Aggregation and Separation as Noninterference Properties

Simon N. Foley*

March 3, 1992

Abstract

This paper proposes a notation that can be used to describe information flow policies that may have transitivity, aggregation and separation (of duty) exceptions. Operators for comparing, composing and abstracting these policies are described. These allow complex policies to be built from simpler policies. A formal semantics is given based on the notion of noninterference for deterministic systems. An unwinding of this definition is developed that can be used for any policy that does not contain a separation exception.

1 Introduction

Confidentiality security is concerned with restricting the disclosure of information in systems. One way of achieving this is to use an information flow policy, which defines the different classes of information (for example, classified, secret, etc.) that can exist in the system, and a flow relation, which describes how information may flow between these classes. Traditionally, information flow policies have formed lattices[7], with good reason, as it is relatively clear how to build state-based enforcement mechanisms. However, it has been shown in [12, 14] that it is desirable to enforce non-transitive information flow policies. That is, a flow policy where information is allowed to flow from class a to class b, and from b to class c, but not from a to c. For example, we may wish to specify that top-secret information may flow to class brigadier (representing the class of information a brigadier

*Address: Computer Science Department, University College, Cork, Ireland. Electronic Mail: simon@security.ucc.ie.

may handle), and brigadier information may flow to class secret, but top-secret information may not flow to secret. Examples of such policies will be considered in this paper.

Another desirable generalization for flow policies concerns the consistency of the class join operator (the least upper bound). Traditionally, if information may flow from a to c and from b to c, then the join of information of class a and b may flow to c. If this assumption is removed, we have a means of describing aggregation flow exceptions: a user at class c is allowed to sink information at class a or class b, but not their aggregate. That real aggregation policies exist (as opposed to unnecessary aggregation policies[18]), is demonstrated by Brewer and Nash in their Chinese Wall paper[4], and supported with further examples by Meadows[21]. Many models for aggregation policies have been proposed[4, 17, 20, 14]. We seek a unified definition of an information flow policy that includes aggregation properties.

Separation of Duty rules are normally associated with integrity policies and models[6]. For example, a check must be proposed by a manager, and cleared by an accountant, or vice-versa, before it can be written. However, separation rules can also apply to information flow (confidentiality) policies. For example, a user of a dial-up database may only view database information if it is accompanied by charging information. This type of separation flow policy can be thought of as the dual of an aggregation policy: only the aggregate may flow, not the individual classes.

In [14, 15] state transition models are described that enforce a large class of these information flow policies. Security is based on an informal notion of information flow under a MAC style regime. This paper consolidates these models by providing a formal semantics for their information flow policies.

Section 2 describes the notation for describing information flow policies that may contain transitive, aggregation and separation exceptions. Relations and operators over flow policies are developed that allow complex flow policies to be compared, composed and abstracted. Section 3 gives a semantics for these information flow policies based on the notion of noninterference. The semantics is abstractly defined in terms of a traces model for deterministic systems. Section 4 considers how this definition can be unwound into conditions on states and state transitions. The unwinding is restrictive in that it does not consider policies with separation exceptions. The unwinding is not unlike that for traditional noninterference, but with some additional conditions.

Our model provides a basis for studying other information flow policies. Section 5 outlines how other models, in particular those that provide dynamic upgrading and partial trust, can be interpreted within our policy framework. Section 6 studies the specification of a complex information flow policy, a Chinese Wall policy[4, 17]. The interested reader should refer to [13, 14] for further examples.

Section 2 is a revised version of material that has been published in [14]. The necessary proofs for Section 2 are given in [13]. We have chosen to use the Z notation[23] to provide a consistent syntax for presenting the mathematics in this paper. In using Z, it has been possible to syntax- and type-check the definitions using the fuzz tool. The appendix contains a short introduction to the Z notation used in this paper. We have avoided using many constructs that are peculiar to Z; much of the notation presented in this paper can be easily recognized from its set and logic origins.

2 Conglomerate Flow Policies

An information flow policy defines the different classes of information that can exist in a system and how information may flow between these classes. We let *class* be the set of all information classes. We are not further interested in its structure, except that it is non-empty. We introduce the basic type

$$[class]$$

Usually a simple relation of type $class \leftrightarrow class$ is used to describe a flow policy. With such a policy there is an implicit assumption that if (information at) class a may flow to class c and class b may flow to class c, then a and b may together flow to c. To remove this assumption it will be necessary to specify whether information formed from the combination of a and b may flow to c. Thus a flow policy shall be represented by a relation of type $(\mathbb{P}\, class) \leftrightarrow class$. Given such a flow policy P and $A : \mathbb{P}\, class$, $a : class$, then $A \mapsto a \in P$ (sometimes written as $A\ P\ a$) means that information at *conglomerate class* A may flow to class a. This is similar to stating that the 'join' (aggregate) of information at the classes in A may flow to class a.

The notion of a flow policy could be further generalized by representing the policy as a relation of type $(\mathbb{P}\, class) \leftrightarrow (\mathbb{P}\, class)$. This would facilitate policies with inconsistent greatest lower bounds. For example, class a may flow to classes b or c, but may not flow to both (i.e., may not flow to conglomerate $\{b, c\}$). Such a generalization is not considered in this paper; it is a topic for further research.

For the purposes of this paper, information flow policies are used to describe constraints on how information may flow in a system. The environment of a system can be viewed as a collection of *users*. These users need not be actual system users, but could correspond to interfaces to other subsystems. It is users that give a degree of meaning to the information that they sink and source, and therefore we associate an information class with each user. This 'binding' is static and gives the class of information that the user is considered to handle (source or sink). A user bound to class a is assumed to be 'aware' of the the information that he may handle, and therefore for consistency, we require that $\{a\} \mapsto a \in P$ hold in the system flow policy P. This requirement does not imply that it is impossible to build a system that prevents information flowing from a user to himself. Rather, given our view, it is futile to attempt to enforce such a policy, since a user is always aware of his own class of information. Similarly, if $a : \mathit{class};\ B : \mathbb{P}\,\mathit{class}$, and suppose the flow policy P includes $B \mapsto a \in P$, then since information at a class a may always flow to itself, then $a \in B$ should also hold: a user bound to class a may sink a conglomerate B of information, and in doing so, is really combining this with information he already knows, i.e., information of class a.

These constraints are reflected in the global constant *policy*, the set of all conglomerate flow policies

$$\begin{aligned} \mathit{policy} == \{\ & P : (\mathbb{P}\,\mathit{class}) \leftrightarrow \mathit{class} \mid \\ & \quad \forall\, a : \mathit{class};\ A : \mathbb{P}\,\mathit{class} \bullet \\ & \qquad A \mapsto a \in P \Rightarrow (a \in A) \wedge (A \subseteq \operatorname{ran} P)\ \} \end{aligned}$$

The constraint on *policy* ensures that information at any class (considered by the policy) may always flow to itself, regardless of the conglomerate it may be involved with. Note that, given the definition of *policy* we have $\operatorname{ran} P = \bigcup \operatorname{dom} P$

The *alphabet* αP of a flow policy P gives the set of classes it is defined in terms of. It is defined as

$$\begin{array}{|l} \alpha : \mathit{policy} \rightarrow \mathbb{P}\,\mathit{class} \\ \hline \forall\, P : \mathit{policy} \bullet \\ \qquad \alpha P = \operatorname{ran} P \end{array}$$

Example 1 A flow policy has two information classes `lo` and `hi`, with the usual interpretation that information may flow from `lo` to `hi`. This policy has alphabet $\{\mathtt{lo}, \mathtt{hi}\}$, and is described by the relation

$$\mathtt{HiLo} == \{\{\mathtt{lo}\} \mapsto \mathtt{lo}, \{\mathtt{hi}\} \mapsto \mathtt{hi}, \{\mathtt{lo}, \mathtt{hi}\} \mapsto \mathtt{hi}\}$$

Since `lo` information may flow to class `hi`, we have $\{\mathtt{lo}, \mathtt{hi}\} \mapsto \mathtt{hi} \in \mathtt{HiLo}$. $\triangle$

The least restrictive flow policy with alphabet C is defined by $\bot C$. This policy permits information to flow from every conglomerate class to every class. The most restrictive flow policy with alphabet C is defined by $\top C$. This policy does not permit any information flow between classes except reflexive flows $\{a\} \mapsto a \in (\top C)$, for $a \in C$.

$$\begin{array}{|l}
\top, \\
\bot : (\mathbb{P}\, \mathit{class}) \to \mathit{policy} \\
\hline
\forall\, C : \mathbb{P}\, \mathit{class} \bullet \\
\qquad \top C = \{\ a : C \bullet \{a\} \mapsto a\ \} \land \\
\qquad \bot C = \{\ A : \mathbb{P}\, C;\ a : C \bullet (A \cup \{a\}) \mapsto a\ \}
\end{array}$$

For example, given classes `lo` and `hi`, we have

$$\top\{\mathtt{lo}, \mathtt{hi}\} = \{\{\mathtt{lo}\} \mapsto \mathtt{lo}, \{\mathtt{hi}\} \mapsto \mathtt{hi}\}$$
$$\bot\{\mathtt{lo}, \mathtt{hi}\} = \{\{\mathtt{lo}\} \mapsto \mathtt{lo}, \{\mathtt{hi}\} \mapsto \mathtt{hi}, \{\mathtt{lo}, \mathtt{hi}\} \mapsto \mathtt{lo}, \{\mathtt{lo}, \mathtt{hi}\} \mapsto \mathtt{hi}\}$$

Specifying all possible maplets in a policy can be tedious. Useful infix operators for describing flow policies are

$$\begin{array}{|l}
_\ \mathrm{sepflow}\ _, \\
_\ \mathrm{aggflow}\ _ : ((\mathbb{P}\, \mathit{class}) \times \mathit{class}) \to \mathit{policy} \\
\hline
\forall\, A : \mathbb{P}\, \mathit{class};\ a : \mathit{class} \bullet \\
\qquad A\ \ \mathrm{sepflow}\ \ a = (\top(A \cup \{a\})) \cup \{(A \cup \{a\}) \mapsto a\} \land \\
\qquad A\ \ \mathrm{aggflow}\ \ a = \bigcup\{\ C : \mathbb{P}\, A \bullet C\ \ \mathrm{sepflow}\ \ a\ \}
\end{array}$$

Given $A : \mathbb{P}\, \mathit{class};\ a : \mathit{class}$, the policy $P = (A\ \mathrm{sepflow}\ a)$ permits information to flow from class conglomerate $(A \cup \{a\})$ to class a. Including $\top(A \cup \{a\})$ in policy P permits reflexive flows of the form $\{b\} \mapsto b \in P$, ensuring that P forms a valid policy. The policy $(A\ \mathrm{aggflow}\ a)$ permits information to flow from any aggregate of classes drawn from A to a. The policy from Example 1 can be described as $(\{\mathtt{lo}\}\ \mathrm{aggflow}\ \mathtt{hi})$. Note that for this example, $(\{\mathtt{lo}\}\ \mathrm{sepflow}\ \mathtt{hi})$ also describes the same policy. How these operators are used in practice is illustrated by the following examples.

