Language-Based Information Erasure

Stephen Chong Andrew C. Myers
Department of Computer Science
Cornell University
{schong,andru}@cs.cornell.edu

Abstract

Real computing systems sometimes need to forget sensitive information. This paper explores the specification and semantics of information erasure *policies, which impose a strong, end-to-end requirement that information be either erased or made less accessible. Simple lattice-based information flow policies, corresponding to a noninterference requirement, are augmented with the ability to express explicit erasure and declassification policies. Examples are given of applying this expressive policy language to real systems. The paper gives tools for reasoning about policy enforcement either statically or dynamically. Further, the significance of these policies to security is formally explained in terms of trace-based semantic security properties: generalizations of noninterference that accommodate erasure and declassification.*

1. Introduction

Information flow control is a promising approach for providing strong, end-to-end security guarantees about the propagation of information within a system. An important (and largely overlooked) aspect of information flow is that real-world systems are often required not to retain certain information after a specific time or event. In this paper we study the problem of describing this *information erasure*. We present a security policy framework that can express information erasure policies, and investigate the semantic security conditions enjoyed by systems that enforce the policies expressible in the framework.

This research was supported in part by the Department of the Navy, Office of Naval Research, ONR Grant N00014-01-1-0968, and by National Science Foundation grants 0208642, 0133302, and 0430161. Andrew Myers is supported by an Alfred P. Sloan Research Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright annotation thereon. The views and conclusions here are those of the authors and do not necessarily reflect the views of these sponsors.

The classic approach to specifying end-to-end information security is based on *noninterference* [16], requiring for example that sensitive information not affect public information. However, noninterference is not a sufficiently precise tool for expressing the security requirements of real systems. It has often been observed (e.g, [35, 31, 28, 40, 32, 22, 7, 30, 36, 25, 15]) that noninterference is too restrictive: real systems need to release some amount of sensitive information. This is usually accomplished by allowing noninterference to be tempered by *declassification*. However, noninterference is sometimes also not restrictive *enough*: sometimes information needs to become more sensitive or to even to be forgotten completely. This paper explores the meaning of strong information security policies that incorporate both declassification and erasure.

Many real-world systems have security requirements regarding the erasure and declassification of information. For example, consider the following systems:

- **Cryptographic devices.** Devices and software that decrypt using secret keys are required to explicitly erase ("zeroize") secret keys once they are done using them.
- **Electronic voting.** Ballots of individual voters must be kept confidential, but the final results of the election, calculated from the voters' ballots, are publicly released. Other information may need to be erased, especially if it connects voters to their ballots.
- **Mobile computing.** A mobile device such as a laptop computer may operate in environments of varying sensitivity and vulnerability. When a mobile device leaves a secure environment where sensitive information is accessible (e.g., corporate headquarters, connected to the corporate LAN) for a less secure environment, the mobile device may need to erase sensitive stored information.
- **Online transactions.** An individual may only be prepared to release sensitive information to a service provider in order to perform a transaction, if the service provider guarantees not to retain the information after the transaction is complete. For example, she may

be prepared to give her credit card details to a merchant to make a purchase, so long as the merchant erases those details afterward.

- **Medical information systems.** Health care providers hold sensitive patient information, including demographic and medical data. In many countries, legislation controls under what conditions patient information may be released, and to whom.

As these examples suggest, the reasons for erasing or declassifying information are diverse, often complex, yet crucial to security. We therefore propose a security policy framework that allows the specification of both erasure and declassification policies, and supports application-specific reasoning about the erasure and declassification of information. In this framework, *erasure policies* specify what policy should initially be enforced on information, the conditions under which the information must be erased, and (since information may be allowed to exist in a system after erasure in a restricted form) what policy must be enforced on information to allow it to survive erasure. *Declassification policies*, first presented in [7], specify what policy should initially be enforced on the information, the conditions under which the information may be declassified, and the policy that should be enforced on the information after declassification.

It is important to note that erasure and declassification policies govern the use of *information* rather than of the *locations* where information is stored. In particular, if a piece of data has an erasure policy, it means that not only should the data itself be erased under the specified conditions, but also any information derived from it should be erased. Thus, erasure policies describe strong, end-to-end restrictions on how information may be used. Information erasure and declassification can be seen as opposites: As time progresses, declassification permits more information flows in a system; by contrast, erasure permits fewer information flows as time moves forward.

Much recent work (e.g., [29, 40, 32, 22, 7, 30, 36, 25, 15]) has focused on security properties that generalize noninterference to permit reasoning about declassification; as this paper shows, many of these security properties have parallels involving information erasure. We show that some of these properties are possessed by systems that enforce our framework's policies.

In our policy framework, erasure and declassification are controlled by certain *conditions* under which erasure and declassification are respectively required and permitted. These conditions are inevitably application-specific, so the framework does not specify the logic for expressing these conditions, preserving generality. The framework is general in another sense; although the policies are intended to be used for program annotation and analysis, the actual form of the programming language is not specified.

This paper does not focus on how to construct systems that enforce our framework's policies; that topic is left to future work. However, we expect that erasure and declassification policies can be enforced through a combination of static analysis (such as a security type system, e.g., [41, 39, 20, 27, 1, 3, 33]) and run-time mechanisms.

The rest of the paper is organized as follows. Section 2 introduces a unified framework for erasure and declassification security policies, and presents two examples of how this framework can express real-world information erasure requirements. Section 3 gives an ordering relation that enables analysis of legal information flows in this framework, together with (and consistent with) a denotational semantics that captures the meaning of policies. Section 4 discusses the semantic security conditions enjoyed by systems that enforce erasure and declassification policies. We discuss related work in Section 5, and conclude in Section 6. Proofs of the main theorems and lemmas are given in appendices.

2. Erasure and declassification policies

This section shows how a single policy framework can incorporate both erasure and declassification policies, building on lattice-based information flow policies. It then presents some example policies that capture real-world information erasure requirements.

2.1. Policies

We assume there is some underlying lattice of security levels $\mathcal{L}$ [11], giving a base vocabulary for expressing erasure and declassification policies. The lattice ordering on $\mathcal{L}$ is written as $\sqsubseteq$.

There are three kinds of policies, given by the syntax in Figure 1.

$\ell \in \mathcal{L}$	Lattice element
c, d	Conditions
$p, q ::=$	Policies
$\quad \ell$	$\quad$ Lattice-level policy
$\quad p \overset{c}{\rightsquigarrow} p'$	$\quad$ Declassification policy
$\quad p \overset{c}{\nearrow} p'$	$\quad$ Erasure policy

Figure 1. Syntax for policies

A lattice-level policy ℓ is the simplest kind of policy: information labeled with security policy ℓ must be used in accordance with the security level $\ell \in \mathcal{L}$. The intuition is that it should only affect information at level ℓ or higher.

An erasure policy $p \overset{c}{\nearrow} p'$ requires that the policy p be enforced on information, and in addition once condition c is

satisfied, policy p' *must* also be enforced on the information. Therefore, once c is satisfied, the system must erase the contents of any location affected by the information if that location is governed by the policy $p \overset{c}{\nearrow} p'$ (or by any other policy not at least as restrictive as p').

For example, consider an erasure policy $L \overset{c}{\nearrow} H$, where H and L are elements of the lattice $\mathcal{L}$ such that $L \sqsubseteq H$ and $L \neq H$. Initially, data labeled with this policy would be usable at the level L. However, once condition c is satisfied, the data must either be erased from the system, or must have the policy H enforced on it; either way, the data is no longer usable at level L, and indeed, at any level ℓ such that $H \not\sqsubseteq \ell$.

