *HQ* are simulated by *Exercise* (HQ). Similarly two types of message are sent, one to *Movement-cell (exercise)* and one to *Exercise (Warehouse)*. Exercise data is interrogated to determine what boots are available. Messages to *Movement-cell (exercise)* consist of the same parts as messages to *Movement-cell*. Messages to *Exercise (Warehouse)* consist of two parts, the number of boots and the kind of boots required.

Movement-cell On receiving a message from *Stock-cell*, or *Stock-cell (exercise)*, an inquiry is made of the system concerning the disposition of lorries, then a message is sent to *Transport* consisting of three parts: the pick up point; the destination; and which lorries are to be used.

Warehouse On receiving a message from *Stock-cell* the warehouse manager prepares the necessary boots for collection. When new boots are received the system is informed of this by a message consisting of two parts, the number of new boots and their type.

Transport On receiving a message from *Movement-cell*, *Transport* takes the appropriate action. When a lorry reports that a delivery has been completed the system is informed of this by a message consisting of the identity of the lorry and its position.

Operator At the window *Operator* a request for a backup of the audit trail can be made at any time. When the system is ready an operator is informed and when an operator loads a tape (or whatever medium is used to backup the audit trail) the system is informed. An operator is informed when the backup is complete.

Security officer *Security officer* receives an audit record of security relevant events when an archive of the system is made. The set of security relevant events are records consisting of which windows have communicated with each other.

When a request from *HQ* is received to downgrade an order the security officer can downgrade it and send it on to *Stock-cell*.

Exercise The window *Exercise* can prepare an order, like *HQ*, consisting of three parts: the number of boots needed; their destination and a purpose code name. *Exercise* can receive messages from *Stock-cell (exercise)* about boots needed for collection. A message can be sent to that part of the system supporting the exercise consisting of two parts, a number (denoting new boots received) and their type.

A.1 Formalizing requirements for security in a CCIS

The first security requirement has already been formalized in the main text.

The second security requirement

2. The windows: *Movement-cell*; *Transport*; *Exercise*; *Operator* and *Warehouse* **must not know the purpose of an order** from *HQ*.

More details about *The rest*, which consists of all the windows of the CCIS which should be ignorant of the purpose of an order whatever its classification, are given.

$$\mathit{The\ rest} \triangleq \mathit{Operator} \cup \mathit{Exercise} \cup \mathit{Transport} \cup \mathit{Movement\text{-}cell} \cup \mathit{Warehouse}$$

The union of the windows is used because this represents possible collaboration between the groups of observers. No more information can be inferred about the system at any subset of *The rest*.

To isolate the purpose component of an order the function *Pse*, which projects out the purpose component of an order, is defined.

$$\forall (R, P, C) : \mathit{Order} \cdot \mathit{Pse}(R, P, C) = P$$

Using the function *Pse* the confidentiality requirement for the window *The rest* can be defined.

$$\mathit{PURPOSE\_CONF} \triangleq \mathit{Pse}^*(tr \upharpoonright \mathit{Order}) \in \mathit{PURPOSE}^*$$

Note that Pse^* is the application of *Pse* to every element in a trace. This predicate requires that the most that can be inferred is that some purpose is associated with the order. The next predicate just requires consistency with an observation through the window *The rest*.

$$\mathit{The\_Rest\_CONSISTENCY} \triangleq tr \upharpoonright \mathit{The\ rest} = l$$

Finally a concession is made to allow an observer to determine how many orders have been issued. This weakening of the requirement is because the consequence of an order is visible and therefore the existence of an order cannot be kept secret.

$$SIZE \triangleq \#tr \restriction Order \leq \#l$$

If components of the CCIS are to satisfy this confidentiality requirement then this weakening must be allowed. Although this problem with the specification has been anticipated it could well have been the case that it would not have come to light until candidate components were compared with it. If this was so then the weakening could then have been introduced and the modified specification checked against the other confidentiality statements. The results of this exercise could then be presented to the customer to see whether this was acceptable. The maximum inference is again a conjunction of the predicates previously defined.

$$The\_Rest\_INF \triangleq \begin{cases} PURPOSE\_CONF \\ \wedge \\ The\_rest\_CONSISTENCY \\ \wedge \\ SIZE \end{cases}$$

This predicate is sufficient to formalize the requirement at the window *The Rest* but as in the first confidentiality statement the other windows must be taken in to account. The components which interact with the window *The rest* will not interact directly with the windows *Operator*, *Security officer* or *Low*, hence the constraint on these windows can be as strong as possible without affecting the chances of a component satisfying the second security requirement. The only window left is *Real*, this window is concerned with all the events which can take place when the CCIS is dealing with the real world and therefore intersects with the window *The rest*. The requirement for the window *Real* is of course the same as *The rest* except that the allowed inference must be consistent with the observations at *Real*.

$$Real\_CONSISTENCY \triangleq tr \restriction Real = l$$

where l is a free variable which will denote the local observation at the window *Real* in the confidentiality statement.

$$Real\_INF \triangleq PURPOSE\_CONF \wedge Real\_CONSISTENCY$$

With this definition the second security requirement can be stated.

$$\mathcal{SR}_2 \triangleq \begin{cases} \textbf{From } l : The\ rest^* \textbf{ MAY\_INFER } The\_rest\_INF \\ \quad \textbf{Over } tr : (The\ rest \cup Order)^* \\ \textbf{From } l : Real^* \textbf{ MAY\_INFER } Real\_INF \\ \quad \textbf{Over } tr : (Real \cup Order)^* \end{cases}$$

The third security requirement

3. The audit trail **must not be corrupted.**

The audit trail is a sequence of messages which cannot be inspected normally. In special circumstances the vault to the archived audit trail can be opened and the series of audits inspected.

$Audits \triangleq \{ archive.s \mid s : Boots^* \}$

Note that *Audits* is not a subset of *Boots* so that *Audits* is well defined, also note that for each event $archive.s$ in *Audits* it is s itself which is an audit trail.

The particular implementation defined later does not audit all events in *Boots*, for example requests for an audit are not audited. A restriction of the alphabet of *Boots* to auditable events is given.

$[Auditable]$

The precise characterization of *Auditable* is left until the fifth security requirement. The complete audit trail for a series of audits is obtained using the function $flatten$.

$\forall t : Audits \cdot flatten(\langle\rangle) = \langle\rangle$
$flatten(\langle archive.s\rangle^\frown t) = s^\frown flatten\ t$

If tr is a free variable over traces then the integrity of the audit trail is

$$AUDIT\ INTEGRITY = \left(\begin{array}{l} flatten\ (tr^\frown\langle archive.s\rangle \upharpoonright Audits) \\ = \\ tr^\frown\langle archive.s\rangle \upharpoonright Auditable \end{array} \right)$$

This predicate states that whenever an archive of an audit trail takes place, all the messages since the last archive are recorded. If the audit trail is corrupted then this predicate will not hold.

The fourth security requirement

4. The window *Operator* **must not know** the audit trail.

The method used to formalize this security requirement is the same as the method for the first two statements; first the events of *Operator* are specified.

$Operator \triangleq \{ready, mounted, complete\}$

When the system is *ready* for a backup it informs an operator; the operator loads a tape, for example, and engages in the event *mounted*. The system archives the audit trail using the events drawn from *Audits* and then informs an operator that the backup is complete by sending the message *complete*. To formalize the fourth security requirement a function describing the limit of inference is defined.

$$infer\_archive(\langle\rangle) = \{\langle\rangle\}$$

$$infer\_archive(l^\frown\langle ready\rangle) = \{ t^\frown\langle ready\rangle \mid t \in infer\_archive(l) \}$$

$$infer\_archive(l^\frown\langle mounted\rangle) = \{ t^\frown\langle mounted\rangle \mid t \in infer\_archive(l) \}$$

$$infer\_archive(l^\frown\langle complete\rangle) = \left\{ t^\frown\langle archive.s, complete\rangle \;\middle|\; \begin{array}{l} t \in infer\_archive(l) \\ \wedge \\ archive.s \in Audits \end{array} \right\}$$

When the events *ready* and *mounted* are observed then an observer must be allowed to know that they have occurred. When the event *complete* is observed then the audit trail has been archived. The function limits an observer at *Operator* to only knowing that when the event *complete* is observed then the audit trail has been archived, but not knowing what has been audited. The predicate describing the confidentiality requirement for the window *Operator* is:

$AUDIT\_CONF \triangleq tr \in infer\_archive(l)$

The usual predicate requiring consistency is also defined.

$Operator\_CONSISTENCY \triangleq tr \upharpoonright Operator = l$

The total requirement for the single window *Operator*is obtained by conjoining the two previous predicates.

$$Operator\_INF \triangleq AUDIT\_CONF \wedge Operator\_CONSISTENCY$$

There should only be one component which is needed to perform backups on behalf of operators and this has no need to interact directly with the other windows except $The\ rest$. To enforce this, the strongest possible confidentiality requirement is put on the other windows, that is the only information which can be inferred is that which has been already observed at a particular window.

The window $The\ rest$ must be considered separately because the window *Operator* is a subwindow. To formalize the confidentiality requirement at this window a function similar to *infer_archive* is defined.

$ded\_archive(\langle\rangle) = \{\langle\rangle\}$
$\forall\, l^\frown\langle e\rangle \in The\ rest \cdot$

$$ded\_archive(l^\frown\langle e\rangle) = \begin{cases} \{\, t^\frown\langle a, e\rangle \mid t \in ded\_archive(l) \wedge a \in Audits \,\} \\ \textbf{If } e = complete \\ \{\, t^\frown\langle e\rangle \mid t \in ded\_archive(l) \,\} \\ \textbf{else} \end{cases}$$

The function *ded_archive* is similar to *infer_archive* except that it is defined over the larger set $The\ rest$. The predicates describing confidentiality, consistency and the total requirement follow the same form as before.

$$Rest\_INF \triangleq The\_rest\_CONSISTENCY \wedge REST\_CONF$$

where $The\_rest\_CONSISTENCY$ is as defined for the security requirement $\mathcal{SR}_2$ and

$REST\_CONF \triangleq tr \in ded\_archive(l)$

The formalization of the fourth security requirement can now be stated.

$$\mathcal{SR}_4 \triangleq \begin{cases} \textbf{From } l : Op\_interface \\ \quad \textbf{MAY\_INFER } Operator\_INF \textbf{ Over } tr : (Operator \cup Audits)^* \\ \textbf{From } l : The\ rest^* \mid l \upharpoonright Operator \in Rest\_interface \\ \quad \textbf{MAY\_INFER } Rest\_INF \textbf{ Over } tr : (The\ rest \cup Audits)^* \end{cases}$$

where

$$Op\_interface \triangleq \{ t : Operator^* \mid t \leq^\frown /\{\langle ready, mounted, complete\rangle\}^* \}$$

and

$$Rest\_interface \triangleq \{ t : Operator^* \mid t \leq^\frown /\{\langle ready, mounted, complete\rangle\}^* \}$$

This confidentiality specification limits the observations which can be made at the window *Operator* to cycles in an archive transaction. This is a special case of the use of design information to simplify the demonstration of compliance later on. With the three events *ready*, *mounted* and *complete* already known, there is only one sensible way in which they can occur. The confidentiality specification limits what an observer can know to "that at least as many archive events have occurred as *complete* events". An observer has no knowledge about the content of the audit trail, even from the order of the sequence of operator events. At the other windows, apart from *The rest*, the confidentiality specification states that an observer should not be able to gain any knowledge at all.

