A Policy Model for Secure Information Flow

Adedayo O. Adetoye and Atta Badii

School of Systems Engineering, University of Reading, Whiteknights, Berkshire, RG6 6AY, UK
a.o.adetoye@reading.ac.uk, atta.badii@reading.ac.uk

Abstract. When a computer program requires legitimate access to confidential data, the question arises whether such a program may illegally reveal sensitive information. This paper proposes a policy model to specify what information flow is permitted in a computational system. The security definition, which is based on a general notion of information lattices, allows various representations of information to be used in the enforcement of secure information flow in deterministic or nondeterministic systems. A flexible semantics-based analysis technique is presented, which uses the input-output relational model induced by an attacker's observational power, to compute the information released by the computational system. An illustrative attacker model demonstrates the use of the technique to develop a termination-sensitive analysis. The technique allows the development of various information flow analyses, parametrised by the attacker's observational power, which can be used to enforce *what* declassification policies.

1 Introduction

The problem of *secure information flow* arises when a computer program must be granted legitimate access to confidential data. When such a program, which might have access to a network or that might otherwise be able to transmit confidential information to unauthorised observers, is executed, we want assurances that only the information that we wish to reveal is released. An *information flow policy* expresses our security concern about the information release that we consider as safe. This leads to the question of how to specify *what* information release is safe. The traditional approach to the specification of information release, or rather, the lack of it, is through the *noninterference* requirement [7]. Noninterference prevents *any* flow of secret information to public areas in a multi-level security system, where information must not flow from *high* to *low*. Thus, noninterference is very restrictive and its usefulness in general practice has been argued [13]. In practice, for example, during encryption, authentication, or statistical analysis, we often want to release *some level* of information. This requires a more general policy model by which we can specify what is the safe level of information to be released. This paper proposes a lattice model to capture this property.

In [16], a taxonomy of declassification mechanisms is introduced based on *what*, *where*, *when* and by *whom* information is released. This paper is concerned about the *what* dimension of information flow, where we want to express the property that the information released by a system does not exceed certain allowed limits. Based on this observation, a *definition of security* is given, which captures the idea that a given information flow is safe.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 1–17, 2009.
© Springer-Verlag Berlin Heidelberg 2009

1.1 Contributions

This paper contributes to the theory of secure information flow through a systematic study of lattices of information as a tool for the enforcement of *what* declassification policies. Although lattice-based approaches are often used in language-based security [14], these lattices are usually of security classes in a multi-level security system, rather than lattices of information. We demonstrate that various *representations* of information such as partial equivalence relations, families of sets, information-theoretic characterisation, and closure operators fit into the lattice model of information, unifying the various definitions under the lattice model. This means that the same partial order-based enforcement technique can be applied to all the representations.

Another contribution to the theory is an input-output *relation model*, presented as a primitive for the semantic analysis of information flow. A systematic approach to deriving the relational model from the operational semantics, which is parametric to a chosen attacker's observational power, is presented. The relational model accounts well for information flow due to nontermination, and the specific termination-sensitive analysis presented demonstrates the correct analysis of diverging programs by using the relational model.

1.2 Plan of the Paper

In Section 2 the lattice model of information is motivated, and a security definition is given which uses the lattice model to enforce *what* declassification policies. Section 3 introduces the relational model primitive as a tool for studying information flow in models of deterministic or nondeterministic systems. Section 4 uses the relational model to develop a representation of information, based on PERs, for the analysis of deterministic system models. A language-based analysis technique is presented for *While* programs with outputs to illustrate how to derive the relational model under a given attacker model in a language-based setting. Similarly to Section 4, Section 5 applies the relational model technique to develop a representation of information, based on families of sets, which captures the information that the attacker may gain when the system can be run repeatedly under fixed inputs. An extension of the *While* language with a nondeterministic construct shows the use of this information representation for information flow analysis in a nondeterministic language setting. We compare our approach with related works in Section 6. Section 7 concludes the paper.

2 Secure Information Flow

The concept of secure information flow suggests an understanding of the notions of information and information flow. A fundamental property of information is the intuitive notion of *information levels*, where we say that one piece of information is *greater* or *more informative* than another. This suggests an ordering of information, which we shall exploit in our information model and security definition. For this reason we shall model information as *lattices*, where the associated *partial order* captures the notion of information levels.

Definition 1 (Information and Information Flow). *We define* information *as elements of a complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$, where the associated partial order $\sqsubseteq$ models the relative degree of informativeness of the elements of $\mathcal{I}$, and the join operation $\sqcup$ models the combination of information.*

Information flow *with respect to the lattice $\mathcal{I}$ is defined as an* extensive *and* monotone *self map on $\mathcal{I}$. Define $\mathcal{F}lows \triangleq \{ f : \mathcal{I} \to \mathcal{I} \mid f \text{ is extensive and monotone} \}$ to be the set of all information flows on $\mathcal{I}$.*

The lattice join operation, which models information combination, is idempotent, commutative, and associative. These properties agree with natural intuitions about information, since idempotency says that the combination of a piece of information with itself should yield the same information [10]. Similarly, the commutativity and associativity properties respectively agree with the intuitions that the order and grouping of information combination should not matter to the end result. Furthermore, for any $s, s' \in \mathcal{I}$, the lattice property, $s \sqsubseteq s'$ iff $s \sqcup s' = s'$, agrees with the idea that the combination of a lesser information with a greater one yields the greater information - where $s \sqsubseteq s'$ is interpreted to mean that the information s is less than or at most equal to s'.

The notion of *information flow* models how the knowledge of an attacker changes due to information release. For any initial knowledge $s \in \mathcal{I}$ that the attacker might have, $f(s)$ represents the final knowledge of the attacker due to the information flow $f \in \mathcal{F}lows$ that the attacker receives. The extensivity property of f (that is, $\forall s \in \mathcal{I}, s \sqsubseteq f(s)$) intuitively means that an attacker's knowledge may only increase by gaining information, and the monotonicity (that is, $\forall s, s' \in \mathcal{I}, s \sqsubseteq s' \implies f(s) \sqsubseteq f(s')$) means that the greater the initial knowledge of the attacker before information release the greater the knowledge afterwards.

Using this ordered structure of information, we can now define what it means for a system to have secure information flow with respect to what information the system releases and a given information flow policy[1].

Definition 2 (*What* Policies and Security). *Given the lattice $\langle \mathcal{I}, \sqsubseteq \rangle$ of information and the set $\mathcal{F}lows$ of information flows over this lattice. An information flow policy $\mathscr{P}$ with respect to the lattice $\mathcal{I}$ is a* subset *of $\mathcal{F}lows$ which specifies the permitted information flows. An information flow $f \in \mathcal{F}lows$ is said to be* permitted *or* allowed *by a policy $\mathscr{P} \subseteq \mathcal{F}lows$ iff there exists a flow function $f' \in \mathscr{P}$ such that $f \sqsubseteq f'$.*

Let P be a program, modelling a system. Furthermore, let $[\![P]\!]^{\mathcal{I}} \subseteq \mathcal{F}lows$ be the information flow property *of the system modelled by P, which describes the information flows caused by this system. The system* satisfies*, and is said to be* secure *with respect to a policy $\mathscr{P} \subseteq \mathcal{F}lows$ iff for all $f \in [\![P]\!]^{\mathcal{I}}$ there exists $f' \in \mathscr{P}$ such that $f \sqsubseteq f'$.*

The order $f \sqsubseteq f'$ between flow functions is the usual pointwise ordering of functions induced by the partial order $\sqsubseteq$ on the lattice $\mathcal{I}$. This partial order regulates the level of information that we allow a system to release. Intuitively, this definition says that the program P (or the system it models) is secure (with respect to the policy $\mathscr{P}$) only if every flow $f \in [\![P]\!]^{\mathcal{I}}$ that is caused by the system is permitted by the policy ($\exists f' \in \mathscr{P}$ such that $f \sqsubseteq f'$). This extensional view of policy enforcement abstractly describes, in terms of the information lattice order, what information flows are permitted in the system.