A declassification policy $p \overset{c}{\rightsquigarrow} p'$ means that the policy p must be enforced on that information, but whenever the condition c is satisfied, the data may be declassified; after declassification, the policy p' must be enforced on the declassified information [7].

Conditions c are used to express when it is permissible to declassify information, and when it is necessary to erase information. If a condition is false at some point during execution, then becomes true, and finally returns to false, any information governed by a policy $p \overset{c}{\nearrow} p'$ during the first period must still be erased when c is false again. And information that was declassified when the condition was true may remain declassified when c is false again.

For generality, we do not specify a logic for conditions; any logic will suffice as long as it is possible to reason about whether a condition c is satisfied at a given point of a system's execution. In particular, given a trace τ of a system, then the relation $\tau \vDash c$ is true if and only if the condition c is satisfied when the system has produced the trace τ. Thus, to instantiate this generic framework, one must choose an appropriate language and semantics for conditions, and provide a sound way to check the satisfaction relation $\tau \vDash c$. (One must also provide sound procedures for checking certain other assertions that are related to the satisfaction relation; more details are found in Section 3.1). For expressive condition languages, the checking of conditions is likely to be incomplete. The effect of incompleteness will be just to make security analysis more conservative.

To develop intuition for the erasure and declassification policies, and to show the expressiveness of the policy framework, we present two examples that highlight real-world uses of information security policies and show how such policies can be represented using our policy framework.

2.2. Example: mobile computing

A mobile device, such as a laptop computer, may operate in several environments of varying sensitivity and vulnerability. When a mobile device leaves a secure environment (where sensitive information is accessible) for a less secure environment, it may be necessary to ensure that the mobile device does not contain any sensitive information: the mobile device needs to erase sensitive information before entering a less secure environment.

For example, suppose a laptop is used both at corporate headquarters and on client sites. At corporate headquarters, it is connected to the corporate LAN, and has access to sensitive documents; at the client site, it may be possible for client personnel to use the laptop.

When sensitive documents are downloaded onto the laptop at headquarters, a suitable security policy for the documents is $H \overset{noLAN}{\nearrow} \top$, where $noLAN$ is true when the laptop is disconnected from the corporate LAN, H is a security level for the sensitive documents, and $\top$ is a security level so high that the laptop is not permitted to hold any data at that level. Thus, the sensitive documents must be removed from the laptop at or before the time that the laptop is disconnected from the LAN. Rather than leave the enforcement of this policy to the laptop user, the document management system on the laptop could automatically enforce this policy, erasing sensitive documents whenever the laptop is disconnected from the LAN. An efficient alternative to erasing the actual documents would be to encrypt them and erase the key.

2.3. Example: online transaction

Consider a consumer purchasing a product from a merchant over a network. In order to complete the transaction, the consumer has to provide a credit card number to the merchant. The merchant promises not to keep any record of the credit card number after the transaction. However, once the consumer has approved the purchase, the merchant must send the credit card number to the bank, which *will* keep a record of the credit card number.

Let M be a security level corresponding to information stored by the merchant. Let B be a security level corresponding to information stored by the bank. Then a suitable policy for the credit card number is $(M \overset{pur}{\rightsquigarrow} B) \overset{end}{\nearrow} B$, where pur is a condition that is true once the consumer has approved the purchase, and end is a condition that is true by the end of the transaction.

Note that the policy $(M \overset{pur}{\rightsquigarrow} B) \overset{end}{\nearrow} B$ allows the merchant to release the credit card details to the bank once the customer has approved the purchase, since (as will be made precise in the following section) information labeled with policy $(M \overset{pur}{\rightsquigarrow} B) \overset{end}{\nearrow} B$ is permitted to be relabeled with the policy B, provided the condition pur is true at the time of relabeling. However, at the end of the transaction, the policy B should be enforced on the credit card number, meaning that the bank is allowed to store the number, but the merchant must have removed the number from his system.

Now suppose that we extend the example so that the consumer can optionally allow the merchant to store the credit card number, for example, to maintain a customer profile, and save the consumer from needing to re-enter the credit card number for subsequent purchases. A suitable policy for the credit card number is now $((M \overset{pur}{\rightsquigarrow} B) \overset{end}{\nearrow} B) \overset{pro}{\rightsquigarrow} (M \overset{pur}{\rightsquigarrow} B)$, where pro is a condition that is true when the consumer has given permission for the merchant to maintain a customer profile. Note that if the consumer gives permission, then the merchant may store the credit card number with a policy $M \overset{pur}{\rightsquigarrow} B$, allowing the merchant to send the credit card number to the bank when the consumer makes a purchase; if the consumer does not give permission, then the merchant is still required to erase the credit card number by the end of the transaction.

3. Semantics

We assume that a system contains locations that are governed by various policies. As information flows between locations, the policy governing the information implicitly changes. In this section, we define a relation over policies that characterizes when it is secure to flow from one policy to another. We then give a semantics for policies, showing that this "may-flow" relation is sound with respect to these semantics. We show that the semantics has some interesting and important properties. Note that the specification of security policies, and the definition of the may-flow relation, are independent of any particular mechanism for enforcing the security policies.

3.1. May-flow relation $p \leq_\tau q$

We now define a may-flow relation $p \leq_\tau q$ on policies that describes permitted information flow. The relation is parameterized by a trace τ, because the declassification and erasure of information depends upon the satisfaction of conditions, which in turn depends on system traces τ. Intuitively, if $p \leq_\tau q$, then information labeled with policy p may securely flow to a location labeled with policy q when the system has produced trace τ. For this flow to be secure, the policy q must be at least as restrictive as the policy p, that is, anything that q permits to be done to information, p permits as well. The one exception to this principle of increasing restrictiveness is declassification, whose whole purpose is to make policies less restrictive.

Figure 2 shows the inference rules for the $p \leq_\tau q$ relation. We assume the set of traces is prefix-closed, and write $\tau' \succeq \tau$ if the trace τ' extends trace τ, and $\tau' \preceq \tau$ if τ' is a prefix of τ. We write $\tau' \succ \tau$ if τ' extends τ and $\tau' \neq \tau$. For convenience, we sometimes refer to a trace τ as if it were a time; this should be understood as referring to the time at which the system has produced the trace τ.

We write $[\tau, \tau') \not\models c$ as an abbreviation for $\forall \tau''.\ \tau' \succ \tau'' \succeq \tau \Rightarrow \tau'' \not\models c$, and $\mathsf{valid}(p, [\tau, \tau'))$ as an abbreviation for $\forall \tau''.\ \tau' \succ \tau'' \succeq \tau \Rightarrow \mathsf{valid}(p, \tau'')$. Intuitively, if $[\tau, \tau') \not\models c$, then c is not satisfied by any trace that extends τ and is a strict prefix of τ'; similarly $\mathsf{valid}(p, [\tau, \tau'))$ is true if $\mathsf{valid}(p, \tau'')$ for every trace τ'' that extends τ and is a strict prefix of τ'.

The rule (MF-LATTICE) states that information may flow from a lattice-level policy ℓ to a lattice-level policy ℓ' provided that $\ell \sqsubseteq \ell'$. Since τ is not mentioned in the premise, such flow is permitted at any time. The rule (MF-TRANS) makes the $\leq_\tau$ relationship transitive on policies.

The declassification rule (MF-DECL) permits information to flow from a declassification policy $p \overset{c}{\rightsquigarrow} p'$ to policy p' at trace τ, provided that the condition for declassification c is satisfied at trace τ. This rule captures the intuitive meaning of declassification policies: declassification may occur when the appropriate condition is satisfied. Note that Rule (MF-DECL) permits flow from $p \overset{c}{\rightsquigarrow} p'$ to p', and p' may permit operations that $p \overset{c}{\rightsquigarrow} p'$ does not.