Example 2 A traditional military-style flow policy can be defined as

$$\texttt{Military} == (\{\texttt{confidential}\} \text{ aggflow } \texttt{secret}) \cup (\{\texttt{confidential}, \texttt{secret}\} \text{ aggflow } \texttt{top-secret})$$

By using the aggflow operator, we can conveniently state that any conglomerate (aggregate) of `confidential`, `secret` or `top-secret` information may flow to `top-secret`. △

Example 3 A private hospital maintains the following kinds of information: patient accounting `acc`; patient treatment `treat`; medical reports `medrep`; financial reports `finrep`, and management information `mgr`. The flow policy is defined as

$$\begin{aligned}\texttt{HospPol} \quad == \quad & (\{\texttt{acc}, \texttt{treat}\} \text{ aggflow } \texttt{mgr}) \cup \\ & (\{\texttt{treat}, \texttt{mgr}\} \text{ aggflow } \texttt{medrep}) \cup \\ & (\{\texttt{acc}, \texttt{mgr}\} \text{ aggflow } \texttt{finrep})\end{aligned}$$

Management may be thought of as coordinating all information: they read patient accounts and treatments in order to write financial and medical reports. However note the non-transitivities in this policy: information may flow from `treat` to `mgr` and from `mgr` to `finrep`, but (for confidentiality reasons) `treat` may not flow to `finrep`. Similarly, `acc` may flow to `mgr` and `mgr` may flow to `medrep`, but (for profitability reasons) `acc` may not flow to `medrep`. △

An information flow policy contains an *aggregation exception* if flows from conglomerates A and B are permitted to flow to class c, but the aggregate $A \cup B$ may not flow to class c.

Example 4 A system maintains information about two different banks. Bank information is classified either as `bank1` or `bank2`. A consultant is allowed to read information on either bank, but not both (a simple Chinese Wall aggregation policy[4]). If `cons` represents the class of information a consultant is allowed to handle, then the policy can be specified as

$$\texttt{BankPol} == (\{\texttt{bank1}\} \text{ aggflow } \texttt{cons}) \cup (\{\texttt{bank2}\} \text{ aggflow } \texttt{cons})$$

Note how the conglomerate {`bank1,bank2`} may not flow to `cons`. △

A flow policy contains a *separation exception* if a conglomerate $A \cup B$ is allowed to flow to class c, but individually A or B may not.

Example 5 Users of a dial-up stock market database must be aware of service charges when they access stock information. This can be captured by the separation policy

$$\begin{aligned}\texttt{Stockpol} == {} & (\{\texttt{charges}\} \ \mathrm{sepflow}\ \texttt{user}) \cup \\ & (\{\texttt{charges}, \texttt{stock}\} \ \mathrm{sepflow}\ \texttt{user})\end{aligned}$$

A user is always allowed to sink charges information, but may only sink stock information as a conglomerate with charges information. $\triangle$

2.1 Classes of Flow Policy

The set of flow policies that do not contain separation exceptions is called the set of aggregation policies and is defined as

$$\begin{aligned}aggpol == \{\ & P : policy \mid (\forall A, B : \mathbb{P}\, class;\ a : class \bullet \\ & ((A \cup B) \mapsto a \in P) \Rightarrow \\ & \quad (((A \cup \{a\}) \mapsto a \in P) \wedge ((B \cup \{a\}) \mapsto a \in P)))\ \}\end{aligned}$$

The set of flow policies that do not contain aggregation exceptions is called the set of separation policies, and is defined as

$$\begin{aligned}seppol == \{\ & P : policy \mid (\forall A, B : \mathbb{P}\, class;\ a : class \bullet \\ & ((A \mapsto a \in P) \wedge (B \mapsto a \in P)) \Rightarrow (A \cup B) \mapsto a \in P)\ \}\end{aligned}$$

The set of flow policies that contain neither separation nor aggregation exceptions is

$$refpol == aggpol \cap seppol$$

That is, the set of reflexive flow policies. Recall that the reason for having the domain of a policy relation over $\mathbb{P}\, class$ is to allow the specification of aggregation and separation exceptions. If there are no such exceptions, as is the case of a policy from *refpol*, then the policy (of type $(\mathbb{P}\, class) \leftrightarrow class$) can be simplified to a relation of type $class \leftrightarrow class$. Finally, let *quaspol* represent the set of policies from *refpol* that are transitive (quasi or pre ordered).

$$\begin{aligned}quaspol == \{\ & P : policy \mid P \in refpol\ \wedge \\ & (\forall A : \mathbb{P}\, class;\ b, c : class \bullet \\ & \quad (A \mapsto b \in P) \wedge (\{b, c\} \mapsto c \in P) \Rightarrow \\ & \qquad ((A \cup \{c\}) \mapsto c \in P))\ \}\end{aligned}$$

2.2 A Framework for Flow Policies

A policy abstraction operator can be defined that hides certain classes of a flow policy, while preserving the flow restrictions over the remaining components. Given a policy P and a set $C : \mathbb{P}\,class$, then $P@C$ gives P abstracted to classes from C, where

$$
\begin{array}{|l}
\_@\_ : (policy \times \mathbb{P}\,class) \rightarrow policy \\
\hline
\forall\, P : policy;\ C : \mathbb{P}\,class \bullet \\
\qquad P@C = \{\ A : \mathbb{P}\,class;\ a : class \mid (A \mapsto a \in P) \wedge (a \in C) \\
\qquad\qquad\qquad\qquad \bullet\, (A \cap C) \mapsto a\ \}
\end{array}
$$

For example, the policy $\texttt{Military}@\{\texttt{confidential}, \texttt{top-secret}\}$ can be viewed as a relabeling of the policy `HiLo`.

There is a partial ordering relation $\sqsubseteq$ defined between flow policies. If $P \sqsubseteq Q$ then we say that Q is no less *restrictive* than P, in that any flow that is not allowed by P will also not be allowed by Q. Note that Q may introduce additional classes.

$$
\begin{array}{|l}
\_ \sqsubseteq \_ : policy \leftrightarrow policy \\
\hline
\forall\, P, Q : policy \bullet \\
\qquad (P \sqsubseteq Q) \Leftrightarrow (\alpha P \subseteq \alpha Q) \wedge (Q@\alpha P) \subseteq P
\end{array}
$$

Consider Example 1. The policy `HiLo` can be thought of as enforcing more flow restrictions that the least restrictive policy $\bot\{\texttt{lo}, \texttt{hi}\}$. Thus, we say that policy `HiLo` is no less restrictive than the least restrictive policy $\bot\{\texttt{lo}, \texttt{hi}\}$, i.e., $\bot\{\texttt{lo}, \texttt{hi}\} \sqsubseteq \texttt{HiLo}$. We also have $\texttt{HiLo} \sqsubseteq \top\{\texttt{lo}, \texttt{hi}\}$. The $\sqsubseteq$ relation can, in a sense, be viewed as a refinement-like relation. If $P \sqsubseteq Q$ then the policy defined by Q is, in a security sense, an acceptable replacement for the policy P. Intuitively, this means that a system secure by policy Q is also secure by policy P.

Abstraction is monotonic with respect to restrictiveness: given $P, Q : policy$; $B, C : \mathbb{P}\,class$, then

$$(B \subseteq C) \wedge (P \sqsubseteq Q) \Rightarrow (P@B \sqsubseteq Q@C)$$

The set of flow policies *policy* forms a lattice with partial order $\sqsubseteq$ and least upper and greatest lower bound operators defined by $\oplus$ and $\otimes$ respectively.

$$
\begin{array}{|l}
\_ \oplus \_, \\
\_ \otimes \_ : (\mathit{policy} \times \mathit{policy}) \rightarrow \mathit{policy} \\
\hline
\forall\, P, Q : \mathit{policy} \bullet \\
\quad P \oplus Q = \{\ A : \mathbb{P}\, \mathit{class};\ a : \mathit{class} \mid \\
\qquad\qquad (A \cup \{a\}) \subseteq (\alpha P \cup \alpha Q) \wedge \\
\qquad\qquad (a \in \alpha Q \Rightarrow (A \cap \alpha Q) \mapsto a \in Q) \wedge \\
\qquad\qquad (a \in \alpha P \Rightarrow (A \cap \alpha P) \mapsto a \in P) \bullet A \mapsto a\ \} \wedge \\
\quad P \otimes Q = (P @ (\alpha P \cap \alpha Q)) \cup (Q @ (\alpha P \cap \alpha Q))
\end{array}
$$

Note that we have, $\alpha(P \oplus Q) = \alpha P \cup \alpha Q$ and $\alpha(P \otimes Q) = \alpha P \cap \alpha Q$. If A is a set of conglomerate policies, then $\bigoplus A$ gives the generalized join of the policies in A, i.e., the least upper bound of the policies in A. The policy join operator is closed over aggregation, separation and reflexive flow policies, but is not closed over quasi ordered policies: if $P, Q \in \mathit{aggpol}$ then $P \oplus Q \in \mathit{aggpol}$; if $P, Q \in \mathit{refpol}$ then $P \oplus Q \in \mathit{refpol}$, and if $P, Q \in \mathit{seppol}$ then $P \oplus Q \in \mathit{seppol}$.

The policy join operator ($\oplus$) may be used to construct complex policies from simpler policies. Given policies P and Q, since policy join gives their least upper bound, then $(P \oplus Q)$ is no less restrictive than policies P and Q. This implies that $(P \oplus Q)$ enforces all the flow restrictions specified by P and by Q. Intuitively, this means that a system secure by policy $(P \oplus Q)$ is also secure by the policies P and Q. The policy intersection operator ($\otimes$) does not appear to have a useful interpretation in the construction of information flow policies; we have included it only for completeness.