[1] We shall sometimes refer to an information flow policy simply as *policy* or *security policy*.

Since $\mathcal{I}$ is a complete lattice, it is easy to show that $\mathcal{F}lows$ also forms a complete lattice under the pointwise function ordering. In particular, the noninterference policy, which prevents any flow of information to the attacker may be modelled by the *identity* map ($id_{\mathcal{I}}$) on the lattice $\mathcal{I}$. This means that a system that satisfies the *noninterference policy* $\{id_{\mathcal{I}}\}$ has the information flow property that regardless of the attacker's previous level of knowledge, the final knowledge remains unchanged and the attacker is unable to benefit information by observing this noninterfering system. The baseline nature of the noninterference policy model $\{id_{\mathcal{I}}\}$ is established by the fact that $id_{\mathcal{I}}$ is the least element of the lattice $\mathcal{F}lows$ of information flows. However, it is clear that other less restrictive policies than $\{id_{\mathcal{I}}\}$ may be specified. Let us examine some example policy patterns under the lattice model of information flow.

2.1 Information Flow Policy Patterns

We examine in this section, *policy patterns*, other than the noninterference pattern, which allow deliberate, but controlled, release of information.

We may wish to have partial (but unconditional) release of information $s' \in \mathcal{I}$ but not more in a system. The required information flow property is captured by the policy $\mathscr{P} = \{f | \forall s \in \mathcal{I}, f(s) = s' \sqcup s\}$, which allows the attacker to combine its knowledge s with the declassified information s', but the attacker may not learn more than this by observing the system which is secure with respect to $\mathscr{P}$. An example of this scenario arises during password authentication, where we wish to release (unconditionally) the information about the equality or not of the stored password and the user-supplied password.

Another scheme is the *conditional release* pattern, where information (s') is released based on having some initial knowledge (s''). This is modelled by the policy $\{f\}$ where $\forall s \in \mathcal{I}, f(s) = s' \sqcup s$ if $s'' \sqsubseteq s$, and $f(s) = s$ otherwise. Under this scheme, the attacker gains some information on the condition that the attacker has at least a given initial information s''. A scenario where such a policy is needed is during decryption in a symmetric key system, where the plaintext may be learnt (the knowledge s') only when the decryption key is known (the knowledge s'').

Another pattern, called *disjunctive flow policy* - after the disjunctive flow pattern of [16], is $\{f, f' | f \nsubseteq f', f' \nsubseteq f\} \subseteq \mathcal{F}lows$. This policy permits at most one of f or f' to be released but not *both* at the same time. It is clear that the notion of disjunctive information flow is only meaningful for incomparable information and information flows, because whenever two information are comparable then the greater already contains the lesser information. An information flow $f'' \in \mathcal{F}lows$ is permitted by the disjunctive policy $\{f, f' | f \nsubseteq f', f' \nsubseteq f\}$, when f'' is smaller than or equal to at most one of f and f' - since f and f' are incomparable. Also, a flow $f'' \sqsupseteq f \sqcup f'$, which contains both f and f' is not permitted since there is no such $f_1 \in \{f, f' | f \nsubseteq f', f' \nsubseteq f\}$ for which $f'' \sqsubseteq f_1$.

In Definition 2 we defined the security property of a system, with respect to a policy, in terms of the system's *information flow property*. In the next section we shall show a way to derive the information flow property of a system from a relation which describes how the system transforms its inputs to publicly observable outputs.

3 The Relational System Model

The question that we want to answer is whether a system that processes confidential data is secure with respect to a given security policy. So, using a suitable representation of information, we want to check whether the information released conforms to our policy requirement. The particular choice of *representation* of information may depend on the model of the system being analysed or the kind of information that we are interested in modelling. In this paper we shall consider information flow analysis under *deterministic* and *nondeterministic* system models using *qualitative* representations of information and show that the representations fit into the lattice model of information. We note that it is possible to consider *quantitative* representations of information, such as *entropy* and other information-theoretic measures, under the lattice model of information because the measures, which are numbers, are naturally ordered.

We shall model a computational system, by an *input-output relation*: $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$, where the system accepts inputs taken from the set $\mathbf{\Sigma}$ and produces public outputs (with respect to an attacker model) in the set $\mathcal{V}$. The relation S models what the attacker observes given the supplied input. Depending on the attacker model, the system model may be *deterministic*, in which case S is a *function*. More generally, however, the model S of the system is said to be *nondeterministic* (with respect to the attacker's view) when the attacker's observation is not necessarily unique for a given input. In the sequel, we shall sometimes refer to the system modelled by the relation S simply as "system S".

Definition 3 (The Relational Model). *The input-output* relational model *of a system is defined as a relation $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$, over the set $\mathbf{\Sigma}$ of the system's inputs and the set $\mathcal{V}$ of observable outputs, where for all input $\sigma \in \mathbf{\Sigma}$ and possible output $v \in \mathcal{V}$, $\sigma\, S\, v$ holds iff the system can produce the output v when supplied with the input σ.*

The inverse image of the relation S at $v \in \mathcal{V}$ is denoted by $S^{-1}(v) \triangleq \{\sigma \in \mathbf{\Sigma} \mid \sigma\, S\, v\}$. When the relational model f is a *function*, we write $f : \mathbf{\Sigma} \to \mathcal{V}$ and for any $\sigma \in \mathbf{\Sigma}$, $f(\sigma)$ stands for the unique output observed when the input σ is supplied. We refer to f as the *functional model* of the system.

4 Analysis for Deterministic System Models

We shall use *Partial Equivalence Relations*[2] (PERs) as a qualitative representation of information for the deterministic system model. We start by motivating the use of PERs for information representation in this setting. Since the deterministic system model is a (total) function $g : \mathbf{\Sigma} \to \mathcal{V}$, where for any input $\sigma \in \mathbf{\Sigma}$ supplied to the system, the attacker observes $g(\sigma) \in \mathcal{V}$, the knowledge gained by the attacker can be described by the ability to *distinguish* which inputs might have been supplied based on the observed output. Thus, given the observed output $v \in \mathcal{V}$ of the system, the attacker knows that the input to the system has been taken from the set $g^{-1}(v) \subseteq \mathbf{\Sigma}$. However, based on this observation the attacker cannot *distinguish* between the inputs σ and σ'

[2] A partial equivalence relation is a *symmetric* and *transitive* binary relation. If in addition the relation is also *reflexive*, then it is an *equivalence relation*.

if $\sigma, \sigma' \in g^{-1}(v)$. More generally, we can model all the input pairs that the attacker cannot distinguish by the *kernel* κ_g of g, which is the *equivalence relation* given by $\forall \sigma, \sigma' \in \mathbf{\Sigma}, \sigma \; \kappa_g \; \sigma' \iff g(\sigma) = g(\sigma')$. Thus, κ_g describes the *information* (or more directly, the *ignorance*) of the attacker based on the *indistinguishability* of input pairs after observing the system's outputs. An input pair is *distinguishable* by the attacker if the pair is *not related* by the equivalence relation κ_g.