The fifth security requirement

5. The *Security officer* **must not know the content of messages** passed between users.

The *Security officer* can monitor security relevant events, these are messages between different windows. The security requirement disallows the *Security officer* from knowing the content of messages passed between windows so the security events that can be observed are sanitized forms of a message. Messages from *HQ* to the *Security officer* and from the *Security officer* to *Stock-cell* are of course known to the *Security officer*, therefore these messages are not security events. The set of security events is a set of pairs, each pair denotes the existence of some communication between parts of the CCIS. For example the pair $(HQ, stock)$ denotes the existence of a communication from HQ to the stock cell.

$$Security\ events \triangleq \left\{ \begin{array}{l} (HQ, stock), (stock, movement), \\ (stock, exercise), (exercise, stock), \\ (stock, warehouse), (movement, transport) \end{array} \right\}$$

To formally state that the *Security officer* must not be able to deduce, using the set *Security events*, the content of messages, more details about the relevant windows must be given.

To submit an order from *HQ* to the CCIS a destination for the order is needed. An over classified order is sent to the *Security officer* for downgrading, a correctly classified order is sent to *Stock-cell*.

$HQ \supset \{ (Security, o), o \mid o : Order \}$

Orders received from *HQ* for re-grading are sent on to *Stock-cell* after approval.

$$\mathit{Security\ officer} \triangleq \begin{cases} \{\mathit{re\text{-}grade}\} \times \mathit{Order} \cup \\ \{\mathit{graded}\} \times \mathit{Order} \cup \\ \{\, \mathit{trail}.s \mid s : \mathit{Security\ events}^* \,\} \end{cases}$$

Move order is a subset of *Stock-cell*, it is the set of orders from *Stock-cell* sent to the *Movement-cell*. *Boot order* is a subset of *Stock-cell*, it is the set of requests from *Stock-cell* submitted to the system for a Warehouse manager. *Lorry order* is a subset of *Movement-cell*, it is the set of orders from the *Movement-cell* submitted to the system for *Transport*.

With the above elaborations, about the windows onto the system, a function $filter$ can be defined which sanitizes a message between the windows visible to the *Security officer*.

$$\mathit{filter}(e) = \begin{cases} (HQ, stock) \\ \textbf{if } e \in \{stk\} \times Order \\ (stock, movement) \\ \textbf{if } e \in Move\ order \\ (stock, warehouse) \\ \textbf{if } e \in Boot\ order \\ (movement, transport) \\ \textbf{if } e \in Lorry\ order \\ (stock, exercise) \\ \textbf{if } e \in \{ex\} \times Boot\ order \\ (exercise, stock) \\ \textbf{if } e \in \{ex\} \times Order \end{cases}$$

The actual form of the sanitization of a sequence of messages is:

$$\begin{aligned} &\mathit{sanitize}\ \langle\rangle = \langle\rangle \\ &\mathit{sanitize}\ \langle e\rangle^\frown t = \begin{cases} \langle \mathit{filter}(e)\rangle^\frown \mathit{sanitize}\ t & \textbf{If } e \in \mathit{dom\ filter} \\ \mathit{sanitize}\ t & \textbf{else} \end{cases} \end{aligned}$$

With the definition of sanitize the set of events in *Auditable*, first introduced in the third security requirement, can be defined.

$$\begin{aligned} \mathit{Auditable} \triangleq\ & \{stk\} \times Order \cup\ Move\ order \\ & \cup\ Boot\ order \cup\ Lorry\ order \\ & \cup\ (\{ex\} \times Boot\ order) \cup\ (\{ex\} \times Order) \end{aligned}$$

With the definition of *Auditable* a function can be defined which describes the limit on the inference that can take place at the window *Real* and the subwindow *Security officer*.

$$\mathit{ded\_trail}(\langle\rangle) = \{\langle\rangle\}$$

$$\forall e : Real | e \neq trail.s \ \cdot \ ded\_trail(l^\frown\langle e\rangle) = \{\, r^\frown\langle e\rangle \ \mid \ r \in ded\_trail(l)\,\}$$

$$ded\_trail(l^\frown\langle trail.s\rangle) = \left\{ \ r^\frown t^\frown\langle trail.s\rangle \ \middle| \begin{array}{l} r \in ded\_trail(l) \\ \wedge \\ t \in Auditable^* \\ \wedge \\ sanitize(t) = s \end{array} \right\}$$

This function limits the inference and is expressed as a predicate.

$$SEC\_CONF = tr \in ded\_trail(l)$$

The predicate is in effect stating that an observer at the window *Security officer* knows that when the event *trail.s* is observed then a request for the trail must have occurred but any sequence of submitted orders consistent with the sanitized trail may have also occurred. The predicate does not forbid the *Security officer* from knowing the content of a message submitted for re-grading or its purpose. The predicate does allow the *Security officer* to know that an order submitted for re-grading comes from *HQ*. The strongest possible constraint is placed on all the other windows to complete the confidentiality specification. The fifth security requirement over *Security officer* is now formalized.

$$\mathcal{SR}_5 \ \triangleq \ \left\{ \begin{array}{l} \textbf{From } l : Security\ officer^* \\ \quad \textbf{MAY\_INFER } tr \upharpoonright Security\ officer = l \wedge SEC\_CONF \\ \quad\quad \textbf{Over } tr : (Security\ officer \cup Auditable)^* \\ \\ \textbf{From } l : Real^* \ \textbf{MAY\_INFER } tr \upharpoonright Real = l \wedge SEC\_CONF \\ \quad\quad \textbf{Over } tr : (Real \cup Auditable)^* \end{array} \right.$$

The confidentiality statement $\mathcal{SR}_5$ places a limit on the knowledge an agent can have over sequences of observed events. The most an observer can infer is between what windows messages have been passed and in the special case of a message requiring downgrading, the message content.

The sixth security requirement

6. The windows: *Stock-cell (real)*; *Movement-cell (real)* and *Warehouse* **must not be corrupted by** the windows: *Exercise*; *Stock-cell (exercise)*; and *Movement-cell (exercise)*.

The non-exercise operation of *Stock-cell* that is when the stock cell is dealing with real world situations as opposed to exercise simulations, is defined as

$$Stock\text{-}cell\ (real) \ \triangleq \ Order \cup Stock\ op$$

The window *Movement-cell* consists of orders received from *Stock-cell* inquiries about the disposition of lorries and orders to *Transport* to move boots.

$$Movement\text{-}cell\ (real) \ \triangleq \ Move\ order \ \cup \ Lorry\_Id \ \cup \ Lorry\_Order$$

Exercise data will legally modify a database of the positions of lorries by ordering boots to be moved. For convenience the users dealing with real data, which should be unaffected by exercise data, are grouped together, *Lorry_Id* and *Transport* are not included because they interact with exercises.

$$Real \triangleq Stock\text{-}cell\ (real) \cup Move\ order \cup Warehouse \cup Lorry\_Order$$

The exercise sub-windows of *Stock-cell* and *Movement-cell* are re-labellings of *Stock-cell (real)* and *Move order* respectively. For convenience data from exercise windows are grouped together.

$$Boots\ (exercise) \triangleq Exercise \cup Stock\text{-}cell\ (exercise) \cup Movement\text{-}cell\ (exercise)$$

With these definitions an integrity requirement can be described for the windows *Operator*, *Security officer*, *Low* and the compound window *Real*. The consistency requirement is expressed in the confidentiality statement for simplicity. The integrity requirement is expressed as a confidentiality requirement.

$$BOOTS\_INTEGRITY = tr \upharpoonright Boots(exercise) \in Boots(exercise)^*$$

The predicate says that no exercise data could have influenced observed real data; because if it did then one would be able to narrow the possible set of exercise operations which could have taken place. If for example one exercise operation could modify some data held on a database then, from the construction of the system, one could in principle deduce which instruction could change the database. If all the exercise operations modified the database in the same way then one could still deduce one bit of information, that an exercise operation had occurred. The predicate is effectively saying that exercise and real operations are carried out on two conceptually (or physically) separated machines, sharing only the lorry resource. The sixth and final security requirement is now formalized.

$$\mathcal{SR}_6 \triangleq \begin{array}{l} \textbf{From}\ l : Real^* \ \textbf{MAY\_INFER}\ tr \upharpoonright Real = l \wedge BOOTS\_INTEGRITY \\ \quad \textbf{Over}\ tr : (Boots(exercise) \cup Real)^* \end{array}$$

A.2 Static representations of the requirements

The security requirement $\mathcal{SR}_2$

The main concern of the confidentiality specification $\mathcal{SR}_2$ was how much could be inferred from an observation of operations from *The rest* and *Real*.

The window *The rest*

The first step in building a static description is to describe what observations can be made.

$$SR2_{The\ rest} \triangleq e : The\ rest \rightarrow SR2_{The\ rest}$$

The interface object $SR2_{The\ rest}$ allows any operation classified from any of the windows in the compound window *The rest* to be performed. To express the desired inference in a static way an inference object for $SR2_{The\ rest}$ is needed.

$$K\_SR2_{The\ rest} \triangleq e : (The\ rest \cup Order) \rightarrow K\_SR2_{The\ rest}$$

The inference object $K\_SR2_{The\ rest}$ of $SR2_{The\ rest}$ legislates that any operation from $The\ rest \cup Order$ is possible. No order interferes with any sequence of operations from *The rest*. $K\_SR2_{The\ rest}$ can easily be shown to be an inference object of $SR2_{The\ rest}$ by applying the CSP restriction operator to the process $K\_SR2_{The\ rest}$.

To check that the interface object $SR2_{The\ rest}$ and its inference object $K\_SR2_{The\ rest}$ give at least as much confidentiality as the specification $\mathcal{SR}_2$ over observations drawn from $The\ rest^*$ they must be compared as functions. The confidentiality specification $\mathcal{SR}_2$ is

From $l : The\ rest^*$ **MAY_INFER** *The_rest_INF* **Over** $tr : (The\ rest \cup Order)^*$
From $l : Real^*$ **MAY_INFER** *Real_INF* **Over** $tr : (Real \cup Order)^*$

As a function

$$\mathcal{SR}_2(l) = \{\, tr : (The\ rest \cup Order)^* \mid tr \upharpoonright The\ rest = l \wedge \#(tr \upharpoonright Order) \leq \#l \,\}$$

for every $l : The\ rest^*$.