Example 6 Consider the aggregation policy from Example 4. Introduce a new class `sso` which represents the class of information the system security officer may handle. The security officer is allowed to sink `bank1` and `bank2` information individually and as an aggregate. The policy governing the security officer is

$$\mathtt{SsoPol} == \{\mathtt{bank1}, \mathtt{bank2}\}\ \mathrm{aggflow}\ \mathtt{sso}$$

and the overall bank flow policy is

$$\mathtt{NewBankPol} == \mathtt{BankPol} \oplus \mathtt{SsoPol}$$

This relation `NewBankPol` can also be calculated as

$$
\begin{array}{l}
\mathtt{NewBankPol} = \\
\quad (\{\mathtt{bank1}, \mathtt{sso}\}\ \mathrm{aggflow}\ \mathtt{cons}) \cup (\{\mathtt{bank2}, \mathtt{sso}\}\ \mathrm{aggflow}\ \mathtt{cons}) \cup \\
\quad (\{\mathtt{bank1}, \mathtt{bank2}, \mathtt{cons}\}\ \mathrm{aggflow}\ \mathtt{sso}) \cup \top\{\mathtt{bank1}, \mathtt{bank2}\}
\end{array}
$$

Under this new policy any flow is permitted so long as it does not violate the flow restrictions of either policy. Thus, since neither policy imposes restrictions on flows between cons and sso, then information is allowed to flow between consultant and security officer. Of course, aggregate information $\{\mathtt{sso}, \mathtt{bank1}, \mathtt{bank2}\}$ is not allowed to flow to cons. △

Example 7 The last example illustrated how the policy join operator may be used to compose policies that have intersecting alphabets. It may also be used to join policies that have completely disjoint alphabets. For example, a policy that enforces the flow restrictions specified by HiLo (Example 1) and BankPol (Example 4) is defined by:

$$\mathtt{HiLoBankPol} == \mathtt{HiLo} \oplus \mathtt{BankPol}$$

This policy permits any flow that does not violate HiLo or BankPol and can be calculated as

$$\begin{aligned}
&\mathtt{HiLoBankPol} = \\
&\quad (\{\mathtt{cons}, \mathtt{bank1}, \mathtt{bank2}\} \text{ aggflow } \mathtt{lo}) \cup \\
&\quad (\{\mathtt{cons}, \mathtt{bank1}, \mathtt{bank2}, \mathtt{lo}\} \text{ aggflow } \mathtt{hi}) \cup \\
&\quad (\{\mathtt{lo}, \mathtt{hi}, \mathtt{bank1}\} \text{ aggflow } \mathtt{cons}) \cup (\{\mathtt{lo}, \mathtt{hi}, \mathtt{bank2}\} \text{ aggflow } \mathtt{cons}) \cup \\
&\quad (\{\mathtt{lo}, \mathtt{hi}\} \text{ aggflow } \mathtt{bank1}) \cup (\{\mathtt{lo}, \mathtt{hi}\} \text{ aggflow } \mathtt{bank2})
\end{aligned}$$

Note how the new policy permits flow from both lo and hi to cons (neither operand policy specifies a restriction to the contrary), but no conglomerate involving *both* bank1 and bank2 may flow to cons. △

It is possible when developing a policy to end up with an unintentionally restrictive result. For example, if a policy was defined as

$$\mathtt{BankPol3} == \mathtt{NewBankPol} \oplus (\{\mathtt{cons}\} \text{ aggflow } \mathtt{bank1})$$

Then $\mathtt{BankPol} \sqsubseteq \mathtt{BankPol3}$ holds, but this new policy no longer permits information flow from bank1 to cons—the join of two policies must preserve the flow *restrictions* of the original policies. When building complex policies it may be worth investigating if the intended allowable flows of the original policy are preserved. For Example 6, $\mathtt{BankPol} = \mathtt{NewBankPol} @ \alpha \mathtt{BankPol}$, giving reassurance that the intentions of the old policy are preserved in the new policy.

The set of all flow policies with the same alphabet A forms an algebra that is a sublattice of *policy* with restrictiveness defined by superset; $\oplus$

defined by intersection; $\otimes$ defined by union, and universal upper and lower bounds defined by $\top A$ and $\bot A$, respectively. Since it forms an algebra, each policy P has a complement within the sublattice defined by (comp P), where

$$\begin{array}{|l}
\text{comp} \ : \mathit{policy} \rightarrow \mathit{policy} \\
\hline
\forall\, P : \mathit{policy} \bullet \\
\qquad \text{comp}\ P = (\top(\alpha P)) \cup ((\bot(\alpha P)) \setminus P)
\end{array}$$

The following example illustrates how the policy complement operator can be used to construct a policy in terms of the flows that are not permitted.

Example 8 Consider the Military style policy from Example 2. Introduce new classes **`brigadier`**, **`colonel`** and **`major`** to represent the class of information that may be handled by a brigadier, colonel and major, respectively. We shall approach specifying this flow policy in terms of the flow restrictions for the individual classes.

A brigadier with the highest rank has no flow restrictions enforced. The policy for a brigadier is simply

$$\mathtt{BrigPol} == \bot\{\mathtt{brigadier}\}$$

A major, with the lowest rank of the three, may not sink secret or top-secret information. This is specified as

$$\mathtt{MajPol} == \text{comp}\ (\{\mathtt{secret}, \mathtt{top\text{-}secret}\}\ \text{aggflow}\ \mathtt{major})$$

The Colonel may not sink top-secret information and thus

$$\mathtt{ColPol} == \text{comp}\ (\{\mathtt{top\text{-}secret}\}\ \text{aggflow}\ \mathtt{colonel})$$

The overall policy must enforce the flow restrictions of the above policies in addition to the usual policy `Military`. The policy is

$$\mathtt{NewMil} == \mathtt{Military} \oplus \mathtt{BrigPol} \oplus \mathtt{ColPol} \oplus \mathtt{MajPol}$$

Policy join ($\oplus$) permits *any* flow that does not violate its operand policies. Thus, for example, the following hold

$$(\alpha\mathtt{NewMil} \mapsto \mathtt{brigadier}) \in \mathtt{NewMil}$$
$$(\{\mathtt{brigadier}, \mathtt{major}\} \mapsto \mathtt{major}) \in \mathtt{NewMil}$$
$$(\{\mathtt{brigadier}, \mathtt{secret}, \mathtt{major}\} \mapsto \mathtt{major}) \notin \mathtt{NewMil}$$

A brigadier may sink any conglomerate (no constraints specifying otherwise are defined by of the operand policies), and may source information to a general. However, the brigadier may not 'forward' secret information to a general. $\triangle$

3 Conglomerate Noninterference

This section defines a meaning for security under a conglomerate policy. It is based on the notion of noninterference[16]. We deliberately choose the original system model from [16] to simplify our exposition. We see no reason why the theory presented in this section could not be redeveloped for other models.

A system is viewed as a deterministic state machine. It operates in an environment which selects actions (commands) to be engaged by the system; the selection of an action causes the system to move to a new state, and results in outputs or views for the environment. Introduce the basic types

$$[state, action, view]$$

Define the state transition function to be of the form

$$do : (state \times action) \to state$$

where $do(s, x)$ gives the state resulting from engaging action x in state s. Let *initstate* be the initial state.

$$initstate : state$$

The environment of the system has some structure in that the set of actions it can engage in is partitioned by conglomerate classes using the binding

$$\begin{array}{|l}
syspol : policy \\
\beta : action \to \mathbb{P}\, class \\
\hline
\forall\, x : action \bullet \\
\qquad (\beta x) \subseteq \alpha syspol
\end{array}$$

Policy *syspol* gives the conglomerate policy to be enforced. Function β gives the conglomerate class at which an action is engaged. The usual multilevel treatment of noninterference associates a single class with each action. For example, a secret user engages a secret action. It is desirable to model actions engaged at conglomerate classes: a colonel (Example 8) may wish to distinguish between entering secret or confidential information. Any action engaged by the colonel that involves secret information (as ultimately determined by the colonel), should have binding $\{\texttt{colonel}, \texttt{secret}\}$. In a practical system identifying such a conglomerate source might be achieved by associating classes (or conglomerates) with system objects such as windows, programs, etc. When the colonel invokes a command within a secret window, this is a conglomerate source $\{\texttt{colonel}, \texttt{secret}\}$[1]. Colonel actions involving confidential information should have binding $\{\texttt{colonel}, \texttt{confidential}\}$.

Engaging an action at some state may be visible, in some way, at a number of different classes (i.e., visible to users bound to those classes). An action x engaged at state s produces a view $out(s, x, a)$ to class a if (s, x, a) is in the domain of (partial) function out

$$out : (state \times action \times class) \rightarrow\!\!\!\!\!\!+\; view$$

A typical interpretation of function out is that at state s, action x generates output $out(s, x, a)$ at class a if $a \in views(s, x)$ where

$$\begin{array}{|l}
views : (state \times action) \rightarrow \mathbb{P}\, class \\
\hline
\forall\, s : state;\; x : action;\; a : class \bullet \\
\qquad (a \in views(s, x)) \Leftrightarrow ((s, x, a) \in \operatorname{dom} out)
\end{array}$$

A sequence of actions (trace) t, engaged from the initial state, will have a final state $do^*(t)$, where

$$\begin{array}{|l}
do^* : (\operatorname{seq} action) \rightarrow state \\
\hline
\forall\, x : action;\; t : \operatorname{seq} action \bullet \\
\qquad do^*\langle\rangle = initstate \wedge \\
\qquad do^*(t \frown \langle x \rangle) = do((do^* t), x)
\end{array}$$

The sequence of views at a class a generated by a trace t is defined by $out^*(t, a)$, where

[1]McLean[19] uses a similar scheme for classifying actions. This strategy can be used in an effort to avoid the problem of high outputs flowing to low outputs.

$$
\begin{array}{|l}
out{*} : ((\mathrm{seq}\, action) \times class) \rightarrow (\mathrm{seq}\, view) \\
\hline
\forall\, t : \mathrm{seq}\, action;\ x : action;\ a : class \bullet \\
\quad out{*}(\langle\rangle, a) = \langle\rangle \wedge \\
\quad (a \in views(do{*}(t), x)) \Rightarrow \\
\qquad out{*}(t \frown \langle x \rangle, a) = out{*}(t, a) \frown \langle out(do{*}(t), x, a) \rangle \wedge \\
\quad (a \notin views(do{*}(t), x)) \Rightarrow \\
\qquad out{*}(t \frown \langle x \rangle, a) = out{*}(t, a)
\end{array}
$$

Define a purge function that removes actions bound to a particular conglomerate class A from trace t to be the trace t/A, where

$$
\begin{array}{|l}
\_/\_ : ((\mathrm{seq}\, action) \times \mathbb{P}\, class) \rightarrow \mathrm{seq}\, action \\
\hline
\forall\, t : \mathrm{seq}\, action;\ A : \mathbb{P}\, class \bullet \\
\quad t/A = t \upharpoonright \{\ x : action \mid \beta x \neq A\ \}
\end{array}
$$

Actions at conglomerate class B can interfere with the view at class a along a trace t, if purging for B along t results in a different view for a, i.e., $out{*}(t, a) \neq out{*}(t/B, a)$. For a given trace, there may be a number of such conglomerate classed actions that can interfere with the view of class a. For a given sequence of actions t, the conglomerate class of *information* that can interfere with the view at class a is given by $\delta(t, a)$, where

$$
\begin{array}{|l}
\delta : ((\mathrm{seq}\, action) \times class) \rightarrow \mathbb{P}\, class \\
\hline
\forall\, t : (\mathrm{seq}\, action);\ a : class \bullet \\
\quad \delta(t, a) = \bigcup\{\ B : \mathbb{P}\, class \mid out{*}(t, a) \neq out{*}(t/B, a)\ \} \cup \{a\}
\end{array}
$$

Note that we automatically include class a in $\delta(t, a)$. This reflects the hypothesis discussed at the beginning of Section 2 that information is considered to flow always to itself.

Definition 1 A system (with policy *syspol* and binding β defined above) is secure iff

$$
\begin{array}{l}
\forall\, t : (\mathrm{seq}\, action);\ a : \alpha syspol \bullet \\
\quad \delta(t, a) \mapsto a \in syspol
\end{array}
$$

$\diamond$

By recording a history (rather than state by state) of the actions interfering with a given class, aggregation exceptions are identified. The capture of a separation exception is not unlike conditional noninterference[16]. However, conglomerate flow requires more than just some action to condition a flow. It also necessitates the action to have some visible effect on the flow.