We can generalise this idea a bit further from equivalence relations to PERs to obtain certain expressive powers. Similarly to equivalence relations, we say that a PER cannot distinguish a pair if the pair is *related* by that PER, but any value that is not in the domain of definition of the PER is considered as *not possible* and therefore distinguishable. For example, we can represent the knowledge of parity of an integer secret via the equivalence relation Par defined as $\forall n, m \in \mathbb{Z}, n \,\text{Par}\, m$ iff $n \mod 2 = m \mod 2$, which distinguishes a pair of integers with different parity. However, the PER Par+ over integers, defined as $\forall n, m \in \mathbb{Z}, n \,(\text{Par+})\, m$ iff $n \,\text{Par}\, m$ and $n, m \geq 0$, represents the knowledge of *parity* and *sign* - since it only relates natural numbers (negative integers are not possible). The use of equivalence relations, and PERs in general, to describe the security property of programs is not new [11,15]. In this paper we are interested in the lattice properties for the enforcement of secure information flow.

Definition 4. *Let $\mathbf{\Sigma}$ be a set. Define PER$(\mathbf{\Sigma})$ to be the set of all PERs over the set $\mathbf{\Sigma}$ and define the information order relation $\sqsubseteq$ on PERs such that for any $R, R' \in$ PER$(\mathbf{\Sigma})$, $R \sqsubseteq R'$ iff for all $\sigma, \sigma' \in \mathbf{\Sigma}$, $\sigma \; R' \; \sigma'$ implies $\sigma \; R \; \sigma'$. The associated information combination, or join operation $\sqcup$, on PER$(\mathbf{\Sigma})$ is defined as $\sigma \; (R \sqcup R') \; \sigma'$ iff $\sigma \; R \; \sigma'$ and $\sigma \; R' \; \sigma'$. More generally, for any subset $\mathcal{R} \subseteq$ PER$(\mathbf{\Sigma})$ let the join of $\mathcal{R}$ be the PER $\bigsqcup \mathcal{R}$, defined for all $\sigma, \sigma' \in \mathbf{\Sigma}$ as $\sigma \bigsqcup \mathcal{R} \; \sigma'$ iff $\forall R \in \mathcal{R}, \sigma \; R \; \sigma'$.*

We know from the definition of $\sqsubseteq$ that $R \sqsubseteq R'$ means that R' can distinguish whatever R can, and thus R' contains more information than R. For example, Par $\sqsubseteq$ Par+ agrees with the intuition of the relative information content of these two PERs. The ordering of PERs by their information content forms a complete lattice of information.

Theorem 1. *The partially ordered set $\langle$PER$(\mathbf{\Sigma}), \sqsubseteq, \sqcup\rangle$ is a complete lattice.*

Definition 5 (Deterministic Information flow). *Let $\mathcal{I} =$ PER$(\mathbf{\Sigma})$ be the lattice of information for a system S whose relational model is given by a function $g_S : \mathbf{\Sigma} \to \mathcal{V}$. The information flow property of this system may be defined as $[\![S]\!]^{\mathcal{I}} = \{f \mid \forall R \in$ PER$(\mathbf{\Sigma}), f(R) = R \sqcup \kappa_{g_S}\}$, where κ_{g_S} is the kernel of the function g_S.*

4.1 Language-Based Analysis

In this section, we shall demonstrate the application of the analysis approach presented above in a language-based setting. We shall use the core deterministic imperative *While* language of Fig. 1, which has outputs, as our basis. The language is similar to the language of [8], and its operational semantics is fairly standard.

While expressions may be boolean-valued (with values taken from $\mathbb{B} \triangleq \{\mathbf{tt}, \mathbf{ff}\}$), or integer-valued (taken from $\mathbb{Z}$). Program states, are maps from variables to values. The evaluation of the expression e at the state σ is summarised as $\sigma(e)$. Expression

$$c ::= \texttt{skip} \mid z := e \mid \texttt{write}\, e \mid c; c \mid \texttt{if}\,(b)\,\texttt{then}\, c\,\texttt{else}\, c \mid \texttt{while}\,(b)\,\texttt{do}\, c$$

Fig. 1. The *While* Language

evaluations are performed atomically and have no side-effect on state. Program actions, ranged over by a, can either be internal ε, which is not observable ordinarily, or output (via a *write* command), where the expression value can be observed. The operational semantics is presented via transition relations between expression configurations ($\langle e, \sigma\rangle \xrightarrow{\varepsilon} \langle \sigma(e), \sigma\rangle$) and command configurations ($\langle c, \sigma\rangle \xrightarrow{a} \langle c', \sigma'\rangle$). A special, terminal command configuration $\langle \cdot, \sigma\rangle$ indicates the termination of a command in state σ. The operational semantics is shown in Fig. 2.

$$\langle \texttt{skip}, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma\rangle \qquad \langle z := e, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma[z \mapsto \sigma(e)]\rangle \qquad \langle \texttt{write}\, e, \sigma\rangle \xrightarrow{\sigma(e)} \langle \cdot, \sigma\rangle$$

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{a} \langle \cdot, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{a} \langle c_2, \sigma'\rangle} \qquad \frac{\langle c_1, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{a} \langle c_1'; c_2, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{tt}, \sigma\rangle \quad \langle c_1, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle}{\langle \texttt{if}\,(b)\,\texttt{then}\, c_1\,\texttt{else}\, c_2, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle} \qquad \frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{ff}, \sigma\rangle \quad \langle c_2, \sigma\rangle \xrightarrow{a} \langle c_2', \sigma'\rangle}{\langle \texttt{if}\,(b)\,\texttt{then}\, c_1\,\texttt{else}\, c_2, \sigma\rangle \xrightarrow{a} \langle c_2', \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{ff}, \sigma\rangle}{\langle \texttt{while}\,(b)\,\texttt{do}\, c, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{tt}, \sigma\rangle \quad \langle c, \sigma\rangle \xrightarrow{a} \langle c', \sigma'\rangle}{\langle \texttt{while}\,(b)\,\texttt{do}\, c, \sigma\rangle \xrightarrow{a} \langle c'; \texttt{while}\,(b)\,\texttt{do}\, c, \sigma'\rangle}$$

Fig. 2. Operational Semantics of *While*

4.2 The Attacker's Observational Power

Let $\mathbf{\Sigma}$ be the set of all states of a program P. The trace of P, starting from the state $\sigma \in \mathbf{\Sigma}$, is denoted by $t_{\langle P,\sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a_0} \langle P_1, \sigma_1\rangle \xrightarrow{a_1} \cdots$, according to the operational semantics. A trace of P at the state σ is said to *terminate* if there is a natural number n such that $t_{\langle P,\sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} \langle \cdot, \sigma'\rangle$, otherwise, the trace is said to be *nonterminating* and P *diverges* at σ.