The traces of $SR2_{The\ rest}$ can be calculated

$$traces(SR2_{The\ rest}) = The\ rest^*$$

therefore the domain of $\mathcal{SR}_2$ is the same as the interface $SR2_{The\ rest}$. For an observation l of the interface the associated inference is

$$\{\, tr : (The\ rest \cup Order)^* \mid tr \upharpoonright The\ rest = l \wedge tr \in traces(K\_SR2_{The\ rest}) \,\}$$

but

$$traces(K\_SR2_{The\ rest}) = (The\ rest \cup Order)^*$$

therefore the associated inference is

$$\{\, tr : (The\ rest \cup Order)^* \mid tr \upharpoonright The\ rest = l \,\}$$

which is a superset of $\mathcal{SR}_2$, therefore $SR2_{The\ rest}$ and its inference object are stronger than $\mathcal{SR}_2$.

The window *Real*

The interface is

$$SR2_{Real} \triangleq e : Real \rightarrow SR2_{Real}$$

this allows any operation which is concerned with the real world operation of the CCIS to be performed. To express the desired inference in a static way an inference object for $SR2_{Real}$ is needed.

$$K\_SR2_{Real} \triangleq e : (Real \cup Order) \rightarrow K\_SR2_{Real}$$

The inference object K_SR2_{Real} of $SR2_{Real}$ legislates that any operation from $Real \cup Order$ is possible. No order interferes with any sequence of operations from $Real$. K_SR2_{Real} can easily be shown to be an inference object of $SR2_{Real}$ by applying the CSP restriction operator to the process K_SR2_{Real}.

To check that the interface object $SR2_{Real}$ and its inference object K_SR2_{Real} give at least as much confidentiality as the specification $\mathcal{SR}_2$ over observations drawn from $Real^*$ they must be compared as functions. The confidentiality specification $\mathcal{SR}_2$ is

From $l : The\ rest^*$ **MAY_INFER** The_rest_INF **Over** $tr : (The\ rest \cup Order)^*$
From $l : Real^*$ **MAY_INFER** $Real\_INF$ **Over** $tr : (Real \cup Order)^*$

As a function

$$\mathcal{SR}_2(l) = \{\, tr : (Real \cup Order)^* \mid tr \upharpoonright Real = l \,\}$$

for every $l : Real^*$.

The traces of $SR2_{Real}$ can be calculated

$$traces(SR2_{Real}) = Real^*$$

therefore the domain of $\mathcal{SR}_2$ is the same as the interface $SR2_{Real}$. For an observation l of the interface the associated inference is

$$\{\, tr : (Real \cup Order)^* \mid tr \upharpoonright Real = l \wedge tr \in traces(K\_SR2_{Real}) \,\}$$

but

$$traces(K\_SR2_{Real}) = (Real \cup Order)^*$$

therefore the associated inference is

$$\{\, tr : (Real \cup Order)^* \mid tr \upharpoonright Real = l \,\}$$

which is a superset of $\mathcal{SR}_2$, therefore $SR2_{Real}$ and its inference object are stronger than $\mathcal{SR}_2$ over the window $Real$.

The other interfaces

The other interfaces and their inference objects are very simple to specify. For example

$$SR2_{Low} \triangleq e : Low \rightarrow SR2_{low}$$

since like $SR2_{Real}$ there are no restrictions on what observations can be made at this interface. No more information is allowed to be inferred at the interface *Low*, therefore the inference object is exactly the same as the interface object.

$$K\_SR2_{Low} \triangleq SR2_{Low}$$

The traces of these objects can be easily calculated and shown to give a function which is equivalent to the confidentiality specification over observations drawn from Low^*. The static representation of confidentiality over the remaining interfaces: *Operator* and *Security officer* are expressed in a similar manner to $SR2_{Low}$ and its inference object K_SR2_{Low}.

$\mathcal{SR}_4$

The main concern of the confidentiality specification $\mathcal{SR}_4$ was how much could be inferred from an observation of operations from *The rest* and *Operator*.

The window *The rest*

The first step in building a static description is to describe what observations can be made.

$$\begin{aligned} SR4_{The\ rest} &\triangleq e : (The\ rest - Operator) \rightarrow SR4_{The\ rest} \\ &\quad [] \\ &\quad ready \rightarrow S_1 \\ S_1 &\triangleq e1 : (The\ rest - Operator) \rightarrow S_1 \\ &\quad [] \\ &\quad mounted \rightarrow S_2 \\ S_2 &\triangleq e2 : (The\ rest - Operator) \rightarrow S_2 \\ &\quad [] \\ &\quad complete \rightarrow SR4_{The\ rest} \end{aligned}$$

The interface object $SR4_{The\ rest}$ allows any operation in the compound window *The rest* to be performed, except for the Operator's operations which must cycle through the sequence *ready*, *mounted* and *complete*. To express the desired inference in a static way an inference object for $SR4_{The\ rest}$ is needed.

$$\begin{aligned} K\_SR4_{The\ rest} &\triangleq e : (The\ rest - Operator) \rightarrow K\_SR4_{The\ rest} \\ &\quad [] \\ &\quad ready \rightarrow K_1 \end{aligned}$$

$$K_1 \triangleq e1 : (The\ rest - Operator) \rightarrow K_1 \ [] \ mounted \rightarrow K_2$$

$$K_2 \triangleq e2 : (The\ rest - Operator) \rightarrow K_2 \ [] \ a : Audit \rightarrow complete \rightarrow K\_SR4_{The\ rest}$$

The first alternative of the inference object $K\_SR4_{The\ rest}$ legislates that any operation from *The rest* is possible except for the Operator's operations. The second alternative of $K\_SR4_{The\ rest}$ legislates that the event *ready*, denoting the start of an archive, can occur. The first alternative of K_1 is similar to $K\_SR4_{The\ rest}$ except that the second alternative of K_1 legislates that the event *mounted* can occur. Finally K_2 is similar to K_1 except that the second alternative of K_2 legislates that an archive of an audit trail can occur followed by the event *complete* returning to the original state $K\_SR4_{The\ rest}$. $K\_SR4_{The\ rest}$ can easily be shown to be an inference object of $SR4_{The\ rest}$ by applying the CSP restriction operator to the process $K\_SR4_{The\ rest}$.

To check that the interface object $SR4_{The\ rest}$ and its inference object $K\_SR2_{The\ rest}$ give at least as much confidentiality as the specification $\mathcal{SR}_4$ over observations drawn from $The\ rest^*$ they must be compared as functions. The confidentiality specification $\mathcal{SR}_4$ is

From $l : Op\_interface$ **MAY_INFER** $Operator\_INF$ **Over** $tr : (Operator \cup Audits)^*$
From $l : The\ rest^* \mid Rest\_interface$ **MAY_INFER** $Rest\_INF$
Over $tr : (The\ rest \cup Audits)^*$

As a function

$$\mathcal{SR}_4(l) = \{\, tr : (The\ rest \cup Audits)^* \mid tr \upharpoonright The\ rest = l \wedge tr \in ded\_archive(l) \,\}$$

for every $l : The\ rest^*$.

The traces of $SR4_{The\ rest}$ can be calculated

$$traces(SR4_{The\ rest}) = \{\, tr : The\ rest^* \mid tr \upharpoonright Operator \leq^\frown / \{\langle ready, mounted, complete \rangle\}^* \,\}$$

therefore the domain of $\mathcal{SR}_4$ is the same as the interface $SR4_{The\ rest}$. For an observation l of the interface the associated inference is

$$\{\, tr : (The\ rest \cup Order)^* \mid tr \upharpoonright The\ rest = l \wedge tr \in traces(K\_SR4_{The\ rest}) \,\}$$

but

$$traces(K\_SR4_{The\ rest}) = \left\{ tr : (The\ rest \cup Audit)^* \;\middle|\; \begin{array}{l} tr \upharpoonright (Operator \cup Audit) \\ \leq \\ ^\frown / \{\langle ready, mounted, a, complete \rangle\}^* \\ \wedge \\ a : Audits \end{array} \right\}$$

therefore the associated inference is

$$\left\{ tr : (The\ rest \cup Audit)^* \;\middle|\; \begin{array}{l} tr \upharpoonright The\ rest = l \\ \wedge \\ tr \upharpoonright (Operator \cup Audit) \\ \leq \\ \frown / \{\langle ready, mounted, a, complete \rangle\}^* \\ \wedge \\ a : Audits \end{array} \right\}$$

which, by definition of *ded_archive*, equals $\mathcal{SR}_4$. $SR4_{The\ rest}$ and its inference object are equivalent to $\mathcal{SR}_4$.

The window *Operator*

The interface is

$$SR4_{Operator} \triangleq ready \rightarrow mounted \rightarrow complete \rightarrow SR4_{Operator}$$

To express the desired inference in a static way an inference object for $SR4_{Operator}$ is needed.

$$K\_SR4_{Operator} \triangleq ready \rightarrow mounted \rightarrow a : Audits \rightarrow complete \rightarrow K\_SR4_{Operator}$$

The inference object $K\_SR4_{Operator}$ of $SR4_{Operator}$ legislates that before the end of the archive cycle an audit trail is archived, but there is no way of telling what is in the audit trail. $K\_SR4_{Operator}$ can easily be shown to be an inference object of $SR4_{Operator}$ by applying the CSP restriction operator to the process $K\_SR4_{Operator}$.

To check that the interface object $SR4_{Operator}$ and its inference object $K\_SR4_{Operator}$ give at least as much confidentiality as the specification $\mathcal{SR}_4$ over observations drawn from $Operator^*$ they must be compared as functions. The confidentiality specification $\mathcal{SR}_4$ as a function is

$$\mathcal{SR}_4(l) = \{\, tr : (Operator \cup Audits)^* \mid tr \upharpoonright Operator = l \wedge tr \in infer\_archive(l) \,\}$$

for every $l : Op\_interface$.

The traces of $SR4_{Operator}$ can be calculated

$$traces(SR4_{Operator}) = Op\_interface$$

since design knowledge was used to restrict the observations $\mathcal{SR}_4$ legislated over. Not surprisingly the domain of $\mathcal{SR}_4$ is the same as the interface $SR4_{Operator}$. For an observation l of the interface the associated inference is

$$\{\, tr : (Real \cup Audit)^* \mid tr \upharpoonright Real = l \wedge tr \in traces(K\_SR4_{Operator}) \,\}$$

but with a structural induction it can be shown that the associated inference is

$$\{\, tr : (Real \cup Order)^* \mid tr \upharpoonright Real = l \wedge tr \in infer\_archive(l) \,\}$$

which equals $\mathcal{SR}_4$, therefore $SR4_{Operator}$ and its inference object are equivalent to $\mathcal{SR}_4$ over the window *Operator*.