Example 9 Consider a simple system enforcing the policy `BankPol` (Example 4). Users may be bound to a single class, `bank1`, `bank2` or `cons`, that is, each command is bound (by β) to a singleton set. A consultant may request a count on the number of actions made by any bank up to that moment. The following commands are possible: *B1cmd* or *B2cmd* representing some action made by a user bound to conglomerate class $\{\texttt{bank1}\}$ or $\{\texttt{bank2}\}$ respectively, and *B1req* or *B2req*, representing a consultant making a request for a count. For simplicity, commands *B1cmd* and *B2cmd* do not generate any output. For this system to be secure (by `BankPol`), the consultant should not be permitted to access counting information about both banks.

Let the system state be represented as a vector of variables denoted $s_{n,m,f}$. The number of bank1 and bank2 requests received so far is given by n and m, respectively. Flag f indicates whether the consultant has requested a count on bank1 or bank2 (denoted by the values `B1acc` or `B2acc`, respectively). Flag f is initially set to value `nil`. A secure specification for this system is described in Table 1. This table describes the resulting

$do(s,x)$	*B1cmd*	*B2cmd*	*B1req*	*B2req*
$s_{n,m,\texttt{nil}}$	$s_{n+1,m,\texttt{nil}}$	$s_{n,m+1,\texttt{nil}}$	$\frac{s_{n,m,\texttt{B1acc}}}{\{\texttt{cons}\mapsto n\}}$	$\frac{s_{n,m,\texttt{B2acc}}}{\{\texttt{cons}\mapsto m\}}$
$s_{n,m,\texttt{B1acc}}$	$s_{n+1,m,\texttt{B1acc}}$	$s_{n,m+1,\texttt{B1acc}}$	$\frac{s_{n,m,\texttt{B1acc}}}{\{\texttt{cons}\mapsto n\}}$	$s_{n,m,\texttt{B1acc}}$
$s_{n,m,\texttt{B2acc}}$	$s_{n+1,m,\texttt{B2acc}}$	$s_{n,m+1,\texttt{B2acc}}$	$s_{n,m,\texttt{B2acc}}$	$\frac{s_{n,m,\texttt{B2acc}}}{\{\texttt{cons}\mapsto m\}}$

Table 1: Enforcing Aggregation Policy `BankPol`.

state transition $do(s,x)$ given state s (along rows) and command x (down columns). If a particular transition generates output, then the entry in the table is of the form $\frac{s}{op}$, where s is the resulting state and op an output function (of type $class \nrightarrow view$) which determines the values for out. For example, given state $s_{n,m,\texttt{nil}}$ and command *B1req*, the output is represented by $\{\texttt{cons} \mapsto n\}$ implying that there is an output n visible to class `cons`. Intuitively this system is secure—as soon as the consultant observes some information concerning one bank (number of requests), he is denied access to the other bank. Section 4, which unwinds the definition of security, will consider how this proof may be conducted. △

Example 10 Consider the simple multilevel policy `HiLo` (Example 1). A (simple) database system maintains information about a *single* aircraft flight. Let `flight` denote this new class of information. There are no restrictions on the flows between this class and classes `lo` and `hi`

$$\texttt{FlightPol} == \texttt{HiLo} \oplus (\bot\{\texttt{flight}\})$$

This is a reflexive (non-transitive) policy. Users may be bound to single classes `lo`, `hi`, or `flight` (a pilot might be a user bound to class `flight`).

We wish to specify a system that permits these users read and write access to the flight information, yet enforcing the above flow policy. This system provides commands of the form $a.x$, where a gives the class of the command (engaged by the user) and x the request: `rd` denotes a read request, and $\texttt{wr}.i$ a write (data i) request. Only read commands generate output (if it is secure to do so) at the class of the requester. A state of this system is represented by a vector $s_{a,i}$, where i gives the (single) flight details and a a security class (`lo` or `hi`) associated with the information. The initial state is $s_{lo,\emptyset}$. Table 2 gives a secure (by policy `FlightPol`) specification of this system. In this specification, the flight information is considered to be

$do(s,c)$	$lo.rd$	$lo.wr.y$	$hi.rd$	$hi.wr.y$	$fl.rd$	$fl.wr.y$	$lo.up$
$s_{lo,x}$	$\frac{s_{lo,x}}{\{lo \mapsto x\}}$	$s_{lo,y}$	$\frac{s_{lo,x}}{\{hi \mapsto x\}}$	$s_{lo,x}$	$\frac{s_{lo,x}}{\{fl \mapsto x\}}$	$s_{lo,y}$	$s_{hi,x}$
$s_{hi,x}$	$s_{hi,x}$	$s_{hi,y}$	$\frac{s_{hi,x}}{\{hi \mapsto x\}}$	$s_{hi,y}$	$\frac{s_{hi,x}}{\{fl \mapsto x\}}$	$s_{hi,y}$	$s_{hi,x}$

Table 2: Enforcing Reflexive Policy `FlightPol`.

either `lo` or `hi` An upgrading command $lo.up$ is provided which forces the level of the flight to `hi`. This may only be engaged at class `lo`. To enter a `hi` flight, the `hi` user must first log on as `lo`, upgrade, and then return to `hi` to update. Once upgraded, the `lo` user no longer has read access to the flight variable.

There are alternative approaches to specifying a system that enforces policy `FlightPol`. In this example, stored flight information is modeled by a flight field (to hold the information) and a class field (for its classification). The upgrading strategy that permits the single variable to be, in a sense shared by the three classes of users, follows the advice on truthful database update[26]. An alternative approach is to polyinstantiate[9]. That is, have two separate copies of the flight variable, one for `lo` and the other for `hi`, information. $\triangle$

4 Unwinding Conglomerate Noninterference

Conglomerate noninterference is defined in terms of sequences of actions and views. This section considers how the definition may be unwound into conditions on states and state transitions. The unwinding approach taken makes an approximation for $\delta(t, a)$ by permitting overestimation of information flow. Thus the unwinding is sufficient for enforcing aggregation policies (any policy from *aggpol*, which includes reflexive and quasi ordered policies), but is unsuitable for policies that include separation exceptions.

The approach taken in this section is to first transform the (aggregation) policy into conditions over a lattice-based policy. The unwinding of conglomerate noninterference then becomes a generalization of the usual unwinding conditions for traditional noninterference. This is significant, as there is a quite a body of experience in building systems that enforce similar unwinding conditions.

4.1 High Water Marks and Aggregation Policies

Given the aggregation policy *syspol*, then for any class $a : \alpha syspol$ the set of conglomerate classes that may flow to a can be considered to be drawn from the enclosed area pictured in Figure 1. Since the policy does not contain

$$lims(a) = \text{maxs}\ \{\ A : \mathbb{P}\, class \mid A \mapsto a \in syspol\ \}$$

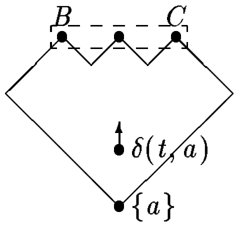

Figure 1: The set $\{\ B : \mathbb{P}\, class \mid B \mapsto a \in syspol\ \}$

separation exceptions, the set is delimited (under the subset ordering) by a lowest element $\{a\}$ and a set of maximums $lims(a)$. Function $\text{maxs}\mathcal{A}$ returns the (smallest) set of maximums in $\mathcal{A}$ under the subset ordering.

Function $\delta(t, a)$ records a history of the information that has interfered with the view of a given class. It follows from its definition that it is monotonic with respect to progress along a trace, i.e.,

$$\forall\, t, s : \operatorname{seq} action;\ a : class \bullet$$
$$\quad \delta(t, a) \subseteq \delta(t \frown s, a)$$

$\delta(t, a)$ is rather like a high water mark on a particular class: it rises to reflect the information classes that can interfere with the view. Having engaged trace t, in a secure system, the area between $\delta(t, a)$ and $lims(a)$ gives the allowable future high water marks for class a. As $\delta(t, a)$ rises, the set of possible future high water marks diminish. For example, if $\delta(t, a)$ heads towards limit C, then eventually limit B will no longer be a potential high water mark.

This interpretation of $\delta(t, a)$ (as a high water mark) provides the key to developing unwinding conditions for security under conglomerate policies. We shall redefine conglomerate security in terms of a system with a lattice-based flow policy providing high water marks (with limits on these water marks). The lattice flow policy is simply a powerset lattice of security classes with subset as the ordering relation.

4.2 Aggregation Policies as Lattices

If *classes* are the components of a conglomerate flow policy, then let *level* be the components of a powerset lattice built from the set of classes.

$$level == \mathbb{P}\, class$$

An aggregation policy P can be mapped to the powerset lattice built from components of αP by the dual mapping

$$\Phi_\perp : aggpol \rightarrow (\mathbb{P}\, class) \rightarrow\!\!\!\!\!+\, level$$
$$\Phi_{lim} : aggpol \rightarrow class \rightarrow\!\!\!\!\!+\, \mathbb{P}\, level$$

$$\forall\, P : aggpol;\ A : \mathbb{P}\, class \bullet A \subseteq (\alpha P) \Rightarrow$$
$$\quad \Phi_\perp\ P\ A = \{\ b : class \mid (\forall\, B : \mathbb{P}\, class;\ c : class \bullet$$
$$\qquad ((A \cup B) \mapsto c \in P) \Rightarrow$$
$$\qquad\quad (A \cup B \cup \{b\}) \mapsto c \in P)\ \}$$

$$\forall\, P : aggpol;\ a : class \bullet a \in \alpha P \Rightarrow$$
$$\quad \Phi_{lim}\ P\ a = \operatorname{maxs}\{\ B : \mathbb{P}\, class \mid B \mapsto a \in P\ \}$$

$(\Phi_{lim} P\ a)$ gives the maximal conglomerates that may flow to class a (recall Figure 1). $(\Phi_\perp P\ A)$ gives the set of classes that are the bounds of conglomerate A; a class b bounds conglomerate A if everything that conglomerates involving A may flow to, then b may also flow to. The definition for $(\Phi_{\perp}\ P\ A)$ requires some explanation. In Lemma 1 we shall prove that, given this mapping then, $A \mapsto a \in P$ iff $\exists\, B : (\Phi_{lim}\ P\ a) \bullet (\Phi_{\perp}\ P\ A) \subseteq B$. In this case we call the mapping *flow preserving*. There can be alternative definitions for $\Phi_{\perp}$ that achieve this same result. An intuitive (and valid) alternative might be to define $\Phi_{\perp}\ P\ A = A$, i.e., the 'bottom' of the diagram in Figure 1. The actual definition chosen for $(\Phi_{\perp}\ P\ A)$ attempts to capture the largest set that includes A and yet maintains Lemma 1. If we consider the singleton conglomerate $\{a\}$, then $\Phi_{\perp}\ P\ \{a\}$ lies somewhere between $\{a\}$ and $\bigcap \Phi_{lim}\ P\ a$. Informally, we notice that the size of $\Phi_{\perp}\ P\ \{a\}$ is proportional to how 'transitive' $\{a\}$ is, in that the more components $\{a\}$ bounds the larger/higher (and closer to $\bigcap \Phi_{lim}\ P\ a$) the value of $\Phi_{\perp}\ P\ \{a\}$ becomes. The reader should keep this description in mind when studying Examples 11, 12 and 14. A degenerate case occurs when we have a transitive policy and $\Phi_{\perp}\ P\ \{a\} = \bigcap \Phi_{lim}\ P\ a$. This case is discussed in Section 5.