We introduce the notion of an attacker's *observational power* (*obs*) as a map from program traces to what the attacker observes. The relative powers of two attackers A and A', modelled respectively by the observational powers obs_A and $obs_{A'}$, may be compared under the proposed model, where we say that the attacker A' is more powerful than the attacker A if there exists a function f such that $obs_A = f \circ obs_{A'}$. A more powerful attacker will gain at least as much information as a less powerful attacker. In this paper we shall consider an attacker model, whose observational power is the function $obs(\cdot)$, which is able to observe only outputs generated by *write* statements and is also able to observe the termination or not of a program. The latter assumption about the attacker appears to be strong, and is usually not modelled in language-based

security. However, given the source code of a program, an attacker may be able to determine when a given program trace will not terminate without actually observing it. Modelling the ability to "observe" nontermination is also important as it may be possible to leak arbitrary amount of information via nontermination channels [2].

Definition 6 (The Semantic Attacker). *Let Σ be the set of all states of a* While *program P and let $\xrightarrow{\varepsilon}{}^{*}$ be the reflexive, transitive closure of the transition relation $\xrightarrow{\varepsilon}$. Furthermore, let $t_{\langle P,\sigma\rangle} = \langle P,\sigma\rangle \xrightarrow{\varepsilon}{}^{*} \langle P',\sigma'\rangle \xrightarrow{a_1} \langle P_1,\sigma_1\rangle \xrightarrow{\varepsilon}{}^{*} \langle P'_1,\sigma'_1\rangle \xrightarrow{a_2} \cdots$ be a canonical representation of the trace of P at $\sigma \in \Sigma$, where for all i, $a_i \neq \varepsilon$. Define the semantic attacker's observation of this trace as*

$$obs(t_{\langle P,\sigma\rangle}) \triangleq \begin{cases} \langle a_1, a_2, \cdots, \uparrow\rangle & \text{if } P \text{ diverges at } \sigma \\ \langle a_1, a_2, \cdots, \downarrow\rangle & \text{otherwise.} \end{cases}$$

The set of all possible observations that the attacker can make is given by $\mathcal{V} = \{obs(t_{\langle P,\sigma\rangle}) \mid \sigma \in \Sigma\}$. The functional model induced by this attacker's observational power is thus given by $g_P : \Sigma \to \mathcal{V}$, defined as $g_P(\sigma) = obs(t_{\langle P,\sigma\rangle})$ which maps the supplied input to the observed output of P under the attacker model.

The information gained by the semantic attacker from P is thus given by the equivalence relation $\lfloor T_P \rfloor$ over states, defined such that for any pair of states $\sigma, \sigma' \in \Sigma$, $\sigma \lfloor T_P \rfloor \sigma'$ iff $obs(t_{\langle P,\sigma\rangle}) = obs(t_{\langle P,\sigma'\rangle})$ - the kernel of g_P. The information flow property of P, over the lattice $\mathcal{I} = PER(\Sigma)$ may therefore be defined as $[\![P]\!]^{\mathcal{I}} = \{f \mid \forall R \in PER(\Sigma), f(R) = R \sqcup \lfloor T_P \rfloor\}$.

The definition of $obs(\cdot)$ formalises the idea that the semantic attacker cannot observe $\xrightarrow{\varepsilon}$ transitions. For nonterminating traces, the token $\uparrow$ is introduced which, in addition to the sequence of output tokens observed on the trace, signals the divergence of the program. Similarly, the token $\downarrow$ identifies a terminating trace. Although the operational semantics definition does not have a direct notion of nontermination, $obs(\cdot)$ can account for it since it is defined over the trace.

To illustrate the definition's termination properties, let *loop* = `while (tt) do skip`, and consider the programs P_1 = `if (h) then skip else` *loop* and P_2 = `write`h; *loop*. Both P_1 and P_2 insecurely reveal the boolean secret h. The analysis shows this because $\lfloor T_{P_1} \rfloor = \lfloor T_{P_2} \rfloor$ is the identity equivalence relation on h. As the analysis of P_2 demonstrates, we can easily show that by appending a trailing *loop* to any program P that always terminates, as in P' = P; *loop*, the information released is the same because $\lfloor T_P \rfloor = \lfloor T_{P'} \rfloor$. However, the analysis of P_3 = *loop*; `write` h shows that P_3 is safe because, as the semantics shows: $\langle P_3,\sigma\rangle \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \langle P_3,\sigma\rangle \xrightarrow{\varepsilon} \cdots$, the trailing `write` h is never executed, and hence $\lfloor T_{P_3} \rfloor$ relates all states, revealing no information.

In the next section we shall demonstrate the use of *what* policies for the enforcement of secure information flow in *While* programs.

4.3 Policies For Encryption

Encryption is an important security primitive which is used widely as a security foundation in many systems. However, in order to protect the secrecy of sensitive data used

during encryption, we require policies that permit encryption to be used safely. Noninterference policies cannot be used for encryption since the resulting (public) cyphertext in an encryption scheme will depend on the supplied plaintext and key which are considered secret. Thus, there remains the problem of the specification of policies that allow safe use of encryption in programs. We demonstrate in this section how our approach can be used to specify and enforce policies that allow encryption to be used safely.

We start by considering an encryption function $\mathcal{E}_1 : K \times M \to C$, which accepts a key chosen from the set K of keys and message or plaintext chosen from the set M, and produces a cyphertext in the set C. Now suppose that $\mathcal{E}_1$ is considered secure and that its implementation, which we shall denote by the expression $\textbf{\textit{enc}}(k, m)$, is correct, so that under any state $\sigma(\textbf{\textit{enc}}(k, m)) = \mathcal{E}_1(\sigma(k), \sigma(m))$. Therefore, we allow the attacker to observe the cyphertext $\textbf{\textit{enc}}(k, m)$ for any choice $k \in K$ of key and plaintext $m \in M$ values. Hence, we can define an equivalence relation $\lfloor \mathcal{E}_1 \rfloor$ which captures this intentional information release, where $\lfloor \mathcal{E}_1 \rfloor$ relates every pair of states σ and σ', where $\mathcal{E}_1(\sigma(k), \sigma(m)) = \mathcal{E}_1(\sigma'(k), \sigma'(m))$. The resulting information flow policy $\mathscr{P}_{\mathcal{E}_1} = \{f \mid R \in \text{PER}(\mathbf{\Sigma}), f(R) = R \sqcup \lfloor \mathcal{E}_1 \rfloor\}$ allows the attacker to observe the ciphertext generated by a correct implementation of the encryption function. Firstly, since the implementation ***enc*** is secure, it satisfies the policy $\mathscr{P}_{\mathcal{E}_1}$.

Now consider a secure (and insecure) data backup scenario (adapted from [1]) as shown in the program listings of Fig. 3. The LHS program P_1 securely releases the encrypted data (*ctxt*) to a public output channel after encrypting the data (*data*) with the key (k). However, the RHS implementation P_2 is insecure because the programmer releases the plaintext data instead of the ciphertext. The analysis detects that the RHS program violates the policy because $\lfloor T_{P_2} \rfloor \not\in \lfloor \mathcal{E}_1 \rfloor$ as required by $\mathscr{P}_{\mathcal{E}_1}$ - unless the encryption function by definition reveals the encrypted data, which violates initial assumption that it is *secure*. Thus, the analysis detects this flaw. This would be useful, for example, to a programmer who can avoid such programming error by checking his or her implementation against the desired policy.

The reason why the noninterference policy cannot be used for encryption lies in the fact that noninterference prohibits any sort of variation in the observed output from being induced by a variation in the secret input to the encryption function. However, one of the reasons why encryption is widely used as a security primitive is the fact that the security lies in the ability to protect secret data even when the encryption function is known. Thus, a good encryption function is already designed so that it is not easily invertible into its constituent arguments, although a variation in its input would cause a variation in its output for the function to be useful. The safe input-to-output variation caused by the *definition* of the encryption function is captured by the equivalence relation $\lfloor \mathcal{E}_1 \rfloor$ in the example above, which allows only the output variations due to the definition of the encryption function to be observed by the attacker.