The declassification introduction rule (MF-DECL-I) describes when it is permissible for information to flow from some policy q to the policy $p \overset{c}{\rightsquigarrow} p'$. First, it must be permitted for information to flow from q to p; second, at all times in the future, if the condition c is satisfied at that time, then it must be permitted for information to flow from q to p' at that time. In addition, at all times between now and the condition c being satisfied, the policy q is always *valid*, that is, information labeled with policy q does not need to be erased. The judgment $\mathsf{valid}(p, \tau)$ describes if a given policy p is valid at time τ. The requirement that q is valid between now and any possible declassification ensures that information flowing from q to $p \overset{c}{\rightsquigarrow} p'$ does not escape any erasure requirements that q may have.

The declassification elimination rule (MF-DECL-E) allows information to flow from a declassification policy $p \overset{c}{\rightsquigarrow} p'$ to the policy p. Intuitively, it is acceptable for information to flow from $p \overset{c}{\rightsquigarrow} p'$ to p, since the policy p is strictly more restrictive than the policy $p \overset{c}{\rightsquigarrow} p'$, which enforces everything that p does but also permits declassification to p'.

The rule (MF-DECL-DECL) describes when information may flow from one declassification policy $p \overset{c}{\rightsquigarrow} p'$ to another, more restrictive declassification policy $q \overset{d}{\rightsquigarrow} q'$. The intuition is that this happens if q is at least as restrictive as p, the policy $q \overset{d}{\rightsquigarrow} q'$ permits declassification only when $p \overset{c}{\rightsquigarrow} p'$ does (that is, whenever the condition d is satisfied, c is satisfied too), and whenever declassification is permitted, q' is at least as restrictive as p'.

As can be seen by inspection of Figure 2, each of the may-flow rules for erasure policies corresponds to a declassification rule. For example, erasure introduction (MF-

$$\text{(MF-LATTICE)}\ \frac{\ell \sqsubseteq \ell'}{\ell \leq_\tau \ell'} \qquad \text{(MF-TRANS)}\ \frac{p \leq_\tau p' \quad p' \leq_\tau p''}{p \leq_\tau p''} \qquad \text{(MF-ERASE-I)}\ \frac{}{p \leq_\tau p \not\stackrel{c}{\nearrow} p'} \qquad \text{(MF-ERASE-E)}\ \frac{p \leq_\tau q \quad p' \leq_\tau q}{p \not\stackrel{c}{\nearrow} p' \leq_\tau q}$$

$$\text{(MF-ERASE-ERASE)}\ \frac{p \leq_\tau q \quad \forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow \tau' \vDash d \text{ and } p' \leq_{\tau'} q'}{p \not\stackrel{c}{\nearrow} p' \leq_\tau q \not\stackrel{d}{\nearrow} q'} \qquad \text{(MF-DECL)}\ \frac{\tau \vDash c}{p \stackrel{c}{\rightsquigarrow} p' \leq_\tau p'} \qquad \text{(MF-DECL-I)}\ \frac{q \leq_\tau p \quad \forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow q \leq_{\tau'} p' \text{ and } \mathsf{valid}(q, [\tau, \tau'))}{q \leq_\tau p \stackrel{c}{\rightsquigarrow} p'}$$

$$\text{(MF-DECL-E)}\ \frac{}{p \stackrel{c}{\rightsquigarrow} p' \leq_\tau p} \qquad \text{(MF-DECL-DECL)}\ \frac{p \leq_\tau q \quad \forall \tau' \succeq \tau.\ \tau' \vDash d \Rightarrow \tau' \vDash c \text{ and } p' \leq_{\tau'} q'}{p \stackrel{c}{\rightsquigarrow} p' \leq_\tau q \stackrel{d}{\rightsquigarrow} q'} \qquad \frac{}{\mathsf{valid}(\ell, \tau)} \qquad \frac{\mathsf{valid}(p, \tau)}{\mathsf{valid}(p \stackrel{c}{\rightsquigarrow} p', \tau)} \qquad \frac{\mathsf{valid}(p, \tau) \quad \tau \nvDash c}{\mathsf{valid}(p \not\stackrel{c}{\nearrow} p', \tau)}$$

Figure 2. Inference rules for $p \leq_\tau q$ and $\mathsf{valid}(p, \tau)$

ERASE-I) is analogous to (MF-DECL-E): information may flow from p to $p \not\stackrel{c}{\nearrow} p'$, since $p \not\stackrel{c}{\nearrow} p'$ is strictly more restrictive than p. An erasure policy $p \not\stackrel{c}{\nearrow} p'$ enforces everything that p does, and in addition requires the information to be erased at certain times.

Erasure elimination (MF-ERASE-E) is analogous to the rule for declassification introduction, allowing information to flow from $p \not\stackrel{c}{\nearrow} p'$ to q at trace τ provided that information can flow both from p to q, and from p' to q at trace τ. Intuitively, information may flow to q since that information would not need to be erased anyway, as information is allowed to flow from p' to q. This rule is not completely analogous to (MF-DECL-I) because it requires that information can flow from p' to q at trace τ. By comparison, the rule (MF-DECL-I) requires that information must be able to flow from q to p' at traces τ' in the future such that $\tau' \vDash c$. The declassification introduction rule differs because declassification is about flows that *may* happen in the future, while erasure is about flows that *must not* happen in the future; if we used the premise $\forall \tau' \succeq \tau . \tau' \vDash c \Rightarrow p' \leq_{\tau'} q$ for (MF-ERASE-E), this would permit some insecure flows. For example, using the weaker rule, it would be possible to derive $L \not\stackrel{c}{\nearrow} H \leq_\tau L$, which is insecure, given the informal meaning of the erasure policy $L \not\stackrel{c}{\nearrow} H$.

The rule (MF-ERASE-ERASE) compares two erasure policies, $p \not\stackrel{c}{\nearrow} p'$ and $q \not\stackrel{d}{\nearrow} q'$, and is similar to (MF-DECL-DECL). Information may flow from $p \not\stackrel{c}{\nearrow} p'$ to $q \not\stackrel{d}{\nearrow} q'$ provided that q is at least as restrictive as p, and whenever $p \not\stackrel{c}{\nearrow} p'$ requires information to be erased, so does $q \not\stackrel{d}{\nearrow} q'$ (that is, whenever c is satisfied, d should be too), and whenever erasure is required, q' is at least as restrictive as p'.

There is no erasure rule analogous to (MF-DECL). This is because erasure policies specify flows that must not happen, which is difficult to capture with inference rules of this style. Instead, the onus of ensuring information is erased at appropriate times falls upon the system that enforces the policies; in Section 4 we discuss what it means for a nondeterministic state-transition system to enforce policies.

To soundly check if $p \leq_\tau q$ holds for any given p, q and τ, it is sufficient for an instantiation of the framework to provide sound procedures for checking the following assertions.

- $\ell \sqsubseteq \ell'$
- $\tau \vDash c$
- $\forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow p \leq_{\tau'} q$
- $\forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow \tau' \vDash d$
- $\forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow [\tau, \tau') \nvDash d$

Note that due to the definition of $\mathsf{valid}(p, \tau)$, a sound procedure for checking $\forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow [\tau, \tau') \nvDash d$ can also be used to check $\forall \tau' \succeq \tau.\ \tau' \vDash c \Rightarrow \mathsf{valid}(p, [\tau, \tau'))$.

What properties does the relation $\leq_\tau$ exhibit? It is easy to establish that, for a fixed trace τ, $\leq_\tau$ is a pre-order: transitive and reflexive. However, it does not form a partial order, as it is not antisymmetric. For example, for any trace τ, condition c, and lattice element $\ell \in \mathcal{L}$, we have both $\ell \leq_\tau \ell \stackrel{c}{\rightsquigarrow} \ell$ and $\ell \stackrel{c}{\rightsquigarrow} \ell \leq_\tau \ell$.