Lemma 1 The dual mapping $(\Phi_{\perp}, \Phi_{lim})$ is flow preserving in the sense that

$$\begin{array}{l}
\forall\, P : aggpol \bullet \\
\quad \forall\, A : \mathbb{P}(\alpha P);\ a : \alpha P \bullet a \in A \Rightarrow \\
\quad\quad (A \mapsto a \in P) \Leftrightarrow \\
\quad\quad (\exists\, B : (\Phi_{lim}\ P\ a) \bullet (\Phi_{\perp}\ P\ A) \subseteq B)
\end{array}$$

Proof (if part) For $A : \mathbb{P}(\alpha P);\ a : \alpha P$, suppose that

$$\exists\, B : (\Phi_{lim}\ P\ a) \bullet (\Phi_{\perp}\ P\ A) \subseteq B$$

For any $b \in A$ we have

$$\begin{array}{l}
\forall\, c : class;\ B : \mathbb{P}\, class \bullet \\
\quad ((A \cup B) \mapsto c \in P) \Rightarrow ((A \cup B \cup \{b\}) \mapsto c \in P)
\end{array}$$

which given the definition of $\Phi_{\perp}$ implies that $A \subseteq \Phi_{\perp}\ P\ A$. Our initial assumption implies $(\Phi_{\perp}\ P\ A) \subseteq B$, and thus $A \subseteq B$. But $B \subseteq \Phi_{lim}\ P\ a$ and we also have $B \mapsto a \in P$. Since P is an aggregation policy $A \subseteq B$ implies that $A \mapsto a \in P$ also holds.

Proof (only if part) Suppose that $A \mapsto a \in P$. For every $b \in \Phi_{\perp}\ P\ A$ then for $B : \mathbb{P}\, class$

$$((A \cup B) \mapsto a \in P) \Rightarrow ((A \cup B \cup \{b\}) \mapsto a \in P)$$

Thus, if $A \mapsto a \in P$ and $b \in \Phi_\perp\ P\ A$ then $A \cup \{b\} \mapsto a \in P$. Similarly, if $c \in \Phi_\perp\ P\ A$ then we also have $A \cup \{b\} \cup \{c\} \mapsto a \in P$. By repeatedly 'adding' values from $\Phi_\perp\ P\ A$ we arrive at

$$(A \mapsto a \in P) \Rightarrow (A \cup \Phi_\perp\ P\ A) \mapsto a \in P$$

But we already know that $A \subseteq (\Phi_\perp\ P\ A)$, which implies that $(\Phi_\perp\ P\ A) \mapsto a \in P$ holds. Given this, the definition of Φ_{lim} implies that there exists some $B \in \Phi_{lim}\ P\ a$, such that $(\Phi_\perp\ P\ A) \subseteq B$, and the lemma is proven. □

Thus the definition for security can be re-stated as: a system (with aggregation policy *syspol* as before) is secure iff

$$\begin{array}{l} \forall\, t : \mathrm{seq}\, action;\ a : \alpha syspol \bullet \\ \qquad \exists\, B : (\Phi_{lim} syspol\ a) \bullet (\Phi_\perp syspol\ (\delta(t, a))) \subseteq B \end{array}$$

The lattice generated by this transformation is called the *implementation lattice* of the conglomerate policy. Note that is not necessary to use a powerset lattice. Given a policy P, a partially ordered (under $\subseteq$) set can be constructed from just those levels referenced by $\Phi_\perp$ and Φ_{lim}. This partially ordered set can in turn be transformed into a lattice using Denning's transformation[8] (see also [12] for an account of the transformation described in that report).

Example 11 Consider the simple policy `BankPol` (Example 4). This policy is transformed into the lattice (optimized using [8]) described in Figure 2 with flow preserving mappings from Table 3. Intuitively, the lev-

class a	$\Phi_\perp$`BankPol` $\{a\}$	Φ_{lim}`BankPol` a
`cons`	$\{$`cons`$\}$	$\{\{$`cons`, `bank1`$\}, \{$`cons`, `bank2`$\}\}$
`bank1`	$\{$`bank1`$\}$	$\{\{$`bank1`$\}\}$
`bank2`	$\{$`bank2`$\}$	$\{\{$`bank2`$\}\}$

Table 3: Flow Preserving Mapping for Policy `BankPol`

els of the implementation lattice correspond to the different conglomerate classes that are expected to exist in the system: $\{$`bank1`$\}$, Bank1 information; $\{$`cons`$\}$, consultant information (excluding banking information), and $\{$`cons`, `bank1`$\}$, aggregate consultant and Bank1 information. Note that, according to the flow policy, level $\{$`bank1`, `bank2`$\}$ does not exist, while levels $\varnothing$ and $\{$`cons`, `bank1`, `bank2`$\}$ are included only to ensure a lattice structure. △

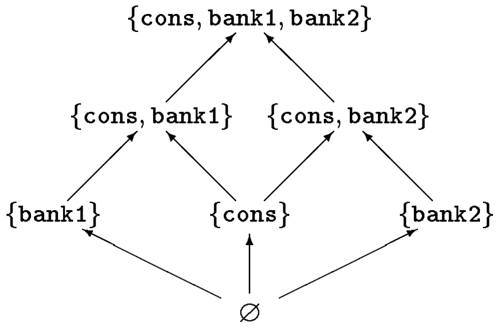

Figure 2: Implementation Lattice for Policy `BankPol`.

Example 12 The policy `HiLoBankPol` (Example 7) can be transformed into the (optimized) lattice described in Figure 3 with the flow preserving mapping from Table 4. Observe that for any class a of `HiLoBankPol`,

a	$\Phi_{\perp}\, syspol\ \{a\}$	$\Phi_{lim}\ syspol\ a$
`cons`	$\{\texttt{cons}, \texttt{lo}\}$	$\{\{\texttt{cons}, \texttt{lo}, \texttt{hi}, \texttt{bank1}\}, \{\texttt{cons}, \texttt{lo}, \texttt{hi}, \texttt{bank2}\}\}$
`bank1`	$\{\texttt{bank1}, \texttt{lo}\}$	$\{\{\texttt{bank1}, \texttt{lo}, \texttt{hi}\}\}$
`bank2`	$\{\texttt{bank2}, \texttt{lo}\}$	$\{\{\texttt{bank2}, \texttt{lo}, \texttt{hi}\}\}$
`lo`	$\{\texttt{lo}\}$	$\{\{\texttt{lo}, \texttt{cons}, \texttt{bank1}, \texttt{bank2}\}\}$
`hi`	$\{\texttt{lo}, \texttt{hi}\}$	$\{\{\texttt{lo}, \texttt{hi}, \texttt{cons}, \texttt{bank1}, \texttt{bank2}\}\}$

Table 4: Flow Preserving Mapping for policy `HiLoBankPol`

class `lo` is a member of $\Phi_{\perp} syspol\{a\}$, reflecting the fact that `lo` can flow to *any* class of the policy. The previous example (11) provided an intuition for levels in an implementation lattice as the conglomerates that may arise in the system. In this current example, a level such as $\{\texttt{bank1}, \texttt{lo}\}$ can be thought of as representing the conglomerates $\{\texttt{bank1}\}$ *and* $\{\texttt{bank1}, \texttt{lo}\}$—in the policy `HiLoBankPol`, flows involving either of these two conglomerates are treated in an identical manner. $\triangle$

4.3 Approximating High Water Marks

In this section we will develop an unwound approximation of the δ function. This approximation over estimates flows and is therefore sufficient for maintaining security.

In addition to generating immediate outputs or views, actions may change

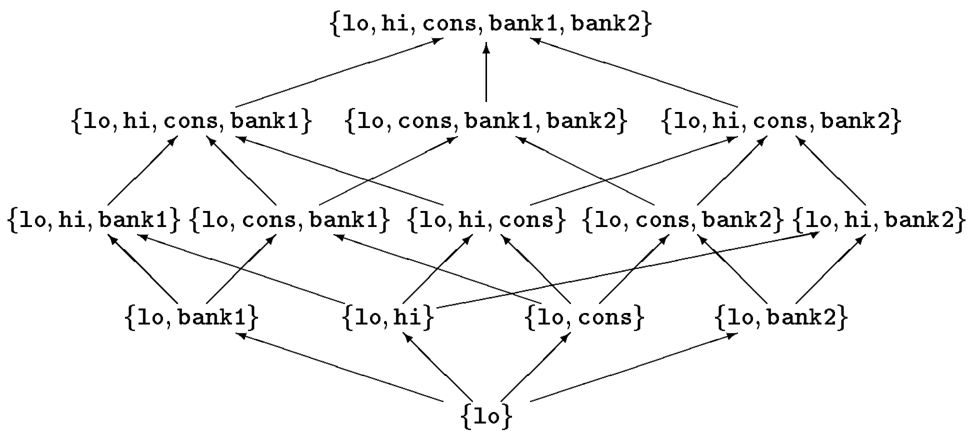

Figure 3: Implementation Lattice for Policy `HiLoBankPol`

the state of the system in such a way that it can affect a view from some later state. The usual way to model this information content in a state is by way of an equivalence relation between states. States s and r *appear* the same to level l if $s \approx_l r$.

$$
\begin{array}{|l}
\_ \approx_{\_} \_ : \mathit{level} \rightarrow (\mathit{state} \leftrightarrow \mathit{state}) \\
\hline
\forall\, l : \mathit{level};\ q, r, s : \mathit{state} \bullet \\
\qquad (s \approx_l s) \wedge \\
\qquad (s \approx_l r) \Rightarrow (r \approx_l s) \wedge \\
\qquad (s \approx_l r) \wedge (r \approx_l q) \Rightarrow (s \approx_l q)
\end{array}
$$

A typical interpretation is to bind all state components to levels (from the calculated lattice), and then $s \approx_l r$ holds iff the values for all objects bound to level l are indistinguishable between states s and r.

An action x with conglomerate binding βx is viewed, in terms of the implementation lattice, as an action with level binding ξx, where

$$
\begin{array}{|l}
\xi : \mathit{action} \rightarrow \mathit{level} \\
\hline
\forall\, x : \mathit{action} \bullet \\
\qquad \xi x = (\Phi_{\perp}\ \mathit{syspol}\ (\beta x))
\end{array}
$$

The level binding for an action is static throughout the history of a system.