$ctxt := \textbf{\textit{enc}}(k, data);$	$ctxt := \textbf{\textit{enc}}(k, data);$
write *ctxt*;	**write** *data*;

Fig. 3. Secure versus Insecure Data Backup

Nondeterministic Encryption. In nondeterministic encryption, such as *cipher-block chaining* encryption mode, an *initialisation vector* (iv) is used along with the key and plaintext such that if a different iv is used, a different ciphertext is generated under the same key and plaintext pair. The term "nondeterministic" refers to the fact that the implementation of such encryption algorithms generally have the property that encrypting the same plaintext several times using the same key would yield different ciphertexts. Let the function $\mathcal{E}_2 : IV \times K \times M \to C$ denote such an encryption scheme, where IV is the set of initialisation vectors. We shall represent by the expression $\boldsymbol{enc}_*(iv, k, m)$ a correct implementation of $\mathcal{E}_2$. Similarly to the encryption policy above, we define $\lfloor \mathcal{E}_2 \rfloor$ as the equivalence relation which relates all states which evaluate $\boldsymbol{enc}_*(iv, k, m)$ to the same ciphertext, and the required declassification policy is similarly defined.

Now, a known problem with declassification schemes is that of *occlusion* [16], where a legitimately declassified information masks the release of other secrets. Being a *what* policy, our policy enforcement prevents such a flow by permitting only the information release that is explicitly allowed by the policy. This problem is illustrated by the program listing of Fig. 4 (adapted from [1]). Suppose that we have declassified the encryption result, then revealing the content of l_1 and l_2 through the *write* statements in the program is permitted, however the value of the boolean secret h will be released additionally by this program because the inequality of l_1 and l_2 will reveal the fact that the *then* branch was executed. The analysis shows this. Let us call this program P, its analysis $\lfloor T_P \rfloor$ has the property that whenever it relates any two states σ and σ' which disagree on the observed ciphertexts, then they must both agree to a value $\sigma(h) = \sigma'(h) = \mathbf{tt}$ of h - revealing h. Hence P violates the declassification policy ($\lfloor T_P \rfloor \notin \lfloor \mathcal{E}_2 \rfloor$) because $\lfloor \mathcal{E}_2 \rfloor$ requires, for example, that $\sigma[h \mapsto \mathbf{ff}]$ must be indistinguishable from σ, which $\lfloor T_P \rfloor$ distinguishes in this case.

$l_1 := \boldsymbol{enc}_*(iv_1, k, m);$
if (h) **then** $l_2 := \boldsymbol{enc}_*(iv_2, k, m);$ **else** $l_2 := l_1;$
write l_1; **write** l_2;

Fig. 4. The Occlusion Problem

5 Analysis for Nondeterministic System Models

We now turn our attention to the security analyses of information flow under nondeterministic system models. We start by motivating the use of families of sets as a representation of information in the nondeterministic setting, which generalises the PER representation used earlier for the deterministic system model.

Consider a system, whose relational model is given by $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$. We can describe the information that the attacker gains on observing the output $v \in \mathcal{V}$ of the system by the inverse image of v under S. The inverse image $S^{-1}(v)$ of v represents the set of all *possible* inputs that can produce the output v in the system modelled by S, and thus describes the attacker's uncertainty about the inputs given the observation of v. It is thus easy to see that the family of sets $\{S^{-1}(v) \mid v \in \mathcal{V}\}$ models the uncertainties of the attacker under the observation of individual outputs of the system modelled by S.

In the special case that S models a deterministic system, in which case S is a function, it is clear that $\{S^{-1}(v) \mid v \in \mathcal{V}\}$ corresponds to the set of partitions of the kernel of the function S, which uniquely identifies the equivalence relation over $\mathbf{\Sigma}$ used to describe the information released in the previous sections. In this sense, the family of sets representation generalises the PER representation.

However, unlike the deterministic model presented earlier, where for any $v, v' \in \mathcal{V}$, $v \neq v'$ implies $S^{-1}(v) \cap S^{-1}(v') = \varnothing$, the inverse images are not necessarily disjoint under a nondeterministic model since the outputs resulting from any given input may not necessarily be unique. This leads to an avenue of information release in nondeterministic systems. The property that the nondeterministic system modelled by S does not necessarily partition its domain introduces the possibility that an attacker might gain further information by repeated execution of the system under a fixed input. To illustrate this, suppose $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$ models a nondeterministic system, where $\mathbf{\Sigma} = \{\sigma_1, \sigma_2, \sigma_3\}$ and $\mathcal{V} = \{v_1, v_2\}$ and where the graph of the relation S is given by $\mathit{graph}(S) = \{(\sigma_1, v_1), (\sigma_2, v_1), (\sigma_2, v_2), (\sigma_3, v_2)\}$. The model is nondeterministic since the input σ_2 can produce outputs v_1 or v_2. By observing an output v_1 the attacker learns that the input must be one of σ_1 and σ_2, as suggested by $S^{-1}(v_1) = \{\sigma_1, \sigma_2\}$. Similarly, on observing the output v_2, the attacker learns that the input is in the set $S^{-1}(v_2) = \{\sigma_2, \sigma_3\}$. However, if under a fixed input the attacker observes outputs v_1 and v_2 in different runs of the system, then the attacker confirms that the input to the system must be σ_2 - derived by taking the intersection $S^{-1}(v_1) \cap S^{-1}(v_2)$. This avenue of information leakage is not available under the deterministic system model since for a fixed input, the output of the system always remains the same. This leads us to a definition of information based on families of sets.

Definition 7 (Lattice of Possibilistic Information). *Let $\Sigma_J = \{\Sigma_j \subseteq \mathbf{\Sigma} \mid j \in J\}$ be a family of subsets of $\mathbf{\Sigma}$ indexed by a set J. Define the operation $\langle\!\langle \cdot \rangle\!\rangle$ on families of subsets of $\mathbf{\Sigma}$ as $\langle\!\langle \Sigma_J \rangle\!\rangle \triangleq \bigcup_{K \subseteq J} \{\bigcap \Sigma_K\}$, which closes the family under intersections. Define the* possibilistic information *set over $\mathbf{\Sigma}$ as $\mathit{FAM}(\mathbf{\Sigma}) \triangleq \{\langle\!\langle \Sigma_J \rangle\!\rangle | \Sigma_J$ is a family of subsets of $\mathbf{\Sigma}\}$ to be the information contained in families of sets over $\mathbf{\Sigma}$. For any $\langle\!\langle \Sigma_J \rangle\!\rangle, \langle\!\langle \Sigma_K \rangle\!\rangle \in \mathit{FAM}(\mathbf{\Sigma})$ define the join operation as $\langle\!\langle \Sigma_J \rangle\!\rangle \sqcup \langle\!\langle \Sigma_K \rangle\!\rangle = \langle\!\langle \Sigma_{J \cup K} \rangle\!\rangle$ and define the partial order $\sqsubseteq$ to be the subset ordering of families in $\mathit{FAM}(\mathbf{\Sigma})$.*

The intuition behind the partial ordering $\langle\!\langle \Sigma_J \rangle\!\rangle \sqsubseteq \langle\!\langle \Sigma_K \rangle\!\rangle$ is that every information token in $\langle\!\langle \Sigma_J \rangle\!\rangle$ (which is derivable by taking the intersection of some elements of Σ_J) is also present in $\langle\!\langle \Sigma_K \rangle\!\rangle$. This leads us to a description of information that can be derived about the nondeterministic system modelled by $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$ as $[\![S]\!] \triangleq \langle\!\langle \{S^{-1}(v) \mid v \in \mathcal{V}\} \rangle\!\rangle$ which identifies the minimal sets of inputs that can produce a given output in the system. Based on this, and choosing $\mathcal{I} = \mathit{FAM}(\mathbf{\Sigma})$, we can define the possibilistic information flow as

$$[\![S]\!]^{\mathcal{I}} = \{f \mid \forall F \in \mathit{FAM}(\mathbf{\Sigma}), f(F) = F \sqcup [\![S]\!]\}.$$

The ordering of information content of elements of $\mathit{FAM}(\mathbf{\Sigma})$ forms a complete lattice of information.