The relation $\leq_\tau$ has a greatest and least element: the top and bottom elements of the lattice $\mathcal{L}$, denoted $\top_\mathcal{L}$ and $\bot_\mathcal{L}$ respectively.

Property 3.1: *For any trace τ, the relation $\leq_\tau$ is a pre-order. Moreover, for all policies p, we have $p \leq_\tau \top_\mathcal{L}$ and $\bot_\mathcal{L} \leq_\tau p$.*

3.2. Semantics

In this subsection, we present a semantics for policies, in terms of the observation level of locations to which information can propagate.

We assume that the underlying lattice $\mathcal{L}$ can describe how observable locations are, and proceed to define the observation level of an arbitrary policy p, $\mathsf{obs}(p)$. The observability of a lattice level policy ℓ is simply ℓ, and the observability of declassification and erasure policies is just the observability of the left subpolicy: $\mathsf{obs}(p \overset{c}{\rightsquigarrow} p') = \mathsf{obs}(p \overset{c}{\not\rightsquigarrow} p') = \mathsf{obs}(p)$.

Given this notion of observation level, we can define a semantics for policies such that the semantics of a policy p describes how information initially labeled p may propagate, and become (or cease to become) observable at various lattice levels, as the system executes.

Figure 3 gives a semantics for a policy p and trace τ, written $[\![p]\!]_\tau$, as a set of pairs of traces and lattice elements, (τ', ℓ), where $\tau' \succeq \tau$. The semantics captures all the possible ways that information labeled with policy p just before trace τ may affect information in the future. More formally, we expect that if information labeled p just before trace τ may propagate to a location labeled q at time τ', then $(\tau', \mathsf{obs}(q)) \in [\![p]\!]_\tau$.

3.3. Consistency of the semantics

To show that the semantics captures its informal meaning precisely (and to prove it), we need some additional concepts and properties that relate the semantics, the observational level, and the may-flow relation.

$$
\begin{aligned}
[\![\ell]\!]_\tau &= \{(\tau', \ell') \mid \tau' \succeq \tau \text{ and } \ell \sqsubseteq \ell'\} \\
[\![p \overset{c}{\rightsquigarrow} p']\!]_\tau &= [\![p]\!]_\tau \cup \{(\tau'', \ell) \in [\![p']\!]_{\tau'} \mid \tau' \succeq \tau \text{ and } \tau' \vDash c\} \\
[\![p \overset{c}{\not\rightsquigarrow} p']\!]_\tau &= ([\![p]\!]_\tau \cap \{(\tau''', \ell) \in [\![p']\!]_{\tau'} \mid \tau' \succeq \tau \\
&\qquad \text{and } [\tau, \tau') \not\vDash c\}) \cup \\
&\qquad \{(\tau', \ell) \in [\![p]\!]_\tau \mid [\tau, \tau') \not\vDash c\}
\end{aligned}
$$

Figure 3. Semantics for policies $[\![p]\!]_\tau$

To begin with, for all policies p, we observe that as time goes on, there are fewer possible ways in which information labeled with policy p may affect information. In particular, for any policy p and traces τ and $\tau' \succeq \tau$, so long as information labeled with p does not need to be erased between τ and τ', then $[\![p]\!]_{\tau'} \subseteq [\![p]\!]_\tau$.

Property 3.2: *Let p be a policy and τ and τ' be traces such that $\tau' \succeq \tau$. If $\mathsf{valid}(p, [\tau, \tau'))$, then $[\![p]\!]_{\tau'} \subseteq [\![p]\!]_\tau$.*

A useful property of the semantics, which will be needed in later proofs, is that for any given policy p and traces τ and τ', the set of lattice levels $\{\ell \mid (\tau', \ell) \in [\![p]\!]_\tau\}$ is closed upward.

Property 3.3: *For all policies p, traces τ and τ' and lattice levels ℓ, if $(\tau', \ell) \in [\![p]\!]_\tau$, then for all ℓ' such that $\ell \sqsubseteq \ell'$ we have $(\tau', \ell') \in [\![p]\!]_\tau$.*

The following theorem shows that the may-flow relation $p \leq_\tau q$ is sound, in the sense that if information may flow from p to q, then $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$, that is, information labeled with policy p at trace τ can affect at least as much in the future as information labeled with policy q can. The proof of the theorem is given in Appendix A.

Theorem 3.4: *For all policies p, q and traces τ, if $p \leq_\tau q$ then $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$.*

The relation $p \leq_\tau q$ tells us that information labeled with policy p may flow to a location labeled q at time τ. However, in general, we are interested in reasoning not only about the locations q that information labeled p may flow to in a single step, but about *all* locations that information labeled p may propagate to. We extend the relation $p \leq_\tau q$ to the relation $p \leq_{\tau \preceq \tau'} q$, to allow us to reason about where information labeled p may propagate from the time the system has produced the trace τ to the time it has produced the trace τ', where $\tau' \succeq \tau$. Figure 4 presents the inference rules for the new relation. We have $p \leq_{\tau \preceq \tau'} q$ if there is some sequence of traces $\tau_1, \ldots, \tau_n$ such that $\tau = \tau_1 \preceq \cdots \preceq \tau_n = \tau'$, and some sequence of policies $p_0, \ldots, p_n$ such that $p = p_0 \leq_{\tau_1} p_1 \leq_{\tau_2} \cdots \leq_{\tau_n} p_n = q$. Moreover, each p_i is valid between τ_i and τ_{i+1}, which ensures that information stored in a location labeled p_i will not be erased before it can propagate, at trace τ_{i+1}.

$$
\frac{p \leq_\tau q \qquad \mathsf{valid}(q, \tau)}{p \leq_{\tau \preceq \tau} q}
\qquad\qquad
\frac{\tau' \succeq \tau'' \succeq \tau \quad p \leq_{\tau \preceq \tau''} p' \qquad \mathsf{valid}(p', [\tau'', \tau')) \qquad p' \leq_{\tau'} q \quad \mathsf{valid}(q, \tau')}{p \leq_{\tau \preceq \tau'} q}
$$

Figure 4. Inference rules for $p \leq_{\tau \preceq \tau'} q$

There is a strong and simple connection between the relation $\leq_{\tau \preceq \tau'}$ and the semantics of policies:

Property 3.5: *For all policies p and q, and traces τ and $\tau' \succeq \tau$, if $p \leq_{\tau \preceq \tau'} q$, then $[\![p]\!]_\tau \supseteq [\![q]\!]_{\tau'}$.*

There is also a simple connection between the observability of a policy and the semantics of that policy.

Property 3.6: *For all policies p and traces τ, if $\mathsf{valid}(p, \tau)$, we have $(\tau, \mathsf{obs}(p)) \in [\![p]\!]_\tau$.*

We can now state the basic theorem that relates the policy semantics to the observational level: if information labeled p at time τ may propagate to a location labeled q at time τ', then $(\tau', \mathsf{obs}(q)) \in [\![p]\!]_\tau$.

Property 3.7: *For all policies p and q and traces τ and τ' such that $\tau' \succeq \tau$, if $p \leq_{\tau \preceq \tau'} q$ then $(\tau', \mathsf{obs}(q)) \in [\![p]\!]_\tau$.*

The above facts allow us to prove some desirable properties about the $p \leq_{\tau \preceq \tau'} q$ relation, with regard to particular kinds of policies. First, we can show that for any policy ℓ and any policy q such that $\ell \leq_{\tau \preceq \tau'} q$, then the observation level of q must be at least that of ℓ. This implies that information labeled ℓ will never be observable at any level ℓ' such that $\ell \not\sqsubseteq \ell'$.