The other interfaces

The other interfaces and their inference objects are very simple to specify. For example

$$SR4_{Low} \triangleq e : Low \rightarrow SR4_{low}$$

No more information is allowed to be inferred at the interface *Low*, therefore the inference object is exactly the same as the interface object.

$$K\_SR4_{Low} \triangleq SR4_{Low}$$

The traces of these objects can be easily calculated and shown to give a function which is equivalent to the confidentiality specification over observations drawn from Low^*. The static representation of confidentiality over the remaining interfaces: *Operator* and *Security officer* are expressed in a similar manner to $SR4_{Low}$ and its inference object K_SR4_{Low}.

$\mathcal{SR}_5$

The main concern of the confidentiality specification $\mathcal{SR}_5$ was how much could be inferred from an observation of operations from *Security officer*.

The window *Operator*

The interface is

$$SR5_{Security\ officer} \triangleq e : Security\ officer \rightarrow SR5_{Security\ officer}$$

To express the desired inference in a static way an inference object for $SR5_{Security\ officer}$ is needed.

$$K\_SR5_{Security\ officer} \triangleq K\_SR5_{\langle\rangle}$$

$$K\_SR5_s \triangleq \left(\begin{array}{l} (re-grade, o) : \{re-grade\} \times Order \rightarrow K\_SR5_s \\ [\!] \\ (graded, o) : \{graded\} \times Order \rightarrow K\_SR5_s \\ [\!] \\ trail.(sanitize(s)) \rightarrow K\_SR5_{\langle\rangle} \\ [\!] \\ (a : Auditable \rightarrow K\_SR5_{s^\frown\langle a\rangle}) \end{array} \right)$$

The inference object $K\_SR5_{Security\ officer}$ of $SR5_{Security\ officer}$ legislates that any order can be received for re-grading and any order sent which has been graded. If a sanitized trail, t, is received then the auditable events which took place, since the last trail was received, must be consistent with it. $K\_SR5_{Security\ officer}$ is an inference object of $SR5_{Security\ officer}$ because

$$K\_SR5_{Security\ officer} \upharpoonright Security\ officer\ =$$
$$op : Security\ officer\ \rightarrow K\_SR5_{Security\ officer} \upharpoonright Security\ officer$$

To check that the interface object $SR5_{Security\ officer}$ and its inference object $K\_SR5_{Security\ officer}$ give at least as much confidentiality as the specification $\mathcal{SR}_5$ over observations drawn from $Security\ officer^*$ they must be compared as functions. The confidentiality specification $\mathcal{SR}_5$ as a function is

$$\mathcal{SR}_5(l) =$$
$$\{\, tr : (Security\ officer \cup Auditable)^* \mid tr \upharpoonright Security\ officer = l \wedge tr \in ded\_trail(l) \,\}$$

for every $l : Security\ officer$.

To show that $K\_SR5_{Security\ officer}$ gives the equivalent amount of inference as $\mathcal{SR}_5(l)$ is not a simple undertaking; it involves an elaborate structural induction which is omitted for reasons of space.

The other interfaces

The other interfaces and their inference objects are very simple to specify. For example

$$SR5_{Low} \triangleq e : Low \rightarrow SR5_{low}$$

No more information is allowed to be inferred at the interface Low, therefore the inference object is exactly the same as the interface object.

$$K\_SR5_{Low} \triangleq SR5_{Low}$$

The traces of these objects can be easily calculated and shown to give a function which is equivalent to the confidentiality specification over observations drawn from Low^*. The static representation of confidentiality over the remaining interfaces: *Operator* and *Security officer* are expressed in a similar manner to $SR5_{Low}$ and its inference object K_SR5_{Low}.

A.3 Design compliance

$MOVE\_QUEUE$

The next part of the system after *ORDER_QUEUE* to be dealt with is the queue of movement orders from *Stock-cell* to *Movement-cell*. Components of the system will

be shown to satisfy the second security policy statement in two ways. The queue of movement orders from *Stock-cell* to *Movement-cell* will be shown to satisfy the confidentiality specification $\mathcal{SR}_2$.

The same buffer design which was used for *ORDER_QUEUE* is chosen. There is a relationship between orders received by *Stock-cell* and movement orders received by *Movement-cell*. An order received by *Stock-cell* is drawn from the cartesian product

$$REQUEST \times PURPOSE \times Class.$$

The set of points between which lorries will pick up boots and deliver is

$$[Location]$$

the set includes every warehouse which can store boots. An order sent to *Movement-cell* is drawn from the cartesian product

$$Location \times REQUEST$$

since *REQUEST* denotes instructions to transport a number of boots to a particular destination; the *Location* field is determined by a stock control officer. Orders sent to *Movement-cell* are denoted by

$$Stock\ order \triangleq Location \times REQUEST.$$

$Stock\ order \subset Stock\ op$ which along with *High* and *Low* gave the window *Stock-cell*. Orders received by *Movement-cell* are denoted by

$$Move\ order \triangleq Location \times REQUEST$$

The alphabets *Stock order* and *Move order* are isomorphic but not identical, thus avoiding unwanted synchronization. A re-labelling function between *Stock order* and *Move order* is

$$stock\_label : Stock\ order \rightarrow Move\ order.$$

For a fixed number $n : \mathbb{N}$ the buffer of orders from *Stock-cell* to *Movement-cell* is defined.

$$MOVE\_QUEUE \triangleq MOVE\_QUEUE_{\langle\rangle}$$

$$MOVE\_QUEUE_{\langle\rangle} \triangleq so : Stock\ order \rightarrow MOVE\_QUEUE_{\langle so\rangle}$$

$$MOVE\_QUEUE_{s^\frown\langle so\rangle} \triangleq (so' : Stock\ order \rightarrow MOVE\_QUEUE_{\langle so'\rangle^\frown s^\frown\langle so\rangle}) \ [] \ (stock\_label(so) \rightarrow MOVE\_QUEUE_s)$$

such that $\#s < n - 1$

$$MOVE\_QUEUE_{s^\frown\langle so\rangle} \triangleq stock\_label(so) \rightarrow MOVE\_QUEUE_s$$

such that $\#s = n - 1$

The intersection of the alphabet of *MOVE_QUEUE* with *The rest* is *Move order*. The intersection of the alphabet of *MOVE_QUEUE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *MOVE_QUEUE* with *Real* is again *Move order*, this is because *The Rest* is a subset of *Real*. The only observable behaviour of *MOVE_QUEUE* is over the traces drawn from *The rest** and *Real**. This means that *MOVE_QUEUE* must be shown to satisfy $\mathcal{SR}_2$ because it legislates over *The rest* and *Real*. $\mathcal{SR}_6$ can be ignored because if *MOVE_QUEUE* satisfies $\mathcal{SR}_6$ then it will satisfy $\mathcal{SR}_6 \wedge_{\mathrm{IF}} \mathcal{SR}_2$. To show that *MOVE_QUEUE* satisfies the second security requirement it must be shown that

$$MOVE\_QUEUE \upharpoonright Real \cup Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

To show this, observations over *Real* and *The rest* need to be considered.

The rest

If $RO \triangleq Real \cup Order$ then

$$MOVE\_QUEUE \upharpoonright RO = mo : Move\ order \rightarrow MOVE\_QUEUE \upharpoonright RO$$

this describes the interface of *MOVE_QUEUE* as *Movement-cell* sees it. As far as *Movement-cell* is concerned it can receive a movement order from *Stock-cell* at any time. Comparing $MOVE\_QUEUE \upharpoonright RO$ with

$$K\_SR2_{The\ rest} \triangleq e : (The\ rest \cup Order) \rightarrow K\_SR2_{The\ rest}$$

of section A.2 shows that

$$K\_SR2_{The\ rest} \upharpoonright Move\ Order = MOVE\_QUEUE \upharpoonright RO$$

Real

The rest is a subset of *Real* and *Move order* is a subset of *The rest*, therefore

$$MOVE\_ORDER \upharpoonright RO$$

is the interface as far as the real world operation of Boots is concerned. To show satisfaction of $\mathcal{SR}_2$ all that needs to be shown is that

$$K\_SR2_{Real} \triangleq e : (Real \cup Order) \rightarrow K\_SR2_{Real}$$

has an arrow to $MOVE\_QUEUE \upharpoonright RO$. Restricting K_SR2_{Real} to *Move order* gives

$$K\_SR2_{Real} \upharpoonright Move\ order = MOVE\_QUEUE \upharpoonright RO$$

since *Move order* $\subset$ *Real*. The arrows from K_SR2_{Real} and $K\_SR2_{The\ Rest}$ to $MOVE\_QUEUE \upharpoonright RO$ mean that by corollary 9

$$MOVE\_QUEUE \upharpoonright RO \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

$INVOICE\_QUEUE$

The next system component to be examined is the queue of requests for boots from *Stock-cell* to *Warehouse*. First some details about the alphabet of the queue are given.

$$[Kind]$$

$Kind$ is the type of boot required as part of an order.

$$Boot\ order \triangleq \mathbb{N} \times Kind$$

Boot order is a request from *Stock-cell* for of a number of boots of a particular kind which are required for collection, note that $Boot\ order \subset Stock\ op$ which is part of the window *Stock-cell*.

$$W\ order \triangleq \mathbb{N} \times Kind$$

W order is a request from *Stock-cell* which has been received at *Warehouse*. The relationship between submitted and received orders is captured by a re-labelling isomorphism between *Boot order* and *W order*.

$$Wlabel : Boot\ order \rightarrow W\ order$$

$$\alpha INVOICE\_QUEUE \triangleq Boot\ order \cup W\ order$$

$$\begin{array}{rcl}
INVOICE_QUEUE & \triangleq & INVOICE_QUEUE_{\langle\rangle} \\
INVOICE_QUEUE_{\langle\rangle} & \triangleq & bo : Boot\ order \rightarrow INVOICE_QUEUE_{\langle bo\rangle} \\
INVOICE_QUEUE_{s^\frown\langle bo\rangle} & \triangleq & (bo' : Boot\ order \rightarrow INVOICE_QUEUE_{\langle bo'\rangle^\frown s^\frown\langle bo\rangle}) \\
 & & [\!] \\
 & & (Wlabel(bo) \rightarrow INVOICE_QUEUE_s) \\
 & & \\
 & & \#s < n-1 \\
 & & \\
INVOICE_QUEUE_{s^\frown\langle bo\rangle} & \triangleq & Wlabel(bo) \rightarrow INVOICE_QUEUE_s \\
 & & \\
 & & \#s = n-1
\end{array}$$

The intersection of the alphabet of *INVOICE_QUEUE* with *The rest* is *W order*. Like *MOVE_QUEUE* the intersection of the alphabet of *INVOICE_QUEUE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *INVOICE_QUEUE* with *Real* is again *W order*, because *The Rest* is a subset of *Real*. The only observable behaviour of *INVOICE_QUEUE* is over the traces drawn from *The rest** and *Real**. This means that *INVOICE_QUEUE* must

be shown to satisfy $\mathcal{SR}_2$ because it legislates over *The rest* and *Real*. $\mathcal{SR}_6$ can be ignored because if *MOVE_QUEUE* satisfies $\mathcal{SR}_6$ then it will satisfy $\mathcal{SR}_6 \wedge_{\text{IF}} \mathcal{SR}_2$. To show that *INVOICE_QUEUE* satisfies the second security requirement it must be shown that

$$INVOICE\_QUEUE \upharpoonright Real \cup Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

To show this, observations over *Real* and *The rest* need to be considered.