Theorem 2. *The family of sets $\langle \mathit{FAM}(\mathbf{\Sigma}), \sqsubseteq, \sqcup \rangle$ over $\mathbf{\Sigma}$ representing the set of possibilistic information is a complete lattice.*

5.1 Language-Based Instantiation

We make a conservative extension to the *While* language, through the introduction of a nondeterministic choice constructor $[]$ to obtain the language *While-ND* (*While* with NonDeterminism) which exhibits nondeterministic behaviour. In *While-ND*, the program $c_1 [] c_2$ makes an invisible but arbitrary choice in the execution either command c_1 or command c_2. Consequently, the operational semantics of *While-ND* extends that of *While* as shown in Fig. 5.

$$\langle c_1 [] c_2, \sigma\rangle \xrightarrow{\varepsilon} \langle c_1, \sigma\rangle \qquad \langle c_1 [] c_2, \sigma\rangle \xrightarrow{\varepsilon} \langle c_2, \sigma\rangle$$

Fig. 5. Extending *While* with Possibilistic Nondeterminism

To deal with the fact that the trace of a *While-ND* program P is no more unique for a given starting state, the observational power is extended to sets of traces so that $obs^*(\cdot)$ is the set of observations for a given starting state, obtained by applying $obs(\cdot)$ to all the traces that can result from that state. This produces a relational model S_P of P, which relates σ to v iff there exists $v \in obs^*(t_{\langle P,\sigma\rangle})$. The resulting possibilistic analysis is also termination-sensitive.

Suppose the integer (secret) h is a parameter to the *While-ND* program given by P = `if(`$h = 0$`)then skip` $[]$ *loop* `else skip`. This program may either terminate or loop indefinitely when the secret value h is chosen to be zero. Thus, it is easy to see that the attacker may learn the value of h to be zero when the program fails to terminate. The set of possible observation of P is given by $\mathcal{V}_P = \{\langle\downarrow\rangle, \langle\uparrow\rangle\}$ where $\langle\downarrow\rangle$ corresponds to the observation during the terminating traces and $\langle\uparrow\rangle$ corresponds to the observation of the diverging trace. If we represent the set of program states as one-tuples: $\{(n) \mid n \in \mathbb{Z}\}$, then the relational model of P is $S_P \subseteq \mathbf{\Sigma} \times \mathcal{V}_P$, whose graph is given by $\{((0), \langle\uparrow\rangle), ((n), \langle\downarrow\rangle) \mid n \in \mathbb{Z}\}$. Thus, we have the following inverse images: $S_P^{-1}(\langle\uparrow\rangle) = \{(0)\}$ and $S_P^{-1}(\langle\downarrow\rangle) = \mathbf{\Sigma}$, and $[\![S_P]\!] = \{\{(0)\}, \mathbf{\Sigma}\}$ reflecting the fact that the attacker can learn when the secret value is zero. Now consider another program P_A = `if(`$h = 0$`)then skip` $[]$ *loop* `else` *loop*. In this case, the observation of *termination* reveals to the attacker that the value of the integer secret h is zero. The analysis is similar to that of P, but now we have $graph(S_{P_A}) = \{((0), \langle\downarrow\rangle), ((n), \langle\uparrow\rangle) \mid n \in \mathbb{Z}\}$ and $S_{P_A}^{-1}(\langle\downarrow\rangle) = \{(0)\}$ and $S_{P_A}^{-1}(\langle\uparrow\rangle) = \mathbf{\Sigma}$. Thus, $[\![S_{P_A}]\!] = \{\{(0)\}, \mathbf{\Sigma}\}$, and it is intuitive P_A releases the same information as P.

We can define a noninterference policy, which prevents any information from being gained about h in the examples above. This policy is given by $\mathscr{P}_{\mathbf{\Sigma}} \triangleq \{f \mid \forall F \in FAM(\mathbf{\Sigma}), f(F) = F\}$, which is modelled by the identity map on the lattice $FAM(\mathbf{\Sigma})$. We see that both P and P_A above violate this policy. This is clear, for example, given an initial knowledge $\{\mathbf{\Sigma}\}$ of the attacker representing lack of information, the attacker learns $\{\{(0)\}, \mathbf{\Sigma}\} \notin \{\mathbf{\Sigma}\}$ through these programs.

Now consider the program P_B = `if(`$h = 0$`)then skip` $[]$ *loop* `else skip` $[]$ *loop* which may or may not terminate regardless of the chosen value of h. Intuitively, this program should not reveal any information to the attacker as its behaviour is independent of the choice of h. This is confirmed by the analysis because

$graph(S_{P_B}) = \{((n), \langle\downarrow\rangle), ((n), \langle\uparrow\rangle) \mid n \in \mathbb{Z}\}$. Thus, $S^{-1}_{P_B}(\langle\downarrow\rangle) = S^{-1}_{P_B}(\langle\uparrow\rangle) = \mathbf{\Sigma}$. This means that $[\![S_{P_B}]\!] = \{\mathbf{\Sigma}\}$, showing the fact that the attacker learns nothing by observing the execution of P_B. Thus, the program P_B satisfies the noninterference policy $\mathscr{P}_{\mathbf{\Sigma}}$ defined above.

Now suppose that P_C is a *While-ND* program which always terminates. Similarly to the analysis under deterministic programs, the information flow of P_C is preserved in the program $P' \triangleq P_C; loop$. This is easy to see because there is set isomorphism between the sets $\mathcal{V}_{P_C}$ and $\mathcal{V}_{P'}$ of outputs of P_C and P' respectively, which appends $\uparrow$ to all $\langle a\rangle \in \mathcal{V}_{P_C}$ such that $\forall \sigma \in \mathbf{\Sigma}, \sigma\ S_{P_C}\ \langle a\rangle \iff \sigma\ S_{P'}\ \langle a, \uparrow\rangle$ where $S_{P_C} \subseteq \mathbf{\Sigma} \times \mathcal{V}_{P_C}$ and $S_{P'} \subseteq \mathbf{\Sigma} \times \mathcal{V}_{P'}$ are respectively the relational models of P_C and P'. Hence, we have $[\![S_{P_C}]\!] = [\![S_{P'}]\!]$. Finally, let P_D be a *While-ND* program such that $P' \triangleq loop; P_D$. Like the deterministic analysis, this program reveals no information since for all $\sigma \in \mathbf{\Sigma}, \sigma\ S_{P'}\langle\uparrow\rangle$ holds and hence $[\![S_{P'}]\!] = \{\mathbf{\Sigma}\}$.