An approximation for the δ function is the unwound high water mark function δ_{uwn}, defined as

$$
\begin{array}{|l}
\delta_{uwn} : (state \times class) \rightarrow level \\
\hline
\forall\, a : class \bullet \\
\qquad \delta_{uwn}(initstate, a) = \bigcap(\Phi_{lim}\ syspol\ a) \\
\forall\, a : class;\ s : state \bullet \\
\qquad \exists\, B : (\Phi_{lim}\ syspol\ a) \bullet \delta_{uwn}(s, a) \subseteq B \\
\forall\, a : class;\ s : state;\ x : action \bullet \\
\qquad \delta_{uwn}(s, a) \subseteq \delta_{uwn}(do(s, x), a)
\end{array}
$$

The (approximate) high water mark for class a at state s is $\delta_{uwn}(s, a)$. Recall from Figure 1 that any approximation for $\delta(t, a)$ must always lie between $\{a\}$ and $\Phi_{lim} syspol\ a$. The constraints given ensure that its value (a level) lies between $\bigcap(\Phi_{lim} syspol\ a)$ and some limit from $(\Phi_{lim} syspol\ a)$, and that it is monotonic along state transitions (i.e., may only float upwards). As information flows to class a, the high water mark of a must be increased to reflect the new information content of a. Since we are enforcing aggregation policies (drawn from *aggpol*) then choosing a greater initial level than $\delta(\langle\rangle, a)$ will have no effect on security. We choose $\delta_{uwn}(initstate, a)$ to have the greatest possible level that is dominated by all limits on a i.e., the greatest lower bound on the elements of $\Phi_{lim} syspol\ a$. In doing so, we do not exclude any conglomerate that the high water mark (of a) may be legally increased by. The following unwinding conditions further constrain $\delta_{uwn}(s, a)$:

Definition 2 (Unwinding Conditions)

C1 An action may not interfere with any part of the state that it may not flow to. This can be thought of as a generalization of the star property from [1].

$$
\begin{array}{l}
\forall\, l : level;\ s : state;\ x : action \bullet \\
\qquad \neg\,((\xi x) \subseteq l) \Rightarrow \\
\qquad\qquad (s \approx_l do(s, x))
\end{array}
$$

C2 The outcome on the state as a result of engaging an action must be completely determined by the visible part of the state. This can be thought of as an extension to the simple security condition from [1].

$$
\begin{array}{l}
\forall\, l : level;\ x : action;\ s, r : state \bullet \\
\qquad ((\xi x) \subseteq l) \wedge (s \approx_l r) \Rightarrow \\
\qquad\qquad do(s, x) \approx_l do(r, x)
\end{array}
$$

C3 The high water mark on each class that has a view on an action must dominate the action itself.

$$\begin{array}{l} \forall\, a : class;\ s : state;\ x : action \bullet \\ \quad a \in views(s, x) \Rightarrow \\ \qquad (\xi x) \subseteq \delta_{uwn}(do(s, x), a) \end{array}$$

C4 Every view must be completely determined by that part of the state that is dominated by the high water mark of the view's class.

$$\begin{array}{l} \forall\, a : class;\ x : action;\ s, r : state \bullet \\ \quad (\forall\, l : level \bullet (l \subseteq \delta_{uwn}(do(s, x), a)) \Rightarrow (s \approx_l r)) \\ \quad \Rightarrow \\ \quad (out(s, x, a) = out(r, x, a)) \end{array}$$

$\diamond$

Conditions C1 and C2 are concerned with ensuring that high actions do not interfere with the low part of the state. Versions of these conditions appear to form the basis of any unwinding for noninterference. For example, they are similar to conditions 2 and 3 in [10] and [11]. Condition C3 ensures consistency between actions and the views that result: every view must dominate the action. Condition C4 ensures that the high portion of the state does not interfere with low views. These last two conditions are concerned with that part of noninterference analysis that considers the relationship between outputs and the system state. For example, they are similar, at least in intent, to condition 4 in [11].

Lemma 2 Purging along a trace for a given conglomerate class has no effect on that part of the state that the conglomerate may not flow to, i.e.,

$$\begin{array}{l} \forall\, l : level;\ t : (\mathrm{seq}\, action);\ A : \mathbb{P}\, class \bullet \\ \quad \neg\, ((\Phi_\perp\ syspol\ A) \subseteq l) \Rightarrow \\ \qquad (do^*(t) \approx_l do^*(t/A)) \end{array}$$

Proof Use induction on the length of a trace. If $\#t = 0$ then $t = t/A$, and $do^*(t) \approx_l do^*(t/A)$, and the base case trivially holds. Assume that the lemma holds for traces up to length n (hypothesis). For any $l : level$, conglomerate $A : \mathbb{P}\, class$ such that $\neg\, ((\Phi_\perp\ syspol\ A) \subseteq l)$, and trace t with $\#t = n$, then:

- Consider trace $t ^\frown \langle x \rangle$ such that $\neg\ ((\xi x) \subseteq l)$. Unwinding condition C1 gives
$$do*(t) \approx_l do(do*(t), x) \qquad (1)$$
If $t ^\frown \langle x \rangle$ is to be purged for ξx, i.e., $\beta x = A$, then $(t ^\frown \langle x \rangle / A) = (t/A)$. But the inductive hypothesis, equation (1) above, and the transitivity of $\approx_l$ together imply that $do*(t ^\frown \langle x \rangle) \approx_l do*(t ^\frown \langle x \rangle / A)$.
If $t ^\frown \langle x \rangle$ is not to be purged for ξx, i.e., $\beta x \neq A$, then in addition to equation (1) above, unwinding condition C1 also implies (since $\neg\ (\xi x \subseteq l)$) that
$$do*(t/A) \approx_l do(do*(t/A), x)$$
This, equation (1), the inductive hypothesis, and transitivity of $\approx_l$, together imply that
$$do*(t ^\frown \langle x \rangle) \approx_l do*(t ^\frown \langle x \rangle / A)$$
Thus we have
$$\neg\ (\xi x \subseteq l) \Rightarrow do*(t ^\frown \langle x \rangle) \approx_l do*(t ^\frown \langle x \rangle / A) \qquad (2)$$
- Consider $t ^\frown \langle x \rangle$ such that $\xi x \subseteq l$. Unwinding condition C2 gives
$$\begin{array}{l} do*(t) \approx_l do*(t/A) \Rightarrow \\ \quad do(do*(t), x) \approx_l do(do*(t/A), x) \end{array}$$
But since $(\xi x) \neq (\Phi_\perp\ syspol\ A)$, then $(t/A) ^\frown \langle x \rangle = (t ^\frown \langle x \rangle)/A$. Thus we have that the inductive hypothesis implies that
$$(\xi x \subseteq l) \Rightarrow do*(t ^\frown \langle x \rangle) \approx_l do*(t ^\frown \langle x \rangle / A) \qquad (3)$$

By induction, equations (2) and (3) imply the lemma holds. □

Lemma 3 For any system that maintains the unwinding conditions, then δ_{uwn} is a suitable approximation for δ in the sense that,

$$\begin{array}{l} \forall\, a : class;\ t : \mathrm{seq}\, action \bullet \\ \quad \delta(t, a) \subseteq \delta_{uwn}(do*(t), a) \end{array}$$

Proof Use induction on the length of a trace. Consider trace t and class a. If $\#t = 0$ then by definition $\delta(t, a) = \{a\}$. The the (initial) high water mark on a is defined to be $\bigcap(\Phi_{lim}\, syspol\ a)$, which is a superset of $\Phi_\perp\ syspol\ \{a\}$. But $a \in \Phi_\perp\ syspol\ \{a\}$ holds (see Lemma 1), and thus $a \in \delta_{uwn}(initstate, a)$. But $\delta(t, a) = \{a\}$ and therefore $\delta(t, a) \subseteq \delta_{uwn}(do^*(\langle\rangle), a)$ holds (base case). Assume that the lemma holds for traces up to length n (hypothesis). Consider the trace $t \frown \langle x \rangle$ such that $\#t = n$. We wish to prove that for any $a, b : class$ then $b \in \delta(t \frown \langle x \rangle, a)$ implies $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$:

- If $b \in \delta(t, a)$, then by the inductive hypothesis $b \in \delta_{uwn}(do^*(t), a)$ holds. But δ_{uwn} may only increase along a trace, and therefore $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$.
- If $a \notin views(do^*(t), x)$, then action x is not visible to class a. Thus if $b \in \delta(t \frown \langle x \rangle, a)$, then $b \in \delta(t, a)$ follows since the purging of action x is not visible to a. The induction hypothesis implies that $b \in \delta_{uwn}(do^*(t), a)$, and thus $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$.
- Consider $b \notin \delta(t, a)$ and $a \in views(do^*(t), x)$—action x is visible in some way to class a. This implies that there is some conglomerate class B of which b is a component, such that the effect of purging B in t is not visible to a, but purging B in $t \frown \langle x \rangle$ is (visible to a), i.e.,

$$\begin{array}{l} \exists\, B : \mathbb{P}\, class \bullet b \in B \land \\ \qquad out^*(t, a) = out^*(t/B, a) \land \\ \qquad\qquad out^*(t \frown \langle x \rangle, a) \neq out^*(t \frown \langle x \rangle / B, a) \end{array} \tag{4}$$

  If action x is purgeable ($\beta x = B$) then by unwinding condition C3 $\xi(x) \subseteq \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$. But $B \subseteq \Phi_\perp\ syspol\ B$ (see Lemma 1) and $b \in B$. Thus $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$.
  If action x is not purgeable ($\beta x \neq B$) then equation 4 implies that $out(do^*(t), x, a) \neq out(do^*(t/B), x, a)$. Given this, unwinding condition C4 implies,

$$\begin{array}{l} \exists\, l : level \bullet \\ \qquad l \subseteq \delta_{uwn}(do^*(t \frown \langle x \rangle), a) \land \neg\, (do^*(t) \approx_l do^*(t/B)) \end{array}$$

  But the purging lemma (2) implies that

$$\begin{array}{l} \forall\, l : level \bullet \\ \qquad \neg\, ((\Phi_\perp\ syspol\ B) \subseteq l) \Rightarrow (do^*(t) \approx_l do^*(t/B)) \end{array}$$

and thus we have

$$\exists\, l : \mathit{level} \bullet$$
$$(l \subseteq \delta_{uwn}(do^*(t \frown \langle x \rangle), a) \wedge ((\Phi_\perp\ \mathit{syspol}\ B) \subseteq l)$$

Since $b \in B$ and thus $b \in \Phi_\perp\ \mathit{syspol}\ B$, the transitivity of $\subseteq$ implies $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$.

Thus, if $b \in \delta(t \frown \langle x \rangle, a)$ then $b \in \delta_{uwn}(do^*(t \frown \langle x \rangle), a)$, and by induction the lemma is proven. □

Proof of the basic security theorem—that the unwinding conditions are sufficient—is now straightforward.