Property 3.8: *For any lattice element ℓ, and for all policies q and traces τ and $\tau' \succeq \tau$, if $\ell \leq_{\tau \preceq \tau'} q$, then $\ell \sqsubseteq \mathsf{obs}(q)$.*

The following property says that for an erasure policy $p \overset{c}{\nearrow} p'$, and any policy q that information labeled $p \overset{c}{\nearrow} p'$ can propagate to, if the information is meant to be erased at some time, then the observation level of any such q after that time must be in the semantics of both p and p'. Thus, for example, given the erasure policy $\ell \overset{c}{\nearrow} \ell'$, at any time after τ' such that $\tau' \vDash c$, the observation level of any location that information could have flowed to from $\ell \overset{c}{\nearrow} \ell'$ is at least $\ell \sqcup \ell'$. Thus, information labeled $\ell \overset{c}{\nearrow} \ell'$ is erased when c is satisfied, after which it is only observable at the level $\ell \sqcup \ell'$.

Property 3.9: *For any erasure policy $p \overset{c}{\nearrow} p'$, and for all policies q and traces τ and $\tau' \succeq \tau$, if $p \overset{c}{\nearrow} p' \leq_{\tau \preceq \tau'} q$ and there is some τ'' such that $\tau' \succeq \tau'' \succeq \tau$ and $\tau'' \vDash c$, then $(\tau', \mathsf{obs}(q)) \in [\![p]\!]_\tau \cap [\![p']\!]_{\tau'''}$, for some τ''' such that $\tau'' \succeq \tau''' \succeq \tau$.*

Finally, for a declassification policy $p \overset{c}{\rightsquigarrow} p'$, and any policy q that information from $p \overset{c}{\rightsquigarrow} p'$ can propagate to, the observation level of q must be in the semantics of either p or p'. For example, given the declassification policy $\ell \overset{c}{\rightsquigarrow} \ell'$, at any time τ', the observation level of any location that information could have flowed to from $\ell \overset{c}{\rightsquigarrow} \ell'$ is bounded below by either ℓ or ℓ'.

Property 3.10: *For any declassification policy $p \overset{c}{\rightsquigarrow} p'$, and for all policies q and traces τ and $\tau' \succeq \tau$, if $p \overset{c}{\rightsquigarrow} p' \leq_{\tau \preceq \tau'} q$ then $(\tau', \mathsf{obs}(q)) \in [\![p]\!]_\tau \cup [\![p']\!]_{\tau''}$ for some τ'' such that $\tau'' \vDash c$.*

4. Security properties

In this section we explore the semantic security properties of systems that enforce the security policies of Section 2. We introduce three new semantic security conditions related to information erasure: noninterference according to policy p, noninterference after erasure, and selective erasure.

We first present a definition for what it means for a nondeterministic state-based system to be *policy enforcing*. We then prove that policy-enforcing systems enjoy several semantic security properties: noninterference [16], noninterference according to policy p, and noninterference after erasure. We also discuss how the semantic security conditions of robust declassification [43, 30, 42] and selective declassification [32] relate to policy-enforcing systems.

4.1. Policy-enforcing systems

Let S be a system. Let Σ_S denote the *feasible states* of S, that is, all states that may occur in some execution of the system S. We assume that states are functions from locations to values, and that all states in Σ_S have the same domain. Let $\rightarrow$ be the *transition relation* of S: for any two feasible states s and s', $s \rightarrow s'$ if and only if S can atomically transition from s to s'.

Let $\mathsf{pol}(\cdot)$ be a function from locations to security policies; if x is a location and $\mathsf{pol}(x) = p$, then the policy p is associated with the location x, and the system S should enforce the policy p on information stored in location x.

The traces of S are sequences of feasible states $s_0 \ldots s_k$ such that $s_{i-1} \rightarrow s_i$ for $i \in 1..k$. We denote the set of traces of a system S by Σ_S^*, and assume that Σ_S^* is prefix-closed. As in Section 3, we use τ to range over traces.

We call the system S *policy enforcing* if the system honors erasure policies, and all information flows that occur in the system are allowed to occur according to the $\leq_\tau$ relation.

In order to honor erasure policies, a system must at least ensure that at all times τ, any location that is meant to be erased at time τ is set to a special value $\perp$ at that time. The value $\perp$ could represent either a constant, or non-existence in the system. The judgment $\mathsf{valid}(p, \tau)$, from the previous section, tells us when a location should be erased: for any p and τ, $\mathsf{valid}(p, \tau)$ is false if a location labeled with p should be erased at time τ.

We can formally define what it means for a system to honor erasure policies, and allow only permitted information flows.

Definition 4.1: System S is *policy enforcing* if

(1) for all finite traces $s_0 \ldots s_k$, and for all locations x, if not $\mathsf{valid}(\mathsf{pol}(x), s_0 \ldots s_k)$, then $s_k(x) = \perp$; and

(2) for all finite traces $s_0 \ldots s_k s_{k+1}$, and all feasible states s'_k there exists a feasible state s'_{k+1} such that $s'_k \rightarrow s'_{k+1}$, and for all locations x, if $s_{k+1}(x) \neq s'_{k+1}(x)$, then $\exists y.\ s_k(y) \neq s'_k(y)$ and $\mathsf{pol}(y) \leq_{s_0 \ldots s_{k+1}} \mathsf{pol}(x)$.

Clause (1) of the definition ensures that the system honors erasure policies. Clause (2) ensures the system allows only permitted information flows. Intuitively, if a system is policy-enforcing, then whenever information flows to a location x with policy p_x, then information flowed from some location y with policy p_y and the $\leq_\tau$ relation permits flow from p_y to p_x at that time.

Nondeterministic systems can be policy enforcing. However, the definition requires that if a location x is set nondeterministically, then only information that is allowed to

flow to $\mathsf{pol}(x)$ may influence the nondeterministic choice. For example, if x is set to a number chosen randomly between 0 and the value held in location y at trace τ, then it better be the case that $\mathsf{pol}(y) \leq_\tau \mathsf{pol}(x)$.

The policy-enforcing definition is possibilistic: clause (2) requires simply the existence of a suitable state s'_{k+1}. We believe suitably modified forms of the theorems in the following subsections should hold for a probabilistic definition of policy-enforcing systems.

The definition of a policy-enforcing system is a strong requirement. In particular, it ensures that no information is leaked through timing or termination channels. Timing and termination channels could be allowed by allowing the transition relation to be reflexive and weakening the requirement in clause (2) that "there exists a feasible state s'_{k+1} such that $s'_k \rightarrow s'_{k+1}$ ..." to the following: there exists a (possibly infinite) sequence of states $s'_k, \ldots, s'_{k+n+1}$ with $s'_i \rightarrow s'_{i+1}$ for $i \in k..(k+n)$ such that for all $i \in k..(k+n)$ and locations x, if $s_k(x) = s'_k(x)$ then $s_k(x) = s'_i(x)$, and either the sequence is infinite, or for all locations x, if $s_{k+1}(x) \neq s'_{k+n+1}(x)$ then there $\exists y.\ s_k(y) \neq s'_k(x)$ and $\mathsf{pol}(y) \leq_{s_0 \ldots s_{k+1}} \mathsf{pol}(x)$.

For ease of exposition, we do not weaken the definition, and assume that policy-enforcing systems do not leak any information through timing or termination channels. We believe that suitably weakened forms of the theorems in the following subsections hold when information may leak through these channels.