5.2 Information-Theoretic Representation

We can perform information-theoretic analyses under the relational model $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$ of a system by assigning probability distributions to the input space $\mathbf{\Sigma}$ to model the attacker's uncertainty about the choice of inputs, and by considering the probability distribution of $\mathcal{V}$ induced by the execution of the system. Because of space restrictions, specific information-theoretic analysis techniques are not shown in this paper. However, we note that quantitative measures, such as Shannon's entropy measure, which describe the attacker's uncertainty about the input distribution before and after observing program outputs can be arranged on a lattice according to the order of the quantitative amount of information that is released about the system inputs. This observation allows us to use the lattice-based approach to enforce security under this setting, where the amount of information that is permitted to be released about a given secret is captured by the numerical order relation $\leq$ on the quantitative information measure.

6 Related Work

This paper falls into the area of language-based security, which is an established and active research area [14]. We have proposed a lattice-theoretic approach to the enforcement of *what* declassification policies [16,17]. Several approaches, such as in [4,5,6,9,11,15], have been applied to study the *what* dimension of information release. While these approaches study the properties of specific representation of information used, we study the problem from a more general viewpoint of information as a lattice structure which can be used to enforce *what* policies. The resulting lattice-based policy model has been shown to be capable of handling controlled information release, such as during encryption, and it does not suffer from the *occlusion* problem.

In [15], PERs are used to describe the security properties of a program. Given a function $f : A \to B$ which is a Scott-style denotation of a program P (information flow due to nontermination is handled by requiring that secret inputs do not affect the termination behaviour), the security property of P is stated as a PER-transformer $(\!|f|\!) : R_A \multimap R_B$, where $R_A \in \mathrm{PER}(A)$ and $R_B \in \mathrm{PER}(B)$. The property $(\!|f|\!) : R_A \multimap R_B$ holds iff for all

$a, a' \in A, a\ R_A\ a' \implies f(a)\ R_B\ f(a')$. Thus, $(\!|f|\!) : R_A \twoheadrightarrow R_B$ may be interpreted as a policy which P satisfies. If we assume that $A = B = \Sigma = H \times L$ represents the set of all states of P, partitioned to the *high*-secret (H) and *low*-public (L) parts, then P is said to be (noninterference) secure iff $(\!|f|\!) : \mathit{all} \bullet \mathit{id} \twoheadrightarrow \mathit{all} \bullet \mathit{id}$, where $\sigma_{\downarrow L}$ is the projection of the state σ to L and for all $\sigma, \sigma' \in \Sigma, \sigma\ \mathit{all} \bullet \mathit{id}\ \sigma'$ iff $\sigma_{\downarrow L} = \sigma'_{\downarrow L}$. The meaning of $(\!|f|\!) : \mathit{all} \bullet \mathit{id} \twoheadrightarrow \mathit{all} \bullet \mathit{id}$ is that the attacker which can only observe the public part of state before $P's$ execution and on its termination cannot distinguish two runs which agree on the initial L-input. This corresponds to an observational model under our approach, which for any initial input state σ is given by $\mathit{obs}_{\twoheadrightarrow}(t_{\langle P,\sigma\rangle}) = \langle \sigma_{\downarrow L}, f(\sigma)_{\downarrow L}\rangle$. Consequently, the information released is characterised by the PER defined such that $\sigma, \sigma' \in \Sigma, \sigma \left\lfloor T_{\langle P,\twoheadrightarrow\rangle} \right\rfloor \sigma' \iff \mathit{obs}_{\twoheadrightarrow}(t_{\langle P,\sigma\rangle}) = \mathit{obs}_{\twoheadrightarrow}(t_{\langle P,\sigma'\rangle})$. This definition computes the least PER $\left\lfloor T_{\langle P,\twoheadrightarrow\rangle} \right\rfloor$, that is, the most refined policy for which P is secure under the attacker model $\mathit{obs}_{\twoheadrightarrow}(\cdot)$, because $(\!|f|\!) : R \bullet \mathit{id} \twoheadrightarrow \mathit{all} \bullet \mathit{id} \implies \left\lfloor T_{\langle P,\twoheadrightarrow\rangle} \right\rfloor \sqsubseteq R \bullet \mathit{id}$. For any $R \bullet \mathit{id}$ that is strictly less than $\left\lfloor T_{\langle P,\twoheadrightarrow\rangle} \right\rfloor$, P will produce outputs that can be distinguished by $\mathit{obs}_{\twoheadrightarrow}(\cdot)$ under some variation of the H-projection of inputs that are related by R.

The abstract noninterference definition of [6] introduces attacker models as abstract interpretations which can observe only properties of data in the concrete domain. The concrete domain is partitioned into two sets H and L, which represent the domain of secret and public values respectively, and state is modelled as tuples in $\Sigma = H \times L$. The attacker is modelled as a pair of abstractions $\langle \eta, \rho \rangle$, where $\eta, \rho \in \mathit{uco}(\mathcal{P}(L))$ are upper closure operators (*extensive*, *monotone*, and *idempotent* maps) on the powerset lattice of public values ordered by subset inclusion. The closure operators η and ρ model what the attacker can observe about the program's public inputs and outputs respectively. The concrete semantics of the program P is formalised using *angelic denotational semantics*, which associates an input-output function, $[\![P]\!] : \Sigma \to \Sigma$, with P and ignores nontermination. Furthermore, the observation of (public) values occur at the beginning of program execution and on program termination. To slightly simplify the notations, we shall denote the concrete semantics of P as a map $[\![P]\!] : H \times L \to L$, throwing away the H projection of state on termination, which is not used.

Our observational model is more general since we place no restriction on the nature of the observational power function, as opposed to the requirement in [6] where they must be closure operators. Furthermore, our observational model is not restricted to the observation of values at the beginning and end of program execution. In particular, the attacker $\langle \eta, \rho \rangle$ may be obtained under our model by defining an observational power function on traces, where for any $\sigma, \hat{\sigma} \in \Sigma$, and trace $t_{\langle P,\sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a} \cdots \xrightarrow{a'} \langle \cdot, \hat{\sigma}\rangle$ we have $\mathit{obs}_{\langle \eta,\rho\rangle}(t_{\langle P,\sigma\rangle}) = \langle \eta(\{\sigma_{\downarrow L}\}), \rho(\{\hat{\sigma}_{\downarrow L}\})\rangle$. This definition says that the attacker only observes the η-property of the L-projection of the initial state and the ρ-property of the L-projection of the terminating state of P. Consequently, the information released under this observational model is the PER $\left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor$ over Σ defined such that for any $\sigma, \sigma' \in \Sigma, \sigma \left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor \sigma'$ iff $\mathit{obs}_{\langle \eta,\rho\rangle}(t_{\langle P,\sigma\rangle}) = \mathit{obs}_{\langle \eta,\rho\rangle}(t_{\langle P,\sigma'\rangle})$. It is thus clear that for any $\sigma, \sigma' \in \Sigma$, where $\langle P, \sigma\rangle \xrightarrow{a_1} \cdots \xrightarrow{a_n} \langle \cdot, \hat{\sigma}\rangle$ and $\langle P, \sigma'\rangle \xrightarrow{a'_1} \cdots \xrightarrow{a'_m} \langle \cdot, \hat{\sigma}'\rangle$ we have

$$\sigma \left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor \sigma' \iff \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \wedge \rho(\{\hat{\sigma}_{\downarrow L}\}) = \rho(\{\hat{\sigma}'_{\downarrow L}\}) \implies \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\{\hat{\sigma}_{\downarrow L}\}) = \rho(\{\hat{\sigma}'_{\downarrow L}\}).$$

By this we immediately obtain the narrow abstract noninterference (NANI) definition:

$$[\eta]P[\rho] \iff \forall \sigma, \sigma' \in \Sigma, \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho([\![P]\!](\sigma)) = \rho([\![P]\!](\sigma')).$$

Thus, $\left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor$ is the least PER over states for which any pair of states that it relates satisfies NANI in P.