Theorem 1 A system that is secure by the unwinding conditions is secure by definition 1.
Proof The high water mark function is always dominated by some limit such that

$$\forall\, a : \mathit{class};\ t : \mathrm{seq}\, \mathit{action} \bullet$$
$$\exists\, B : (\Phi_{lim}\ \mathit{syspol}\ a) \bullet \delta_{uwn}(do^*(t), a) \subseteq B$$

The definition of Φ_{lim} implies that, given a, t and B such that the above holds, then $B \mapsto a \in \mathit{syspol}$. But syspol is an aggregation policy, and therefore $\delta_{uwn}(s, a) \subseteq B$ implies that $\delta_{uwn}(t, a) \mapsto a \in \mathit{syspol}$, which implies by Lemma 3 that $\delta(t, a) \mapsto a \in \mathit{syspol}$. Thus we have

$$\forall\, a : \mathit{class};\ t : \mathrm{seq}\, \mathit{action} \bullet$$
$$\delta(t, a) \mapsto a \in \mathit{syspol}$$

That is, the system is secure. □

Example 13 Consider again policy `BankPol`. Figure 4 gives an algorithm for a possible implementation mechanism. The state includes counter variables `B1count` and `B2count`, with the obvious interpretation. Variables `B1hwm`, `B2hwm` and `Conshwm` implement the (approximate) high water mark function for classes `bank1`, `bank2` and `cons`, respectively. Their initial values are calculated according to the value of δ_{uwn} for the initial state. From Example 11, classes `bank1` and `bank2` map to single levels in the implementation lattice, and thus their values remain fixed. Class `cons` has a floating

```
var
  B1count, B2count       : integer;
  Conshwm, B1hwm, B2hwm  : level;

begin
B1count:= 0;       B2count:= 0;
B1hwm  := {bank1}; B2hwm:= {bank2};
Conshwm:= {cons};

while true do
  begin
  input next Command;
  case Command of
    B1cmd : B1count:= B1count+1;
    B2cmd : B2count:= B2count+1;
    B1req : if (Conshwm <= {cons,bank1}) then
              begin
              Conshwm:= Conshwm + {bank1};
              output B1count to consultant
              end;
    B2req : if (Conshwm <= {cons,bank2}) then
              begin
              Conshwm:= Conshwm + {bank2};
              output B2count to consultant
              end
  end
```

Figure 4: Enforcing Policy `BankPol`.

high water mark (`Conshwm`) which is initially set to level `cons`. The algorithm permits this to float to one of its limits based on which bank the consultant accesses.

To prove this system secure we must compute a view equivalence relation such that unwinding conditions C1 to C4 hold. This is often a trial and error process and the reader is referred to [11] for background on developing such relations. For this example, we associate a level from the implementation lattice with each state component. Associating or binding a level with a state component implies that the component is expected to hold information at that level, i.e., at that conglomerate class. Thus variables `B1count` and `B2count` are bound to levels {`bank1`} and {`bank2`}, respectively. The values

$l : level$	$s \approx_l s'$
$\varnothing$	*true*
$\{\texttt{cons}\}$	$\texttt{Conshwm} = \texttt{Conshwm}'$
$\{\texttt{bank1}\}$	$\texttt{B1count} = \texttt{B1count}'$
$\{\texttt{bank2}\}$	$\texttt{B2count} = \texttt{B2count}'$
$\{\texttt{cons}, \texttt{bank1}\}$	$\texttt{Conshwm} = \texttt{Conshwm}'$
$\{\texttt{cons}, \texttt{bank2}\}$	$\texttt{Conshwm} = \texttt{Conshwm}'$
$\{\texttt{cons}, \texttt{bank1}, \texttt{bank2}\}$	*true*

Table 5: View Equivalence between states s and s'.

of the high water mark variables must also be considered in the determination of $\approx_l$. Variables `B1hwm` and `B2hwm` are static and thus their bindings have no impact on the view equivalence, while `Conshwm` can change. We adopt the strategy that any change in `Conshwm` effects the views at every level that it could hold, i.e. $\{\texttt{cons}\}$, $\{\texttt{cons}, \texttt{bank1}\}$ and $\{\texttt{cons}, \texttt{bank2}\}$. If variables at state s are referenced as `B1count`, `B2count`, etc., while variables at state s' are referenced as `B1count`$'$, `B2count`$'$, then Table 5 defines a suitable view equivalence relation. Under this relation, the unwinding conditions C1 to C4 hold and thus the system is secure. $\triangle$

Example 14 Consider again Example 10. Figure 5 gives the implementation lattice and mapping for the (reflexive) policy `FlightPol` (class `fl` abbreviates `flight`). Given this implementation lattice the (unwound) high water mark function on classes is static at the value given by Φ_{lim}. The

$$\begin{array}{c} \{\texttt{lo}, \texttt{hi}, \texttt{fl}\} \\ \uparrow \\ \{\texttt{lo}, \texttt{fl}\} \end{array}$$

a	$\Phi_\perp\texttt{FlightPol}\ \{a\}$	$\Phi_{lim}\texttt{FlightPol}\ a$
$\{\texttt{lo}\}$	$\{\texttt{lo}, \texttt{fl}\}$	$\{\{\texttt{lo}, \texttt{fl}\}\}$
$\{\texttt{hi}\}$	$\{\texttt{lo}, \texttt{hi}, \texttt{fl}\}$	$\{\{\texttt{lo}, \texttt{hi}, \texttt{fl}\}\}$
$\{\texttt{fl}\}$	$\{\texttt{lo}, \texttt{fl}\}$	$\{\{\texttt{lo}, \texttt{hi}, \texttt{fl}\}\}$

Figure 5: Implementation of Policy `FlightPol`.

state of this system maintains information about the single flight. From the implementation lattice we see that there are two kinds of flight information, $\{\texttt{lo}, \texttt{fl}\}$ and $\{\texttt{lo}, \texttt{hi}, \texttt{fl}\}$. We thus require some way to relate the flight information with these levels. In Example 10, Table 2, this is achieved by introducing a second variable which indicates the level currently associated with the flight variable. Call the flight variable *FlVal* and its level variable *VarHwm*. Using the same strategy as Example 13 for calculating the view equivalence relation we have:

$$\begin{aligned} s \approx_l s' \quad \Leftrightarrow \quad & (VarHwm = VarHwm') \wedge \\ & ((VarHwm = l) \Rightarrow (FlVal = FlVal')) \end{aligned}$$

Under this relation the unwinding conditions are met and thus the system is secure.

This system uses a restricted form of high water mark on the flight variable—it may be raised only from below. In this way, it is not possible for a Trojan Horse operating at `hi` to signal a `lo` user by either raising the high water mark or not, on what is initially a `lo` variable. If we attempted to use the more general high water mark approach[24] where it could be raised from above, it would not be possible to prove security.

In this example we are using a (restricted form of) high water mark on a state variable to reflect the fact that it can be used to hold different classes of information. This is not the same as the already discussed notion of a high water mark on classes (i.e., δ and δ_{uwn}) which gives a high water mark on information sunk to a class of the environment. $\triangle$

The flow preserving mapping provides information that may assist in the implementation of the system. In the previous example (14) the implementation lattice defined two possible levels for flight information. The system specification must consider how any state variable designed to hold flight information relates to these levels. We chose a (restricted) high water mark approach. An alternative design might be to polyinstantiate, i.e. have multiple copies of, the variable for all levels of information it is expected to hold. In the case of `FlightPol`, there would be two versions of the flight variable: one for `lo` users and the other for `hi` users.

5 Classes of Flow Policy and their Models

5.1 Reflexive Flow Policies

If a flow policy is reflexive, then each class will have a single maximum in $\Phi_{lim} P\ a$, since $(A \mapsto c \in P) \wedge (B \mapsto c \in P)$ implies $A \cup B \mapsto c \in P$, and thus

$$\Phi_{lim} P \ a = \{(\bigcup\{ \ B : \mathbb{P}\, class \mid B \mapsto a \in P \ \})\}$$

Given this, the conditions on states and state transitions simplify. Each class is bound to a static high water mark as pictured in Figure 6. A reflexive

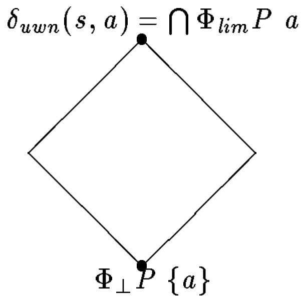

Figure 6: Static High Water Marks for Reflexive Policies

policy corresponds to a lattice policy plus pairs of bindings such that a user bound to singleton conglomerate $\{a\}$ may sink any level (conglomerate) under $\delta_{uwn}(s, a)$ and may source to any level over $\Phi_{\perp} P \ \{a\}$.

The security objectives for reflexive policies are similar to those for partial trust[2, 5]. A (user at) class a is trusted to handle information at any level between $\Phi_{\perp} P \ \{a\}$ and $\delta_{uwn}(s, a)$. From Example 10, a pilot (class `flight`) is allowed to read `hi` information and write to `lo`. However, the pilot cannot use the system to copy `hi` information to `lo`—cutting a section of `hi` text and pasting it into a `lo` window is not permitted. Of course the pilot could re-type, as `lo`, the `hi` information back into the system. But this is not a concern since binding a pilot to class `flight` is associating a degree of trust with that user in that he handles such information appropriately.

In examples 13 and 14 the implementation lattice and level bindings suggest some implementation detail: if we plan to use some (state) object to hold information at class a then in the implementation it may hold information at any level bounded by $\Phi_{\perp} P \ \{a\}$ and the limits on $\delta_{uwn}(s, a)$. This is not unlike the high water mark approaches with upper limits[24, 25]. Examples 13 and 14 use high water mark-based implementations. Polyinstantiation-based implementations are also possible, where copies of the variable should be made for the levels bounded between $\Phi_{\perp} P \ \{a\}$ and $\delta_{uwn}(s, a)$.

Reflexive policies provide a convenient abstraction of these security models. It is possible to reason about the nature of the policies in an abstract manner without resorting to a specific system model.

5.2 Quasi Ordered Flow Policies

If the policy to be enforced forms a quasi ordered set then the flow preserving mapping simplifies further to a single point in the powerset lattice.

$$\Phi_{\perp} P\ \{a\} = \bigcup \{\ B : \mathbb{P}\, class \mid B \mapsto a \in P\ \}$$
$$\Phi_{lim} P\ a = \{(\Phi_{\perp} P\ a)\}$$

This corresponds to the Birkhoff powerset mapping[3], which maps a quasi ordered set to a partial order. In this case, the high water mark model is similar to the traditional lattice-based models such as [10].

6 Policy Specification: Chinese Wall

In [4] Brewer and Nash describe a model for enforcing Chinese Wall policies. Such policies are a good example of aggregation, and this section develops a general policy framework for specifying Chinese Walls.

A stock market database holds confidential information on different organizations. A unique class is used to represent the information of each organization. Organizations are classified in terms of conflict (of interest) sets. Two organizations appear in the same conflict set if they have conflicting business interests. The aim of our Chinese Wall policy is to prevent a consultant from gaining information on more than one organization within any conflict set.

A conflict set is a component drawn from the set

$$\textit{conflict-set} == \mathbb{P}\, class$$

For example, the conflict set *Banks* : *conflict-set* might give the set of banking organizations, i.e., the set of classes that represent information from those banks. A number of (possibly intersecting) conflict sets may be defined over a collection of organizations.