The rest

Using the abbreviation $RO = Real \cup Order$

$$INVOICE\_QUEUE \upharpoonright RO = wo : W\ order \rightarrow INVOICE\_QUEUE \upharpoonright RO$$

this describes the interface of *INVOICE_QUEUE* as *Warehouse* sees it. As far as *Warehouse* is concerned it can receive an order, from *Stock-cell*, to get boots ready for collection at any time.

Comparing $INVOICE\_QUEUE \upharpoonright RO$ with $K\_SR2_{The\ rest}$ of section A.2 shows that

$$K\_SR2_{The\ rest} \upharpoonright W\ order = INVOICE\_QUEUE \upharpoonright RO$$

Real

The rest is a subset of *Real* and *W order* is a subset of *The rest*, therefore

$$INVOICE\_ORDER \upharpoonright RO$$

is the interface as far as the real world operation of Boots is concerned. To show satisfaction of $\mathcal{SR}_2$ all that needs to be shown is that K_SR2_{Real} has an arrow to $INVOICE\_QUEUE \upharpoonright RO$. Restricting K_SR2_{Real} to *W order* gives

$$K\_SR2_{Real} \upharpoonright W\ Order = INVOICE\_QUEUE \upharpoonright RO$$

since $W\ order \subset Real$. The arrows from K_SR2_{Real} and $K\_SR2_{The\ Rest}$ to

$$INVOICE\_QUEUE \upharpoonright RO$$

means that by corollary 9

$$INVOICE\_QUEUE \upharpoonright RO \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

BOOT_DATABASE

The component *BOOT_DATABASE* is a database of the disposition of boots of different kinds at different locations. The alphabet of *BOOT_DATABASE* is

$$Kind \cup Kind \times \mathbb{P}(Location \times \mathbb{N}) \cup Kind \times Location \times \mathbb{N}$$

Enquiries from *Stock-cell* on the disposition of boots have the type *Kind*. The reply from the database is a tuple consisting of the kind of boot, where it is stored and how many there are, replies from the database to *Stock-cell* have type

$$Kind \times \mathbb{P}(Location \times \mathbb{N})$$

Warehouse updates the database when boots are moved from or are delivered to a warehouse, the updates have type

$$Kind \times Location \times \mathbb{N}$$

$$BOOT\_DATABASE \triangleq BOOT\_DATABASE_{reply_0}$$

$$BOOT\_DATABASE_{reply} \triangleq$$

$$\left(\begin{array}{l} k : Kind \rightarrow reply(k) \rightarrow BOOT\_DATABASE_{reply} \\ [] \\ (k, update) : Kind \times (Location \times \mathbb{N}) \rightarrow BOOT\_DATABASE_{reply \oplus \{k \mapsto update\}} \end{array} \right)$$

where $reply \triangleq Kind \rightarrow \mathbb{P}(Location \times \mathbb{N})$
and $reply_0$ is the initial state of the database.

The intersection of the alphabet of *BOOT_DATABASE* with *The rest* is

$$Kind \times Location \times \mathbb{N}$$

which is the set of updates to the database made by *Warehouse*. The intersection of the alphabet of *BOOT_DATABASE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *INVOICE_QUEUE* with *Real* is again

$$Kind \times Location \times \mathbb{N}$$

because *The Rest* is a subset of *Real*. The only observable behaviour of *BOOT_DATABASE* is over the traces drawn from *The rest** and *Real**. This means that *BOOTS_DATABASE* must be shown to satisfy $\mathcal{SR}_2$ because it legislates over *The rest* and *Real*. $\mathcal{SR}_6$ can be ignored because if *MOVE_QUEUE* satisfies $\mathcal{SR}_6$ then it will satisfy $\mathcal{SR}_6 \wedge_{\mathrm{IF}} \mathcal{SR}_2$. To show that *BOOTS_DATABASE* satisfies the second security requirement it must be shown that

$$BOOTS\_DATABASE \upharpoonright Real \cup Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

To show this, observations over *Real* and *The rest* need to be considered.

The rest

Using the abbreviation $RO = Real \cup Order$

$$BOOT\_DATABASE \upharpoonright RO = u : Kind \times (Location \times \mathbb{N}) \rightarrow BOOT\_DATABASE \upharpoonright RO$$

this describes the interface of *BOOT_DATABASE* as *Warehouse* sees it. As far as *Warehouse* is concerned it can do any update to the database at any time.

Comparing $BOOT\_DATABASE \upharpoonright RO$ with $K\_SR2_{The\ rest}$ of section A.2 shows that

$$K\_SR2_{The\ rest} \upharpoonright Kind \times (Location \times \mathbb{N}) = BOOT\_DATABASE \upharpoonright RO$$

Real

The rest is a subset of *Real* and

$$Kind \times (Location \times \mathbb{N}) \subset The\ rest$$

therefore $BOOT\_DATABASE \upharpoonright RO$ is the interface as far as the real world operation of Boots is concerned. To show satisfaction of $\mathcal{SR}_2$ all that needs to be shown is that K_SR2_{Real} has an arrow to $BOOT\_DATABASE \upharpoonright RO$. Restricting K_SR2_{Real} to $Kind \times (Location \times \mathbb{N})$ gives

$$K\_SR2_{Real} \upharpoonright Kind \times (Location \times \mathbb{N}) = BOOT\_DATABASE \upharpoonright RO$$

The arrows from K_SR2_{Real} and $K\_SR2_{The\ Rest}$ to $BOOT\_DATABASE \upharpoonright RO$ mean that by corollary 9

$$BOOT\_DATABASE \upharpoonright RO \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

LORRY_DATABASE

The component *LORRY_DATABASE* is a database of the disposition of the lorries, which carry the boots, at different locations. The alphabet of *LORRY_DATABASE* is

$$Lorry\_Id \cup Location \cup Lorry\_Id \times Location$$

Enquiries from *Movement-cell* on the disposition of boots have the type *Lorry_Id*. The reply from the database is the location of the lorry, replies from the database to *Movement-cell* have type *Location*. *Transport* updates the database when lorries move, the updates have type

$$Lorry\_Id \times Location$$

$$LORRY\_DATABASE \triangleq LORRY\_DATABASE_{pos_0}$$

$$LORRY\_DATABASE_{pos} \triangleq$$

$$\left(\begin{array}{l} id : Lorry\_Id \rightarrow pos(id) \rightarrow LORRY\_DATABASE_{pos} \\ [] \\ (id, loc) : Lorry\_Id \times Location \rightarrow LORRY\_DATABASE_{pos \oplus \{id \mapsto loc\}} \end{array} \right)$$

where $pos \triangleq Lorry\_Id \rightarrow Location$
and pos_0 is the initial state of the database.

The intersection of the alphabet of *LORRY_DATABASE* with *The rest* is the alphabet of *LORRY_DATABASE*, since the alphabet of *LORRY_DATABASE* is a subset of *The rest*. The intersection of the alphabet of *LORRY_DATABASE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *LORRY_DATABASE* with *Real* is again the alphabet of *LORRY_DATABASE* because *The Rest* is a subset of *Real*. The only observable behaviour of *LORRY_DATABASE* is over the traces drawn from $The\ rest^*$ and $Real^*$. This means that *LORRY_DATABASE* must be shown to satisfy $\mathcal{SR}_2$ because it legislates over *The rest* and *Real*. $\mathcal{SR}_6$ can be ignored because if *MOVE_QUEUE* satisfies $\mathcal{SR}_6$ then it will satisfy $\mathcal{SR}_6 \wedge_{\mathrm{IF}} \mathcal{SR}_2$. To show that *LORRY_DATABASE* satisfies the second security requirement it must be shown that

$$LORRY\_DATABASE \upharpoonright Real \cup Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

To show this, observations over *Real* and *The rest* need to be considered. Using the abbreviation $RO = Real \cup Order$

$$LORRY\_DATABASE \upharpoonright RO = LORRY\_DATABASE$$

since

$$\alpha LORRY\_DATABASE \subset The\ rest \subset Real$$

By theorem 7 in chapter 5, *LORRY_DATABASE* satisfies any confidentiality specification over *The rest* and *Real*, therefore by corollary 9

$$LORRY\_DATABASE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

LORRY_QUEUE

$$Move\_Order \triangleq \mathbb{P}(Kind \times \mathbb{N}) \times Location \times Lorry\_Id$$

A move order from *Movement-cell*is for n boots of a given kind to be transported to the given location using the lorry with the given identification.

$$Lorry\_Order \triangleq \mathbb{P}(Kind \times \mathbb{N}) \times Location \times Lorry\_Id$$

The orders received at *Transport* are of type *Lorry_Order*. A re-labelling function is needed to turn submitted orders from *Movement-cell* to received orders at *Transport*.