4.1.1. Policies as covert channels. Given a condition c that may occur in a policy, the satisfaction of c may depend on the trace of the system, τ, as evidenced by the relation $\tau \vDash c$ used in Sections 2 and 3. Thus, the policies enforced on locations may provide covert storage channels, modulated by the satisfaction of conditions. For example, information labeled with policy $\ell \overset{c}{\rightsquigarrow} \ell'$, for some $\ell \not\sqsubseteq \ell'$, may flow to a location labeled ℓ' only when the condition c is satisfied; if the satisfaction of c depends on some sensitive information, observing that the information flow occurred may reveal it.

To model the information that may be obtained by observing the satisfaction or non-satisfaction of a condition, we assume that for any condition c occurring in any policy in the image of $\mathsf{pol}(\cdot)$, there is a (probably fictitious) location x_c that stores the satisfaction of c. That is, for any trace $s_0 \ldots s_k$, we have $s_k(x_c) = \mathit{true}$ if and only if $s_0 \ldots s_k \vDash c$. The policy that x_c is labeled with, $\mathsf{pol}(x_c)$, describes the information that may be learned by observing whether c is satisfied.

We define a notion of *condition independence* to describe when information is independent of the satisfaction of any conditions; this notion will be useful in later subsections discussing semantic security properties. Intuitively, a policy p is condition-independent if information labeled with p cannot affect the satisfaction, or non-satisfaction, of any condition c in the system.

Definition 4.2: A policy p is *condition-independent* if for all conditions c that occur in any policy in the image of $\mathsf{pol}(\cdot)$, and for all traces τ and $\tau' \succeq \tau$, we have $p \not\preceq_{\tau \preceq \tau'} \mathsf{pol}(x_c)$, where x_c stores the satisfaction of c.

For example, if for all conditions c and traces τ, the relation $\tau \vDash c$ can be determined by statically examining the code of the system, and the code of the system is labeled with policy $\bot_{\mathcal{L}}$, then for any condition c, we have $\mathsf{pol}(x_c) = \bot_{\mathcal{L}}$; thus, any policy p such that $p \not\leq_\tau \bot_{\mathcal{L}}$ for all τ will be condition independent.

4.1.2. Making systems policy-enforcing. The definition of policy-enforcing systems is of little practical use when building systems that are intended to enforce the security policies. The development of techniques to build and/or verify that systems enforce the security policies is the subject of future work.

Since it is difficult for purely run-time mechanisms to enforce strict information flow policies [12], we envision static analysis as the primary method of building policy-enforcing systems, for example, a type system similar to those that are used in security-typed languages (e.g., [41, 39, 20, 27, 1, 3, 33]). However, additional run-time mechanisms, such as memory regions [38, 14, 17], to ensure that location lifetimes are limited appropriately, may prove useful in the enforcement of erasure policies.

The connection between security and the definition of a policy-enforcing system is not immediately apparent. However, the definition provides the tools needed to prove that policy-enforcing systems satisfy various more intuitive semantic security conditions, as discussed in the rest of this section.

4.2. Noninterference

Noninterference [16] is a semantic security condition which requires that high security inputs do not affect low security outputs. The precise definitions of input, output, and high and low security lead to slightly different definitions of noninterference. In this context, we will assume that the system's input is given in a single location; that the system's output is all values stored in the locations during the subsequent execution of the system; and that information is low security if it is observable by a given attacker, and high security otherwise.

More precisely, consider an attacker who, for some lattice element ℓ, is able to observe all and only locations x such that $\mathsf{obs}(\mathsf{pol}(x)) \sqsubseteq \ell$. A location is regarded as high security if it is not observable by the attacker, and low security if it is.

We define an *observational equivalence* relation on Σ_S, such that for any state $s \in \Sigma_S$, $[s]^{\mathcal{L}}_{\ell}$ is the equivalence class of states that are indistinguishable to the attacker; that is, $[s]^{\mathcal{L}}_{\ell} = [s']^{\mathcal{L}}_{\ell}$ if for all locations x such that $\mathsf{obs}(\mathsf{pol}(x)) \sqsubseteq \ell$, we have $s(x) = s'(x)$.

Using this notion of observational equivalence, we can state the definition of noninterference.

Definition 4.3: A system S is *noninterfering at level ℓ* for location h if for any two values v_1 and v_2, and any state s such that both $s_0 = s[h \mapsto v_1]$ and $s'_0 = s[h \mapsto v_2]$ are feasible, if $s_0 \dots s_k$ is a trace of S then there is a trace $s'_0 \dots s'_k$ such that $[s_k]^{\mathcal{L}}_{\ell} = [s'_k]^{\mathcal{L}}_{\ell}$.

In general, a system that is policy-enforcing may not be noninterfering for all levels ℓ. For example, if for some location x, the policy $\mathsf{pol}(x)$ enforced on x is a declassification policy $\ell' \overset{c}{\rightsquigarrow} \ell$ for some $\ell' \not\sqsubseteq \ell$, then the system will not be noninterfering at level ℓ. However, for any given location h, S will be noninterfering for h at any level ℓ such that $\ell \notin \{\ell' \mid (\tau, \ell') \in [\![\mathsf{pol}(h)]\!]_\epsilon\}$. (We write ϵ to denote the empty trace.)

Theorem 4.4: *If S is a policy-enforcing system, and h is any location such that $\mathsf{pol}(h)$ is condition independent, and ℓ is any lattice element such that $\ell \notin \{\ell' \mid (\tau, \ell') \in [\![\mathsf{pol}(h)]\!]_\epsilon\}$, then S is noninterfering at level ℓ for location h.*

The following lemma is key to the proof of this theorem, and links the execution of a policy-enforcing system with the semantics of policies. The proof can be found in Appendix B.

Lemma 4.5: *For a policy-enforcing system S, and condition-independent policy p_h, let $s_0 \dots s_k s_{k+1}$ and $s'_0 \dots s'_k$ be two traces such that for all locations x, if $s_k(x) \neq s'_k(x)$ then $p_h \leq_{\epsilon \preceq s_0 \dots s_k} \mathsf{pol}(x)$. Then there exists an s'_{k+1} such that $s'_k \to s'_{k+1}$ and for all locations x, if $s_{k+1}(x) \neq s'_{k+1}(x)$ then $p_h \leq_{\epsilon \preceq s_0 \dots s_{k+1}} \mathsf{pol}(x)$.*

Proof of Theorem 4.4: Let h be a location such that $\mathsf{pol}(h)$ is condition independent, and ℓ be a lattice element such that $\ell \notin \{\ell' \mid (\tau, \ell') \in [\![\mathsf{pol}(h)]\!]_\epsilon\}$. Let v_1 and v_2 be two values, and s a state such that both $s_0 = s[h \mapsto v_1]$ and $s'_0 = s[h \mapsto v_2]$ are feasible. Let $s_0 \dots s_k$ be a trace of S.

By induction on k using Lemma 4.5, we can construct a trace $s'_0 \dots s'_k$ such that for any location x such that $s_k(x) \neq s'_k(x)$, then $\mathsf{pol}(h) \leq_{\epsilon \preceq s_0 \dots s_k} \mathsf{pol}(x)$.

Let x be any location such that $s_k(x) \neq s'_k(x)$, and thus $\mathsf{pol}(h) \leq_{\epsilon \preceq s_0 \dots s_k} \mathsf{pol}(x)$. By Property 3.7, we have $(s_0 \dots s_k, \mathsf{obs}(\mathsf{pol}(x))) \in [\![\mathsf{pol}(h)]\!]_\epsilon$. Thus, by Property 3.3, $\mathsf{obs}(\mathsf{pol}(x)) \not\sqsubseteq \ell$, and so $[s_k]^{\mathcal{L}}_{\ell} = [s'_k]^{\mathcal{L}}_{\ell}$. ∎

Note that Theorem 4.4 allows us to conclude that if ℓ is the policy enforced on location h, then S is noninterfering for h at any level ℓ' such that $\ell \not\sqsubseteq \ell'$. This means that even though there may be erasure and declassification occurring in the policy-enforcing system S, information stored in a location with policy ℓ will never be observable below ℓ.