The NANI definition causes what is referred to as "deceptive flows", when η-undistinguished public input values cause a variation which makes P to violate NANI. In order to deal with this problem, abstractions of L values are passed as program parameters and another abstraction $\phi \in uco(\mathcal{P}(H))$ is introduced on the input secret values. This results in the abstract noninterference (ANI) property, $[\eta]P(\phi \rightsquigarrow [\![\rho)$, of [6].

Let $\sigma \in \Sigma$ be a state, and define the set $\Sigma_\sigma^{\eta,\phi}$ of L-projections of the terminating states of P due to the execution of P from any starting state which agrees with σ on the η-property of the L-projection and on the ϕ-property of the H-projection to be

$$\Sigma_\sigma^{\eta,\phi} \triangleq \left\{ \hat{\sigma}_{\downarrow L} \;\middle|\; \begin{array}{l} \sigma'' \in \Sigma. \langle P, \sigma''\rangle \xrightarrow{a} \cdots \xrightarrow{a'} \langle \cdot, \hat{\sigma}\rangle. \\ \eta(\{\sigma''_{\downarrow L}\}) = \eta(\{\sigma_{\downarrow L}\}), \phi(\{\sigma''_{\downarrow H}\}) = \phi(\{\sigma_{\downarrow H}\}). \end{array} \right\}$$

In [6] the ANI property, $[\eta]P(\phi \rightsquigarrow [\![\rho)$, is now defined to hold iff for all $\sigma, \sigma' \in \Sigma$, $\eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi})$.

We can obtain this observational model under our framework by defining an observational power function on traces, such that for any $\sigma \in \Sigma$, and terminating trace $t_{\langle P,\sigma\rangle}$ we have $obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma\rangle}) = \left\langle \eta(\{\sigma_{\downarrow L}\}), \rho(\Sigma_\sigma^{\eta,\phi}) \right\rangle$. This definition requires that no public output can be distinguished by ρ for any initial state which is L-indistinguishable from to σ under η and H-indistinguishable from σ under ϕ. Thus, as usual, the information released under our relational model is the PER $\left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor$ over Σ defined such that for any $\sigma, \sigma' \in \Sigma$, $\sigma \left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor \sigma'$ iff $obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma\rangle}) = obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma'\rangle})$. Hence, for all $\sigma, \sigma' \in \Sigma$, we have that

$$\sigma \left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor \sigma' \iff \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \wedge \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi}) \implies \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi}).$$

By this we obtain ANI property $[\eta]P(\phi \rightsquigarrow [\![\rho)$, where $\left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor$ is the least PER over Σ, for which any pair of states that it relates satisfies ANI in P. We note that this property, which prevents the attacker from gaining information ϕ about secret inputs (see [12]), can also be arranged on a lattice of information, in particular, because $\phi \in uco(\mathcal{P}(H))$ forms a complete lattice.

In [3,8], action-based operational semantics approaches are used in deterministic language settings to check program security, similar to our labelled-transition semantics.

However, the analyses are not termination-sensitive. Furthermore, these approaches assume a fixed attacker model, whereas analysis is parametrised by a chosen attacker's observational power under our approach. With respect to the model of information, the gradual release knowledge of [3] is modelled by sets $\Sigma \subseteq \mathbf{\Sigma}$, which represent the knowledge (more precisely, the uncertainty) of the attacker at any point in time. These sets shrink over time and the knowledge is monotone, which agrees with the extensivity and monotonicity properties our information flow function definition. However, the PER over $\mathbf{\Sigma}$ representation generalises the subsets of $\mathbf{\Sigma}$ representation, since for any $\Sigma \subseteq \mathbf{\Sigma}$, there is a PER R_Σ which encodes this set, where $\sigma \; R_\Sigma \; \sigma'$ holds iff $\sigma, \sigma' \in \Sigma$. Hence, every instantiation of knowledge in [3] can be encoded as PERs.

7 Conclusion

We have presented a policy model for secure information flow based on lattices of information to enforce *what* declassification policies. We demonstrated that various representations of information such as PERs, families of sets, information-theoretic measures, and closure operators may be unified under the lattice model of information, providing us with a uniform way to enforce policies based on the lattice order. A termination-sensitive analysis method was also presented, which is parametric to a chosen attacker model, and which derives a system's information flow property from the operational semantics. A static analysis technique and type system is currently being developed as an application of the relational model approach. An area of future work is to study the integration of the lattice-of-information model with other mechanisms such as security classes in a multi-level security system for the enforcement of secure information flow. Such an integration would allow us to express not only *what* properties in policies, but also *who* properties.

References

1. Askarov, A., Hedin, D., Sabelfeld, A.: Cryptographically-masked flows. Theoretical Computer Science 402(2-3), 82–101 (2008)
2. Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 333–348. Springer, Heidelberg (2008)
3. Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: IEEE Symposium on Security and Privacy, pp. 207–221. IEEE Computer Society, Los Alamitos (2007)
4. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15(3), 321–371 (2007)
5. Cohen, E.S.: Information transmission in computational systems. In: SOSP 1977: Proceedings of the sixth ACM symposium on Operating systems principles, pp. 133–139. ACM Press, New York (1977)
6. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 186–197. ACM Press, New York (2004)

7. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, pp. 11–20. IEEE Computer Society Press, Los Alamitos (1982)
8. Le Guernic, G., Banerjee, A., Jensen, T.P., Schmidt, D.A.: Automata-based confidentiality monitoring. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 75–89. Springer, Heidelberg (2006)
9. Joshi, R., Leino, K.R.M.: A semantic approach to secure information flow. Science of Computer Programming 37(1-3), 113–138 (2000)
10. Kohlas, J.: Information Algebras: Generic Structures for Inference. Springer, Heidelberg (2003)
11. Landauer, J., Redmond, T.: A lattice of information. In: Proceedings of the Computer Security Foundations Workshop VI (CSFW 1993), Washington, Brussels, Tokyo, pp. 65–70. IEEE, Los Alamitos (1993)
12. Mastroeni, I.: On the Rôle of abstract non-interference in language-based security. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 418–433. Springer, Heidelberg (2005)
13. Ryan, P., McLean, J., Millen, J., Gligor, V.: Non-interference, who needs it? In: 14th IEEE Computer Security Foundations Workshop (CSFW 2001), Washington, Brussels, Tokyo, pp. 237–240. IEEE, Los Alamitos (2001)
14. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
15. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation 14(1), 59–91 (2001)
16. Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: CSFW 2005: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW 2005), Washington, DC, USA, pp. 255–269. IEEE Computer Society, Los Alamitos (2005)
17. Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. Journal of Computer Security (2007)