The classes in a conflict set must be disjoint: one bank may not discover anything about any other. A consultant may sink information only about one class within a conflict set. Define

$$
\begin{array}{|l}
\textit{conflict-pol} : (\textit{conflict-set} \times \textit{class}) \rightarrow \textit{policy} \\
\hline
\forall\, \textit{cset} : \textit{conflict-set};\ \textit{cons} : \textit{class} \bullet \\
\qquad \textit{conflict-pol}(\textit{cset}, \textit{cons}) = \bigcup\{\textit{org} : \textit{cset} \bullet (\{\textit{org}\} \text{ aggflow } \textit{cons})\}
\end{array}
$$

The policy *conflict-pol*(*Banks*, *cons*) specifies that a consultant, class *cons*, may sink information about only one organization within the conflict set *Banks*.

A consultant may consult for a number of organizations so long as he obeys the conflict policy for each conflict set, i.e., he may consult for a number of organizations so long as there is no conflict of interest. Define

$$
\begin{array}{|l}
\textit{China} : (\mathbb{P}\, \textit{conflict-set}) \times \textit{class} \rightarrow \textit{policy} \\
\hline
\forall\, A : \mathbb{P}\, \textit{conflict-set};\ \textit{cons} : \textit{class} \bullet \\
\qquad \textit{China}(A, \textit{cons}) = \bigoplus\{\textit{cset} : A \bullet \textit{conflict-pol}(\textit{cset}, \textit{cons})\}
\end{array}
$$

If A specifies a set of conflict sets, and *cons* a consultant's class, then the policy *China*(A, *cons*) describes the Chinese Wall policy. For example, given

$$
\begin{array}{|l}
\textit{Banks}, \textit{Oil}, \textit{Insurance} : \textit{conflict-set} \\
\textit{cons} : \textit{class}
\end{array}
$$

the flow policy is specified as *China*({*Banks*, *Oil*, *Insurance*}, *cons*).

The granularity of aggregation in a conglomerate flow policy extends to information classes, not system components (files, users, etc). Thus in the Chinese Wall example, if there are a number of consultants (as there would normally be), the policy views them as a single entity such that when one consultant accesses a particular bank, all other consultants are prevented from accessing other banks. An alternative Chinese Wall policy that reflects the fact that there may be different classes of consultant information is parameterized by a set of consultant classes

$$
\begin{array}{|l}
\textit{China2} : (\mathbb{P}\, \textit{conflict-set}) \times (\mathbb{P}\, \textit{class}) \rightarrow \textit{policy} \\
\hline
\forall\, A : \mathbb{P}\, \textit{conflict-set};\ \textit{cons-set} : \mathbb{P}\, \textit{class} \bullet \\
\qquad \textit{China2}(A, \textit{cons-set}) = \bigoplus\{c : \textit{cons-set} \bullet \textit{China}(A, c)\}
\end{array}
$$

The policy *China2*(A, *cons-set*), defines a Chinese Wall policy for the consultant classes defined in *cons-set* over the organizations in A. The classes of *cons-set* might be used to represent individual consultants, or alternatively groups of consultants liable to collude.

It is clear that for any practical policy, the number of possible information classes will be very large. It would not be practical to *directly* implement the implementation lattice. Implementations must be sought that preserve (but do not necessarily build) the lattice policy.

At first, it might appear that specifying conglomerate policies would be a laborious affair, given that all classes of information must be enumerated. However, we view it as an advantage. We have a framework in which the model designer is obliged to precisely state the exact meaning of policies in the context of the system. The policy relations and operators (Section 2) allow comparison and combination of these complex policies.

7 Conclusions

Conglomerate flow policies are stateless specifications of security in a system. They describe the different classes of information that can exist in a system and how they may propagate. The operators for composition and comparison allow the construction of complex information flow policies. These policies provide a unified notation for reasoning about confidentiality, aggregation and separation. Conglomerate policies provide a convenient abstraction of security: we may reason about the nature of a policy without resorting to system details.

We propose a semantics for conglomerate policies based on noninterference, and unwind this definition for a class of useful policies—aggregation policies. The resulting conditions are not unlike the traditional unwinding conditions for noninterference. This is significant as there is quite a body of experience in building systems that enforce similar conditions.

Acknowledgements

This work has benefited greatly from discussions with the members of the 'Thursday Club' at Oxford, especially Jeremy Jacob and Simon Wiseman. Special thanks also go to Jeremy Jacob for help with fuzz. The anonymous referees gave many helpful suggestions and criticisms throughout the review process. Part of this work was funded by the Royal Signals and Radar Establishment while the author was at Cranfield IT Institute.

Appendix: The Z Notation

The Z notation is based on strongly typed set theory and first order logic. Much of the Z notation can be recognized from its set and logic origins. This appendix describes some of the possibly less obvious notation that appears in the paper.

Sets

A set may be defined in Z using set specification in comprehension. This is of the form $\{ D \mid P \bullet E \}$, where D gives declarations, P a predicate and E an expression. The components of $\{ D \mid P \bullet E \}$ are the values taken by expression E when the variables introduced by D take all possible values that make the predicate P true. For example, the set of squares of all even natural numbers is defined as $\{ n : \mathbb{N} \mid (n \bmod 2) = 0 \bullet n^2 \}$. When there is only one variable in the declaration and the expression consists of just that variable, then the expression may be dropped. For example, the set of all even numbers may be written as $\{ n : \mathbb{N} \mid (n \bmod 2) = 0 \}$. An equivalent way to define this set is as $\{ n : \mathbb{N} \bullet (2 * n) \}$—the predicate may be dropped if it equals true. Sets may also be defined in display form such as $\{\mathtt{lo}, \mathtt{hi}\}$.

Functions and Relations

In Z, relations and functions are represented as sets of pairs. A (binary) relation R, declared as having type $A \leftrightarrow B$, is a component of $\mathbb{P}(A \times B)$. For $a \in A$ and $b \in B$, then the pair (a, b) is written as $a \mapsto b$, and $a \mapsto b \in R$ means that a is related to b under relation R. If a relation name begins with an underscore, for example $\_R\_$ say, then R can be used as an infix relation. For example, with the infix relation $\_ \leq \_$ the expression $2 \leq 4$ is the infix sugared version of expression $2 \mapsto 4 \in \_ \leq \_$. Functions are treated as special forms of relations.

Declarations

A variable declaration in Z is of the form $V : E$ (or $V_1, \ldots V_n : E$), where V is an identifier and E an expression that defines a set. Variable V has type, the component type of E, and may take on any value from E. Local variables may be introduced by existential and universal quantification. Global variables are introduced by the syntax

$$
\begin{array}{|l}
\textit{Declarations} \\
\hline
\textit{Predicates}
\end{array}
$$

Where *Predicates* may further constrain variables declared in *Declarations*. Such global declarations may be made without the predicate part. The variables declared in *Declarations* may be further constrained by arbitrary predicates introduced later in the text (recall the definition of δ_{uwn} in Section 4 and its subsequent constraints defined in C1 to C4). The syntax

$$Ident == Expression$$

introduces a global constant *Ident* whose type and value equals that of *Expression*.

$\varnothing$	The empty set.
$\mathbb{P}\, A$	The powerset of A.
$A \setminus B$	Set A less the components in B.
$\bigcup A$	Generalized union.
$\mathrm{dom}\, R$	The domain of R.
$\mathrm{ran}\, R$	The range of R.
$A \rightarrow B$	Set of total functions from A to B.
$A \nrightarrow B$	Set of partial functions from A to B.
$A \leftrightarrow B$	Set of relations between A and B.
$\langle\rangle$	The empty sequence.
$\mathrm{seq}\, A$	Set of all sequences over A.
$s \frown t$	Catenation of sequences s and t.
$s \upharpoonright A$	Sequence s with components not in A removed.
$\#s$	The length of sequence s.

Figure 7: Some Z notation.

References

[1] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", Technical Report, MTR-2997, Mitre Corporation, March 1976.

[2] D.E. Bell, "Security Policy Modeling for the Next Generation Packet Switch", Proceedings 1988 Symposium on Security and Privacy, pages 212–216, IEEE Computer Society, Oakland, CA., May 1988.

[3] G. Birkhoff, "Lattice Theory", Volume XXV of American Mathematical Society Colloquium Publications, American Mathematical Society, Third Edition, 1967.

[4] D.F. Brewer and M.J. Nash, "The Chinese Wall Security Policy", Proceedings 1989 Symposium on Security and Privacy, pages 206–258, IEEE Computer Society, Oakland, CA., May 1989.

[5] M. Branstad et. al., "Trusted Mach Design Issues", in Proceedings Third Aerospace Computer Security Conference, December 1987.

[6] D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Models", In Proceedings 1987 Symposium on security and Privacy, IEEE Computer Society, Oakland, CA., April 1987

[7] D.E. Denning, "A Lattice Model of Secure Information Flow", Communications of the ACM, 19(5), pages 236–243, May 1976.

[8] D.E. Denning, "On the Derivation of Lattice Structured Information Flow Policies", Technical Report CSD TR180, Purdue University, 1976.

[9] D.E. Denning et. al., "A Multilevel Relational Data Model", In 1987 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, April, 1987.

[10] T. Fine, et. al., "Noninterference and Unwinding for LOCK", In Proceedings Computer Security Foundations Workshop II, IEEE Computer Society Press, Franconia N.H., June, 1989.

[11] T. Fine, "Constructively Using Noninterference", In 1989 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, May, 1989.

[12] S.N. Foley, "A Model for Secure Information Flow", In 1989 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, May, 1989.

[13] S.N. Foley, "Unifying Information Flow Policies", Report 90020, Royal Signals and Radar Establishment, October 1990.

[14] S.N. Foley, "A Taxonomy for Information Flow Policies and Models", In 1991 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, May, 1991

[15] S.N. Foley, "Separation of Duty Using High Water Marks", In Computer Security Foundations Workshop IV, IEEE Computer Society Press, Franconia N.H., June, 1991.

[16] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control", In IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, April, 1984.

[17] T.Y. Lin, "Chinese Wall Security Policy – An Aggressive Model", In Proceedings of the 5th Aerospace Symposium on Computer Security and Privacy, 1989.

[18] T.F. Lunt, "Aggregation and Inference: Facts and Fallacies", In 1988 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, April, 1988.

[19] J. McLean, "Security Models and Information Flow", In 1990 IEEE Symposium on Security and Privacy, pages 162–169, IEEE Computer Society, Oakland, May, 1989.

[20] C. Meadows, "Extending the Brewer-Nash Model to a Multi-level Context", In 1990 IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland, CA., May 1990.

[21] C. Meadows, "Aggregation Problems: A Position Paper", RADC Database Security Workshop, June 1990.

[22] C. Meadows, "Policies for Dynamic Upgrading", IFIP Workshop on Database Security, September 1990.

[23] J.M. Spivey, "The Z Notation", Prentice Hall, 1989.

[24] C. Weissman, "Security Controls in the ADEPT-50 Time Sharing System", In Proceedings of the 1969 AFIPS Fall Joint Computer Conference, pages 119–133, AFIPS Press, Arlington, Va, 1969.

[25] J.P.L. Woodward, "Exploiting the Dual Nature of Sensitivity Labels", In 1987 IEEE Symposium on Security and Privacy, Oakland CA., April 1987.

[26] A. Wood, "The SWORD Model of Multilevel Secure Databases", Report 90008, Royal Signals and Radar Establishment, U.K., June 1990.