4.3. Noninterference according to policy p

In this section we present a semantic security condition that depends upon the policy p of the input location. The new semantic security condition is called *noninterference according to policy p*, and gives a relatively detailed description of the information flow behavior of a policy-enforcing system, permitting fine-grained reasoning about the information flow in different executions of such a system. Noninterference according to policy p is defined in terms of the semantics of p, $[\![p]\!]_\epsilon$.

Definition 4.6: A system is *noninterfering according to policy p* if for any location h such that $\mathsf{pol}(h) = p$, any two values v_1 and v_2, and any state s such that both $s_0 = s[h \mapsto v_1]$ and $s'_0 = s[h \mapsto v_2]$ are feasible, then for any trace $s_0 \dots s_k$ there is a trace $s'_0 \dots s'_k$ such that for all lattice levels ℓ, if $(s_0 \dots s_k, \ell) \notin [\![p]\!]_\epsilon$, then $[s_k]^{\mathcal{L}}_{\ell} = [s'_k]^{\mathcal{L}}_{\ell}$.

Policy-enforcing systems are noninterfering according to policy p, as stated by the following theorem.

Theorem 4.7: *If S is a policy-enforcing system, then for all condition-independent policies p, S is noninterfering according to policy p.*

Proof: Let h be a location such that $\mathsf{pol}(h)$ is condition independent. Let v_1 and v_2 be two values, and s a state such that both $s_0 = s[h \mapsto v_1]$ and $s'_0 = s[h \mapsto v_2]$ are feasible. Let $s_0 \dots s_k$ be a trace of S. By induction on k using Lemma 4.5, we can construct a trace $s'_0 \dots s'_k$ such that for any location x such that $s_k(x) \neq s'_k(x)$, then $\mathsf{pol}(h) \leq_{\epsilon \preceq s_0 \dots s_k} \mathsf{pol}(x)$. By Property 3.7, $(s_0 \dots s_k, \mathsf{obs}(\mathsf{pol}(x))) \in [\![p]\!]_\epsilon$. Thus, for any lattice level ℓ such that $(s_0 \dots s_k, \ell) \notin [\![p]\!]_\epsilon$, by Property 3.3, we have $\mathsf{obs}(\mathsf{pol}(x)) \not\sqsubseteq \ell$, and so $[s_k]^{\mathcal{L}}_{\ell} = [s'_k]^{\mathcal{L}}_{\ell}$. ∎

For a given trace τ, determining the set $\{\ell \mid (\tau, \ell) \in [\![p]\!]_\epsilon\}$ depends only on the sequences of conditions that are satisfied at each step of the trace. Thus, we can give an equivalent statement of noninterference according to policy p simply in terms of sequences of satisfied conditions.

First, we define $\mathsf{conds}(s_0 \dots s_k, p)$, which takes a trace $s_0 \dots s_k$ and a policy p, and returns a sequence of sets of conditions, such that the ith set consists of the conditions occurring in p that are satisfied by the trace $s_0 \dots s_i$. More formally, $\mathsf{conds}(s_0 \dots s_k, p) = C_0 \dots C_k$ where for $i \in 0..k$, $C_i = \{c \mid c \text{ occurs in } p \text{ and } s_0 \dots s_i \vDash c\}$. It turns out that we can remove any empty sets from the sequence without any modification to the results.

Figure 5 defines a function $\mathsf{lvl}(p, C_0 \dots C_k)$ that takes a policy p and a sequence of sets of satisfied conditions, and

returns a lower bound on the set $\{\ell \mid (\tau, \ell) \in [\![p]\!]_\epsilon\}$, where $\mathsf{conds}(\tau, p) = C_0 \ldots C_k$. The fact that $\mathsf{lvl}(p, C_0 \ldots C_k)$ returns a lower bound is made precise by the following property, which would allow an equivalent statement of noninterference according to p in terms of $\mathsf{lvl}(p, \mathsf{conds}(\tau, p))$, with no reference to the semantics of p at all.

$$\mathsf{lvl}(\ell, C_0 \ldots C_k) = \ell$$

$$\mathsf{lvl}(p \overset{c}{\rightsquigarrow} p', C_0 \ldots C_k) = \mathsf{lvl}(p, C_0 \ldots C_k) \sqcap \bigsqcap \{\mathsf{lvl}(p', C_i \ldots C_k) \mid c \in C_i\}$$

$$\mathsf{lvl}(p \not\overset{c}{\rightsquigarrow} p', C_0 \ldots C_k) = \begin{cases} \mathsf{lvl}(p, C_0 \ldots C_k) & \text{if } \forall i \in 1..k.\ c \notin C_i \\ \mathsf{lvl}(p, C_0 \ldots C_k) \sqcup & \text{otherwise} \\ \quad \bigsqcup \{\mathsf{lvl}(p', C_{i+1} \ldots C_k) \mid \forall j \in 1..i.\ c \notin C_j\} & \end{cases}$$

Figure 5. Definition of $\mathsf{lvl}(p, C_0 \ldots C_k)$

Property 4.8: *For any policy p and trace τ, $(\tau, \ell) \in [\![p]\!]_\epsilon$ if and only if $\mathsf{lvl}(p, \mathsf{conds}(\tau, p)) \sqsubseteq \ell$.*

Noninterference according to policy p allows fine-grained reasoning about the end-to-end information flow behavior of a system, even in the presence of declassification and erasure. The equivalent statement of noninterference according to p allows us to reason about the behavior of a system solely in terms of sequences of satisfied conditions. For example, consider a policy-enforcing system whose input has the policy $H \overset{d}{\rightsquigarrow} (L \not\overset{c}{\rightsquigarrow} H)$ for a simple two point lattice where $L \sqsubseteq H$ and $L \neq H$. By considering possible sequences of satisfied conditions, we can make some strong statements about the information flow behavior of the system. Any trace of the system in which the condition d is never satisfied will reveal no information about the input to an attacker who can observe only at level L, since $\mathsf{lvl}(H \overset{d}{\rightsquigarrow} (L \not\overset{c}{\rightsquigarrow} H), C_0 \ldots C_k) = H$, where $d \notin C_i$ for all $i \in 0..k$. This seems a reasonable claim, because if d is never satisfied, no declassification of the input may occur. Similarly, we can see that for any trace in which the condition d is never satisfied after the condition c is, no information about the input is available at the end of the trace to an L-attacker. This result is more interesting: despite the fact that information about the input is declassified and observable at level L during the execution, by the end of the trace, no information about the input is available at level L.

Noninterference according to policy p allows reasoning about the interaction between declassification and erasure, resulting in stronger security guarantees than can be achieved in the absence of information erasure.

Noninterference according to policy p generalizes a useful semantic security condition *noninterference until declassification* [7], and its equivalent for erasure policies, *noninterference after erasure*.

4.4. Noninterference after erasure

Noninterference until declassification [7] is a security property that ensures an attacker cannot observe any information about a secret input until an appropriate sequence of declassifications has occurred. In the presence of information erasure, there is a corresponding semantic security condition for erasure: *noninterference after erasure*. Intuitively, after an appropriate sequence of erasures have occurred on some input data, an attacker should not be able to view any information about the input. Unlike noninterference until declassification, where more information about the input becomes available as time progresses, the system holds *less* information about the input as the appropriate erasures occur. Noninterference after erasure provides a useful security guarantee for privacy and anonymity concerns, where we would like to ensure that certain information is not retained by a system.