$$L : Move\_Order \rightarrow Lorry\_Order$$

The queue of orders from *Movement-cell* to *Transport* has the same basic design as the other queues. For a fixed number $n : \mathbb{N}$ the buffer of orders from *Movement-cell* to *Transport* is:

$$LORRY\_QUEUE \triangleq LORRY\_QUEUE_{\langle\rangle}$$

$$LORRY\_QUEUE_{\langle\rangle} \triangleq mo : Move\_Order \rightarrow LORRY\_QUEUE_{\langle mo\rangle}$$

$$LORRY\_QUEUE_{s^\frown\langle mo\rangle} \triangleq \left(\begin{array}{l} mo' : Move\_Order \rightarrow LORRY\_QUEUE_{\langle mo'\rangle^\frown s^\frown\langle mo\rangle} \\ [] \\ L(mo) \rightarrow LORRY\_QUEUE_s \end{array} \right)$$

such that $\#s < n - 1$

$$LORRY\_QUEUE_{s^\frown\langle mo\rangle} \triangleq mo \rightarrow LORRY\_QUEUE_s$$

such that $\#s = n - 1$

The intersection of the alphabet of *LORRY_QUEUE* with *The rest* is the alphabet of *LORRY_QUEUE*, since *Movement-cell* and *Transport* are both a subset of *The rest* which means that

$$\alpha LORRY\_QUEUE \subset The\ rest$$

The intersection of the alphabet of *LORRY_QUEUE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *LORRY_QUEUE* with *Real* is again the alphabet of *LORRY_QUEUE* because *The Rest* is a subset of *Real*. The only observable behaviour of *LORRY_QUEUE* is over the traces drawn from *The rest** and *Real**. This means that *LORRY_QUEUE* must be shown to satisfy $\mathcal{SR}_2$. To show that *LORRY_QUEUE* satisfies the second security requirement it must be shown that

$$LORRY\_QUEUE \upharpoonright Real \cup Order \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

To show this, observations over *Real* and *The rest* need to be considered. Using the abbreviation $RO = Real \cup Order$

$$LORRY\_QUEUE \upharpoonright RO = LORRY\_QUEUE$$

since

$$\alpha LORRY\_QUEUE \subset The\ rest \subset Real$$

By theorem 7 in chapter 5, *LORRY_QUEUE* satisfies any confidentiality specification over *The rest* and *Real*, therefore by corollary 9

$$LORRY\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_2$$

AUDIT_PROC

In this part an auditing component of the system is shown to satisfy the third security policy statement which has been formalized as a safety property. If the auditing component satisfies the safety specification then we know that the system as a whole will respect the third security policy statement. The reason for this is the CSP law for satisfaction

$$P \ \textbf{sat}\ S(tr) \wedge Q \ \textbf{sat}\ T(tr) \Longrightarrow P \| Q \ \textbf{sat}\ S(tr \upharpoonright \alpha P) \wedge T(tr \upharpoonright \alpha Q)$$

guarantees that every relevant trace will have the property described by the predicate $AUDIT\_INTEGRITY$.

Boots component $AUDIT\_PROC$

$$AUDIT\_PROC \triangleq AUDIT\_PROC_{\langle\rangle}$$

$$AUDIT\_PROC_s \triangleq \left(\begin{array}{l} (archive.s : Audits \rightarrow trail.(sanitize(s)) \rightarrow AUDIT\_PROC_{\langle\rangle}) \\ [] \\ (b : Auditable \rightarrow AUDIT\_PROC_{s^\frown\langle b\rangle}) \end{array} \right)$$

Theorem 14

$$AUDIT\_PROC \ \textbf{sat}\ AUDIT\_INTEGRITY$$

Proof The proof is by induction over the length of traces of $AUDIT\_PROC$ restricted to *Audits*. BASE CASE: $(\#(tr \upharpoonright Audits = 1)$

In this case for $s : Auditable^*$

$$tr \upharpoonright Auditable \cup Audits = s^\frown\langle archive.s\rangle$$

which can be shown by induction on the second limb of the definition of $AUDIT\_PROC_s$.

By definition of $\upharpoonright$

$$\begin{array}{rcl} flatten(s^\frown\langle archive.s\rangle \upharpoonright Audits) & = & flatten(\langle archive.s\rangle) \\ & = & s \\ & = & s^\frown\langle archive.s\rangle \upharpoonright Auditable \end{array}$$

STEP CASE: $((\#tr \upharpoonright Audits = n) \Longrightarrow flatten(tr \upharpoonright Audits) = tr \upharpoonright Auditable)$

As in the base case we have that for $s : Auditable^*$, $t : (Auditable \cup Audits)^*$ and $\#(tr^\frown tr' \upharpoonright Audits) = n + 1$

$$tr^\frown tr' \upharpoonright Auditable \cup Audits = tr^\frown s^\frown \langle archive.s \rangle \upharpoonright Auditable \cup Audits$$

which can be shown by induction on the second limb of the definition of $AUDIT\_PROC_s$.

By definition of $\upharpoonright$

$$\begin{array}{l} flatten(tr^\frown s^\frown \langle archive.s \rangle \upharpoonright Audits) \\ = flatten(tr \upharpoonright Audits)^\frown flatten(s^\frown \langle archive.s \rangle) \\ = tr \upharpoonright Auditable^\frown flatten(s^\frown \langle archive.s \rangle \upharpoonright Audits) \\ [\text{By induction hypothesis}] \\ = tr \upharpoonright Auditable^\frown flatten(\langle archive.s \rangle) \\ [\text{Since } s : Auditable^*] \\ = tr \upharpoonright Auditable^\frown s \\ [\text{By definition of } flatten] \\ = tr^\frown s^\frown \langle archive.s \rangle \upharpoonright Auditable \\ [\text{Since } \langle archive.s \rangle \notin Auditable] \end{array}$$

This shows that for any trace in $AUDIT\_PROC$ of the form $tr^\frown \langle archive.s \rangle$ then

$$flatten(tr^\frown \langle archive.s \rangle \upharpoonright Audits) = tr^\frown \langle archive.s \rangle \upharpoonright Auditable$$

which is $AUDIT\_INTEGRITY$. □

$ARCHIVE$

In this part the component $ARCHIVE$ is defined and shown to satisfy the security policy statement concerning what an operator performing an archive is allowed to know.

$$\alpha ARCHIVE = Operator \cup Audits$$

The archiving process initiated by an operator is defined as follows:

$$ARCHIVE \triangleq (ready \rightarrow mounted \rightarrow archive.s : Audits \rightarrow complete \rightarrow ARCHIVE)$$

The intersection of the alphabet of $ARCHIVE$ with *The rest* is *Operator*. The intersection of the alphabet of $ARCHIVE$ with the other windows in *Boot Windows* is the empty set, except for *Operator* and *Real*. The intersection of the alphabet of $ARCHIVE$ with *Operator* is obviously *Operator*. The intersection of the alphabet of $ARCHIVE$ with *Real* is again *Operator* because *The Rest* is a subset of *Real*. The only observable behaviour of $ARCHIVE$ is over the traces drawn from *The rest**,

Operator* and *Real**. This means that *ARCHIVE* must be shown to satisfy $\mathcal{SR}_2$ and $\mathcal{SR}_4$, since $\mathcal{SR}_2$ does not allow any inference over *Operator*. To show that *ARCHIVE* satisfies the fourth security requirement, if the window *Operator* is ignored, it must be shown that

$$ARCHIVE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_4$$

To show this, observations over *Operator* and *The rest* need to be considered.

The rest

$$ARCHIVE \upharpoonright (The\ rest \cup Audit) = ARCHIVE$$

Comparing *ARCHIVE* with $K\_SR4_{The\ rest}$ it can be shown that

$$K\_SR4_{The\ rest} \upharpoonright (Operator \cup Audit) = ARCHIVE$$

using the properties of restriction.

Operator

$$ARCHIVE \upharpoonright (Operator \cup Audit) = ARCHIVE$$

To show satisfaction of $\mathcal{SR}_4$ all that needs to be shown is that $K\_SR4_{Operator}$ has an arrow to *ARCHIVE*. Recall that in section A.2 $K\_SR4_{Operator}$ was defined as

$$K\_SR4_{Operator} \triangleq ready \rightarrow mounted \rightarrow a : Audits \rightarrow complete \rightarrow K\_SR4_{Operator}$$

which is the same as *ARCHIVE*, therefore the identity arrow goes from $K\_SR4_{Operator}$ to *ARCHIVE*. The arrows from K_SR4_{Real} and $K\_SR4_{Operator}$ to *ARCHIVE* mean that by corollary 9

$$ARCHIVE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_4$$

if the window *Real* is ignored. To show true satisfaction it is necessary to consider *ARCHIVE* with respect to $\mathcal{SR}_6$, then it will be the case that

$$ARCHIVE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_4 \wedge_{\text{IF}} \mathcal{SR}_6$$

RE-GRADE_QUEUE

In this part the component *RE-GRADE_QUEUE* is defined and shown to satisfy the security policy statement concerning what the security officer is allowed to know. The queue of requests, for re-grading an order, from *HQ* to *Security officer* has the same basic design as the other queues. For a fixed number $n : \mathbb{N}$ the buffer of requests from *Stock-cell* to *Security officer* is:

$$RE\text{-}GRADE\_QUEUE \triangleq RE\text{-}GRADE\_QUEUE_{\langle\rangle}$$

$$RE\text{-}GRADE\_QUEUE_{\langle\rangle} \triangleq$$

$$(security, o) : \{security\} \times Order \rightarrow RE\text{-}GRADE\_QUEUE_{\langle o\rangle}$$

$$RE\text{-}GRADE\_QUEUE_{s^\frown\langle o\rangle} \triangleq$$

$$\left(\begin{array}{l} (security, o') : \{security\} \times Order \rightarrow RE\text{-}GRADE\_QUEUE_{\langle o'\rangle^\frown s^\frown\langle o\rangle} \\ [\!] \\ (re\text{-}grade, o) \rightarrow RE\text{-}GRADE\_QUEUE_{s} \end{array} \right)$$

such that $\#s < n - 1$

$$RE\text{-}GRADE\_QUEUE_{s^\frown\langle o\rangle} \triangleq (re\text{-}grade, o) \rightarrow RE\text{-}GRADE\_QUEUE_{s}$$

such that $\#s = n - 1$

The intersection of the alphabet of *RE-GRADE_QUEUE* with *Security officer* is $\{re - grade\} \times Order$. The intersection of the alphabet of *RE-GRADE_QUEUE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *RE-GRADE_QUEUE* with *Real* is again $\{re-grade\}\times Order$ because *Security officer* is a subset of *Real*. The only observable behaviour of *RE-GRADE_QUEUE* is over the traces drawn from *Security officer* and $Real^*$. This means that *RE-GRADE_QUEUE* must be shown to satisfy $\mathcal{SR}_5$ and $\mathcal{SR}_6$, since $\mathcal{SR}_5$ does not allow any inference over *Real*. To show that *RE-GRADE_QUEUE* satisfies the fifth security requirement, if the window *Real* is ignored, it must be shown that

$$RE\text{-}GRADE\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

To show this, only observations over *Security officer* need to be considered.

$$RE\text{-}GRADE\_QUEUE \upharpoonright (Security\ officer \cup Auditable) =$$
$$(re\text{-}grade, o) : \{re\text{-}grade\} \times Order \rightarrow$$
$$RE\text{-}GRADE\_QUEUE \upharpoonright (Security\ officer \cup Auditable)$$

Comparing *RE-GRADE_QUEUE* with $K\_SR5_{Security\ officer}$ it can be shown that

$$K\_SR5_{Security\ officer} \upharpoonright (\{re - grade\} \times Order) =$$
$$RE\text{-}GRADE\_QUEUE \upharpoonright (Security\ officer \cup Auditable)$$

using the properties of restriction. The arrows from $K\_SR5_{Security\ officer}$ to *RE-GRADE_QUEUE* means that by corollary 9

$$RE\text{-}GRADE\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

if the window *Real* is ignored. To show true satisfaction it is necessary to consider *RE-GRADE_QUEUE* with respect to $\mathcal{SR}_6$, then it will be the case that

$$RE\text{-}GRADE\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5 \wedge_{\text{IF}} \mathcal{SR}_6$$

GRADED_QUEUE

In this part the component *GRADED_QUEUE* is defined and shown to satisfy the security policy statement concerning what the security officer is allowed to know. The queue of orders which have been graded by the *Security officer*, from *Security officer* to *Stock-cell* has the same basic design as the other queues. For a fixed number $n : \mathbb{N}$ the buffer of requests from *Security officer* to *Stock-cell* is:

$$GRADED\_QUEUE \triangleq GRADED\_QUEUE_{\langle\rangle}$$

$$GRADED\_QUEUE_{\langle\rangle} \triangleq$$

$$(graded, o) : \{graded\} \times Order \rightarrow GRADED\_QUEUE_{\langle o\rangle}$$

$$GRADED\_QUEUE_{s^\frown\langle o\rangle} \triangleq$$

$$\left(\begin{array}{l} (graded, o') : \{graded\} \times Order \rightarrow GRADED\_QUEUE_{\langle o'\rangle^\frown s^\frown\langle o\rangle} \\ [\!] \\ (sec\_grade, o) \rightarrow GRADED\_QUEUE_s \end{array}\right)$$

such that $\#s < n - 1$

$$GRADED\_QUEUE_{s^\frown\langle o\rangle} \triangleq (sec\_grade, o) \rightarrow GRADED\_QUEUE_s$$

such that $\#s = n - 1$

The intersection of the alphabet of *GRADED_QUEUE* with *Security officer* is $\{graded\} \times Order$. The intersection of the alphabet of *GRADED_QUEUE* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *GRADED_QUEUE* with *Real* is again $\{graded\} \times Order$ because *Security officer* is a subset of *Real*. The only observable behaviour of *GRADED_QUEUE* is over the traces drawn from *Security officer* and $Real^*$. This means that *GRADED_QUEUE* must be shown to satisfy $\mathcal{SR}_5$ and $\mathcal{SR}_6$, since $\mathcal{SR}_5$

does not allow any inference over *Real*. To show that *GRADED_QUEUE* satisfies the fifth security requirement, if the window *Real* is ignored, it must be shown that

$$GRADED\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

To show this, only observations over *Security officer*need to be considered.

$$GRADED\_QUEUE \upharpoonright (Security\ officer \cup Auditable) =$$
$$(graded, o) : \{graded\} \times Order \rightarrow GRADED\_QUEUE \upharpoonright (Security\ officer \cup Auditable)$$

Comparing *GRADED_QUEUE* with $K\_SR5_{Security\ officer}$ it can be shown that

$$K\_SR5_{Security\ officer} \upharpoonright (\{graded\} \times Order) =$$
$$GRADED\_QUEUE \upharpoonright (Security\ officer \cup Auditable)$$

using the properties of restriction. The arrows from $K\_SR5_{Security\ officer}$ to *GRADED_QUEUE* mean that by corollary 9

$$GRADED\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

if the window *Real* is ignored. To show true satisfaction it is necessary to consider *GRADED_QUEUE* with respect to $\mathcal{SR}_6$, then it will be the case that

$$GRADED\_QUEUE \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5 \wedge_{\text{IF}} \mathcal{SR}_6$$

AUDIT_PROC again!

The auditing component *AUDIT_PROC* was shown to satisfy the specification that guaranteed that the audit trail was not corrupted. It was not shown to satisfy any of the confidentiality specifications. To add *AUDIT_PROC* to the system it must be shown to satisfy some confidentiality specification, or else it could violate the confidentiality requirement which has been established for the other components.

The intersection of the alphabet of *AUDIT_PROC* with *Security officer* is $\{ trail.t \mid s : Auditable \wedge sanitize(a) = t \}$. The intersection of the alphabet of *AUDIT_PROC* with the other windows in *Boot Windows* is the empty set, except for *Real*. The intersection of the alphabet of *AUDIT_PROC* with *Real* is again $\{ trail.t \mid s : Auditable \wedge sanitize(a) = t \}$ because *Security officer* is a subset of *Real*. The only observable behaviour of *AUDIT_PROC* is over the traces drawn from *Security officer* and $Real^*$. This means that *AUDIT_PROC* must be shown to satisfy $\mathcal{SR}_5$ and $\mathcal{SR}_6$, since $\mathcal{SR}_5$ does not allow any inference over *Real*. To show that *AUDIT_PROC* satisfies the fifth security requirement, if the window *Real* is ignored, it must be shown that

$$AUDIT\_PROC \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

To show this, only observations over *Security officer* need to be considered.

$$AUDIT\_PROC \upharpoonright (Security\ officer \cup Auditable) =$$
$$AUDIT\_PROC_{\langle\rangle} \upharpoonright (Security\ officer \cup Auditable)$$

$$AUDIT\_PROC_s \upharpoonright (Security\ officer \cup Auditable) =$$

$$\left(\begin{array}{l} trail.(sanitize(s)) \rightarrow AUDIT\_PROC_{\langle\rangle} \upharpoonright (Security\ officer \cup Auditable) \\ [\!] \\ (a : Auditable \rightarrow AUDIT\_PROC_{s^\frown\langle a\rangle}) \upharpoonright (Security\ officer \cup Auditable) \end{array} \right)$$

Comparing $AUDIT\_PROC$ with $K\_SR5_{Security\ officer}$ it can be shown that

$$K\_SR5_{Security\ officer} \upharpoonright \{ trail.t \mid s : Auditable \wedge sanitize(a) = t \}$$
$$=$$
$$AUDIT\_PROC \upharpoonright (Security\ officer \cup Auditable)$$

using the properties of restriction. The arrows from $K\_SR5_{Security\ officer}$ to $AUDIT\_PROC$ mean that by corollary 9

$$AUDIT\_PROC \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5$$

if the window *Real* is ignored. To show true satisfaction it is necessary to consider $AUDIT\_PROC$ with respect to $\mathcal{SR}_6$, then it will be the case that

$$AUDIT\_PROC \ \textbf{s-sat}^{\text{Boot Windows}} \ \mathcal{SR}_5 \wedge_{\text{IF}} \mathcal{SR}_6$$

The exercise components

The exercise components of the Boots CCIS are identical to the following components already encountered:

$ORDER\_QUEUE$, $INVOICE\_QUEUE$, $MOVE\_QUEUE$, $LORRY\_QUEUE$ and $BOOT\_DATABASE$.

The exercise component counterparts are defined by re-labelling the components listed above. We define five re-labelling functions, one for each component. Each re-labelling function adds the label ex to each event in the alphabet.

$$R_{OQ} \triangleq (\lambda y : \alpha ORDER\_QUEUE \cdot (y, ex))$$

$$R_{IQ} \triangleq (\lambda y : \alpha INVOICE\_QUEUE \cdot (y, ex))$$

$$R_{MQ} \triangleq (\lambda y : \alpha MOVE\_QUEUE \cdot (y, ex))$$

$$R_{LQ} \triangleq (\lambda y : \alpha LORRY\_QUEUE \cdot (y, ex))$$

$$R_{BD} \triangleq (\lambda y : \alpha BOOT\_DATABASE \cdot (y, ex))$$

$$ORDER\_QUEUE\_EX \triangleq R_{OQ}(ORDER\_QUEUE)$$

$$INVOICE\_QUEUE\_EX \triangleq R_{IQ}(INVOICE\_QUEUE)$$

$$MOVE\_QUEUE\_EX \triangleq R_{MQ}(MOVE\_QUEUE)$$

$$LORRY\_QUEUE\_EX \triangleq R_{LQ}(LORRY\_QUEUE)$$

$$BOOT\_DATABASE\_EX \triangleq R_{BD}(BOOT\_DATABASE)$$

The window *Boots (exercise)* was defined earlier in this appendix to be equal to

$$Exercise \cup Stock\text{-}cell\ (exercise) \cup Movement\text{-}cell\ (exercise)$$

this is the complete set of events concerned with exercise operations, consequently the intersection of the alphabet of any of the exercise components with *Boots (exercise)* is the alphabet of that component. For example

$$\alpha BOOT\_DATABASE\_EX \cap Boots\ (exercise) = \alpha BOOT\_DATABASE\_EX$$

The intersection of the alphabet of any of the exercise components with the other windows in *Boot Windows* is the empty set. The only observable behaviour of the exercise components is over the traces drawn from *Boots (exercise)*. By theorem 7 in chapter 5, all of the exercise components will satisfy any confidentiality specification over *Boots (exercise)*.

The real components

The components of Boots which are involved with the real world operation of the CCIS can be composed in parallel to give:

$$REAL\_PROC \triangleq \begin{array}{l} MOVE\_QUEUE \| INVOICE\_QUEUE \| BOOT\_DATABASE \| \\ LORRY\_QUEUE \| ARCHIVE \| RE\text{-}GRADE\_QUEUE \| \\ GRADED\_QUEUE \| AUDIT\_PROC \end{array}$$

The alphabet of *REAL_PROC* intersects with all the windows in *Boot Windows* except for *Boots (exercise)*. The components which are observable at these windows have already been dealt with except for the window *Real*.

$$\alpha(REAL\_PROC) \cap Boots\ (exercise) = \{\}$$

This means that exercise information does not affect the real operation of Boots. This is formally shown by observing that through *Real* every behaviour of

$$REAL\_PROC \upharpoonright (Real \cup Boots\ (exercise))$$

is visible, hence by theorem 7 in chapter 5,

$$REAL\_PROC \upharpoonright (Real \cup Boots\ (exercise))$$

will satisfy any confidentiality specification over *Real*. In particular

$$REAL\_PROC \upharpoonright (Real \cup Boots\ (exercise))\ \textbf{s-sat}^{\text{Boot Windows}}\ \mathcal{SR}_6$$

Bibliography

[Arib & Manes 74]
Michael Arib and Ernest Manes, **Machines in a category: an expository introduction**. SIAM Review, Vol. 16, No. 2, April 1974.

[Barr & Wells 90]
Michael Barr and Charles Wells, **Category theory for computing science**, Prentice Hall 1990.

[Bell73a]
D.E. Bell and L.J. LaPadula, **Secure Computer Systems: Mathematical Foundations**, MTR-2547, Vol. 1, MITRE Corp., November 1973

[Bell73b]
D.E. Bell and L.J. LaPadula, **Secure Computer Systems: A Mathematical Model**, MTR-2547, Vol. 2, MITRE Corp., November 1973

[Bell74]
D.E. Bell and L.J. LaPadula, **Secure Computer Systems: A Refinement of the Mathematical Model**, MTR-2547, Vol. 3, MITRE Corp., November 1974

[Brewer & Nash 89]
David Brewer and Michael Nash, **The Chinese Wall Security Policy**, Proceedings 1989 IEEE Symposium on Security and Privacy, Oakland.

[Brookes 83]
S.D. Brookes, **A model for communicating sequential processes. Oxford University, D.Phil. Thesis 1983**.

[Brookes et al]
S.D. Brookes, C.A.R. Hoare, A.W. Roscoe, **A theory of communicating sequential processes. JACM 31, 1984. pg 560 - 599**.

[Brookes & Roscoe]
S.D. Brookes and A.W. Roscoe, **An improved failures model for communicating processes**. Proceedings of the Pittsburgh Seminar on concurrency, Springer LNCS 197, 1985.

[CRAMM]
CRAMM User Guide, CCTA.

[G-C & Sanders 91]
J. Graham-Cumming and J.W. Sanders, **On the refinement of Non-Interference**, Proceedings 1991 IEE Computer Security Foundations Workshop, IV. Franconia

[Cuppens 90]
F. Cuppens, **An Epistemic and Deontic Logic for reasoning about computer security**, Proceedings of the European symposium on research in computer security, Toulouse, France, October 1990.

[Davies 91]
Jim Davies, **Specification and Proof in Real-time Systems**, DPhil. thesis, Oxford University, 1991.

[Davies & Schneider 89]
Jim Davies & Steve Schneider, **An Introduction to Timed CSP**, Technical Monograph PRG-75, Oxford University Computing Laboratory, Programming Research Group, 11 Keble Road, Oxford OX1 3QD. ISBN 0-902928-57-0, 1989.

[Davies et al 90]
Jim Davies, D. Jackson and Steve Schneider, **Making things happen in Timed CSP**, Oxford University Programming Research Group Technical Report, TR-2-90, 1990.

[Dobson & McDermid 89]
J.E. Dobson and J.A. McDermid, **Security Models and Enterprise Models**, Database Security: Status and Prospects II, Ed. C.E. Landwehr, Elsevier Science Publishers, Amsterdam, 1989.

[Eizenberg 89]
G. Eizenberg, **Mandatory policy: A secure system model**. In AFCET, editor, European Workshop on Computer Security, Paris 1989.

[Feiertag 80]
R. J. Feiertag, **A Technique for Proving Specifications are Multilevel secure**, Technical report CSL-109, SRI Computer Laboratory, Menlo Park, CA., January 1980.

[Foley 88]
S. Foley, **A Theory and model for secure information flow**, Ph.D Thesis 1988. Department of computer science, University College, Cork.

[Foster 89]
J.M. Foster, **The Algebraic Specification of a Target Machine: Ten15**, from High Integrity Software, C.T. Sennett (Ed.), Pitman Press 1989.

[Gray 90]
J.W. Gray, **Probabilistic Interference**, Proceedings 1990 IEEE Symposium on Security and Privacy, Oakland.

[Gray 91]
J.W. Gray, **Toward a Mathematical Foundation for Information Flow Security**, Proceedings 1991 IEEE Symposium on Security and Privacy, Oakland.

[Glasgow & McEwen 88]
J. Glasgow and G. McEwen, **Reasoning about knowledge and permission in secure distributed systems.** Proceedings of the computer security foundations workshop Franconia 1988.

[Goguen 72]
J.A. Goguen, **Minimal Realization of machines in closed categories.** Bulletin of the American mathematical society. Vol 78, No. 5, September 1972.

[Goguen 73a]
J.A. Goguen, **Realization is Universal.** Mathematical System Theory. Vol 6, pp 359-374, 1973.

[Goguen 73b]
J.A. Goguen, **Categorical foundations for general systems theory.** In F. Pichler and R. Trappl, editors, Advances in Cybernetics and Systems Research, pages 121-130. Transcripta Books, 1973.

[Goguen & Meseguer 82]
J.A. Goguen and J. Meseguer, **Security policies and security models**, Proceedings 1982 IEEE Symposium on Security and Privacy, Oakland.

[Goguen & Meseguer 84]
J.A. Goguen and J. Meseguer, **Unwinding and inference control**, Proceedings 1984 IEEE Symposium on Security and Privacy, Oakland.

[Goguen 89]
J.A Goguen, **A Categorical Manifesto**, Technical monograph PRG-72, Oxford University Computing Laboratory, March 1989

[Goguen 92]
J.A Goguen, **Sheaf Semantics for Concurrent Interacting Objects**, Mathematical Structures in Computer Science, 1992.

[Goldblatt 84]
R. Goldblatt, **Topoi, the categorial analysis of logic**. North Holland 1984.

[Hoare 80]
C.A.R. Hoare, **A model for communicating sequential processes.** On the construction of programs. C.U.P. 1980. pg 229 - 248.

[Hoare et al 81]
C.A.R. Hoare, S.D. Brookes and A.W. Roscoe. **A theory of communicating sequential processes**. Oxford University Computing Laboratory technical monograph PRG-16, 1981.

[Hoare 85]
C.A.R. Hoare, **Communicating sequential processes**, Prentice Hall 1985.

[Hoare 88]
C.A.R Hoare, **An approach to category theory for computer scientists**. Private communication.

[Jackson 90]
D. Jackson, **Specifying Timed Communicating Sequential Processes in Temporal Logic**, Oxford University Programming Research Group Technical Report, TR-5-90, 1990.

[Jacob 88a]
J.L Jacob, **Security specifications**, Proceedings 1988 IEEE Symposium on Security and Privacy, Oakland.

[Jacob 88b]
J.L Jacob, **A security framework**, Proceedings of the computer security foundations workshop, June 12-15 1988, Franconia, New Hampshire, Sponsored by the IEEE computer society.

[Jacob 89]
J. Jacob, **On the derivation of secure sub-components**, Proceedings 1989 IEEE Symposium on Security and Privacy, Oakland.

[Jacob 89b]
J.L Jacob, **Security refinement is not Ordinary refinement**, Proceedings 1989 Workshop in Refinement, Open University, Milton Keynes.

[Jacob 90a]
Specifying Security Properties, in C. A. R. Hoare, editor. Developments in Concurrency and Communication, (the proceedings of the Year of Programming Institute in Concurrent Programming), Addison Wesley, 1990

[Jacob 90b]
J.L Jacob, **Categorising non-interference**, Proceedings of the computer security foundations workshop, June 12-15 1990, Franconia, New Hampshire, Sponsored by the IEEE computer society.

[Halpern & Moses 90]
J. Y. Halpern and Y. Moses, **Knowledge and common knowledge in a distributed environment**. J.A.C.M Vol. 37, No. 3, July 1990

[He 89]
He Jifeng, Private communication 1989

[Lambek & Scott 86]
J. Lambek and P.J. Scott, **Introduction to higher categorical logic**, Cambridge University Press 1986.

[Mac Lane]
S. Mac Lane, **Categories for the Working Mathematician**, Graduate texts in mathematics, Springer-Verlag

[McLean 87]
J. McLean, **Reasoning About Security Models**, Proceedings 1987 IEEE Symposium on Security and Privacy, Oakland, CA. IEEE Computer Society, April 1987

[McCullough 87]
D. McCullough, **Noninterference and the Composability of Security Properties**, Proceedings 1987 IEEE Symposium on Security and Privacy, Oakland, CA. pp 177 - 186, IEEE Computer Society, April 1987

[Meadows 92]
C. Meadows, **Using Traces of Procedure Calls to Reason About Compositionality**, Proceedings 1992 IEEE Symposium on Security and Privacy, Oakland.

[Mess. & Mont. 88] J. Meseguer & U. Montanari, **Petri Nets Are Monoids: A New Algebraic Foundation for Net Theory**. Proceedings of the IEEE Symposium On Logic In Computer Science 1988.

[Morgan 90]
C. Morgan, **Programming from Specifications**, Prentice Hall International, 1990.

[Olderog & Hoare 86]
E.-R. Olderog and C.A.R. Hoare, **Specification-Oriented Semantics for Communicating Processes**. Acta Informactica 23, 1986.

[Reed & Roscoe 86]
George M. Reed and A.W. Roscoe, **A timed model for communicating sequential processes**. Proceedings of ICALP'86. Springer LNCS 226, 1986. pg 314-323.

[Reed & Roscoe 88]
George M. Reed and A.W. Roscoe, **Metric spaces as models for real-time concurrency**, Proceedings of the third workshop onthe mathematical foundations of programming language semantics. Springer LNCS 1988.

[Reed 88]
George M. Reed, **A uniform mathematical theory for real-time distributed computing**. DPhil Thesis, Oxford University

[Reisig 85]
W. Reisig, **Petri Nets**, Springer Verlag, 1985.

[Roscoe 82]
A.W. Roscoe, **A mathematical theory of communicating sequential processes**. Oxford University D.Phil thesis 1982.

[Rushby 82]
J.M. Rushby, **Proof of Separability**, 5^{th} International Symposium on Programming, Turin, Italy 1982, Springer-Verlag LNCS No. 137.

[Ryan 90]
P. Ryan, **A CSP formulation of non-interference and unwinding**. Proceedings of the computer security foundations workshop, June 12-15 1990, Franconia, New Hampshire, Sponsored by the IEEE computer society.

[Sanders & He 85]
A predicate model for CSP, Private communication

[Schneider 89]
S.A. Schneider, **Correctness and Communication in Real-time Systems**, DPhil. thesis, Oxford University, 1989.

[Spivey 88]
J. M. Spivey, **Understanding Z**, Cambridge University Press

[Sutherland 86]
D. Sutherland, **A model of information**, Proc. 9^{th} National Computer Security conference, 1986, U.S. National Computer Security Center & U.S. National Bureau of Standards

[Terry 89a]
P. Terry and S. Wiseman, **A "New" Security Policy Model**, Proceedings 1989 IEEE Symposium on Security and Privacy, Oakland, CA. IEEE Computer Society, April 1989

[Terry 89b]
P. Terry, **The SMITE approach to security**, RSRE Malvern, Report No. 89014

[Walters 89]
R. F. C. Walters, **An imperative language based on distributive categories**, Research Report 89-26, Dept of Pure Mathematics, The university of Sydney, Australia.