The definition of noninterference after erasure closely parallels that of noninterference until declassification. We write $\tau \models c_1 \ldots c_m$ when there is a non-decreasing sequence of natural numbers $n_1 \ldots n_m$ such that $\mathsf{conds}(\tau, p) = C_1 \ldots C_k$ and $c_i \in C_{n_i}$ for $i \in 1..m$.

Definition 4.9: A system is *noninterfering at security level ℓ after conditions $c_1 \ldots c_m$* for location h such that $\mathsf{pol}(h) = \ell_1\ {}^{c_1}\!\not\rightsquigarrow (\ell_2\ {}^{c_2}\!\not\rightsquigarrow (\cdots\ {}^{c_{k-2}}\!\not\rightsquigarrow (\ell_{k-1}\ {}^{c_{k-1}}\!\not\rightsquigarrow \ell_k) \cdots)$, where $m < k$, if for any two values v_1 and v_2, and any state s such that both $s_0 = s[h \mapsto v_1]$ and $s'_0 = s[h \mapsto v_2]$ are feasible, if $\tau_1 = s_0 \ldots s_k$ is a trace such that $\tau_1 \models c_1 \ldots c_m$ then there is a trace $\tau_2 = s'_0 \ldots s'_k$ such that $\tau_2 \models c_1 \ldots c_m$ and $[s_k]^{\mathcal{L}}_\ell = [s'_k]^{\mathcal{L}}_\ell$.

Theorem 4.10: *If S is a policy-enforcing system, then for any condition-independent policy $p \equiv \ell_1\ {}^{c_1}\!\not\rightsquigarrow (\ell_2\ {}^{c_2}\!\not\rightsquigarrow (\cdots\ {}^{c_{k-2}}\!\not\rightsquigarrow (\ell_{k-1}\ {}^{c_{k-1}}\!\not\rightsquigarrow \ell_k) \cdots)$, any location h such that $\mathsf{pol}(h) = p$, and any security level ℓ such that, $\ell_1 \sqcup \ldots \sqcup \ell_{m+1} \not\sqsubseteq \ell$, for $m < k$, then S is noninterfering at security level ℓ after conditions $c_1 \ldots c_m$ for location h.*

Proof: For any trace τ such that $\tau \models c_1 \ldots c_m$, we have $\ell_1 \sqcup \ldots \sqcup \ell_{m+1} \sqsubseteq \mathsf{lvl}(p, \mathsf{conds}(\tau, p))$. The result follows from Theorem 4.7. ∎

4.5. Robustness

Robust declassification [43, 30, 42] is a semantic security condition that restricts what information an active attacker may obtain from a system that declassifies information. In particular, a system is robust if an active attacker

(who can modify low-integrity aspects of the system and observe its execution), is unable to learn more than a passive attacker (who can only observe the system's execution).

In a language-based setting [42, 30], the robustness condition is enforced by ensuring that the decision to declassify information (as well as the information to be declassified) is trusted. This trust requirement can be treated as a declassification condition in the policy language.

With the introduction of information erasure, it is natural to ask whether there is a corresponding "robust erasure" semantic security condition? Interestingly, the semantic security condition is the same: performing modifications to low-integrity parts of the system should not reveal more information to the active attacker than to a passive attacker. Thus, an active attacker should not be able either to cause more information to be declassified, or to prevent the erasure of information. Therefore, robustness is a more general property that applies to information flow policies in general, not just to declassification. In a language-based setting, robustness would be enforced by ensuring that the decision to erase data is trusted, and cannot be subverted by the attacker: any erasure that should occur in the absence of an active attacker should also occur in the presence of the attacker.

Investigating the language-based enforcement of erasure policies, and determining suitable conditions on erasure policies to ensure robustness remains future work.

4.6. Selective declassification and erasure

Selective declassification [32] was introduced as part of the decentralized label model [28, 29], and requires the owners of data to authorize all declassifications of that data. Which owners are required to give their authorization for a given declassification depends on what security levels the data is being declassified from and to. Pottier and Conchon [32] present selective declassification as a combination of information flow and access control, where a number of declassification operations are locked at appropriate levels of authority; access control allows only suitably authorized principals to unlock the declassification operations, and only unlocked declassification operations can declassify information. Selective declassification, like robust declassification, attempts to prevent inappropriate declassifications by requiring a certain condition to be true when declassification occurs. Like robust declassification, such a condition can be incorporated into this policy framework.

In the presence of information erasure, there is a corresponding concept of selective erasure: all owners of data must give their authorization for the *non*-erasure of data. That is, information that is meant to be erased is only permitted to exist provided all owners of that information permit its existence. In a language-based setting where owners authorize code, a system needs to ensure that information is erased before control leaves the authorized code. It is not clear how to characterize selective erasure as a combination of access control and information flow.

5. Related work

As far as we are aware, no previous work has addressed information erasure from an information flow perspective, nor considered language-based enforcement of erasure policies. Some work has considered the secure deletion (or non-deletion) of information from magnetic disks and semiconductor devices (e.g., [18, 19, 2]), but this work is not at a language-level of abstraction, nor does it consider strong end-to-end security properties. Revocation of access rights is a form of erasure for access control; some work has investigated automatic revocation based on temporal constraints (e.g., [5, 21]).

There has however been much recent work on semantic security properties that hold in the presence of downgrading of security policies. The most relevant (noninterference until declassification, robust declassification, and selective declassification) have already been discussed in Section 4. A recent paper by Sabelfeld and Sands [37] provides a good survey of semantic security definitions for systems that perform declassification, as well as suggesting some guiding principles for declassification mechanisms.

Li and Zdancewic [22] present a framework of security policies that enforce the end-to-end semantic security condition *relaxed noninterference*. Their security policies express *what* information can be declassified, by specifying functions on secret data whose result is permitted to be non-secret. Their work is largely orthogonal to the declassification policies of this framework, which focus primarily on expressing *when* information can be released, rather than on fine-grained specifications of what information may be released. Supporting reasoning about relaxed noninterference would be an interesting extension to the security policies presented here.

Sabelfeld and Myers [36] present *delimited release*, a semantic security condition that allows reasoning about noninterference in the presence of declassification through "escape hatch" expressions. Like relaxed noninterference, delimited release allows a precise specification of *what* information may be declassified, by specifying computations on secret data whose result is non-secret.

Giacobazzi and Mastroeni [15] generalize noninterference by making it parametric with respect to what information an attacker can analyze about the input and output of a program. As such, *abstract noninterference*, like relaxed noninterference and delimited release, focuses on *what* information can be declassified.

Cuppens and Gabillon [10] consider the problem of temporal downgrading rules in a multi-level database. They present a language, based on a modal first order logic, that captures the semantics of temporal databases, and permits the specification of downgrading rules; their downgrading rules are expressive, permitting the specification of downgrading at a specific time, after a delay, or on a certain event (such as a user explicitly requesting to downgrade the information).

Intransitive noninterference [35, 31, 34] is an information flow condition based on noninterference that describes the behavior of systems that declassify information. While intransitive noninterference does not address information erasure, there is a close connection between it and the enforcement of the $p \leq_\tau q$ relationship. In fact, declassification policies are an extension of intransitive noninterference with temporal properties: in each computation step, information flows between levels only if that flow is permitted *and* appropriate conditions are true for that computation step [7].

Recent work by Mantel and Sands [25] places intransitive noninterference in a language setting, providing a bisimulation-based security condition for multi-threaded programs that controls where information can be declassified, and a type system that enforces this condition.

Some other approaches to reasoning about declassification in an information flow setting, such as *quantitative information flow* (e.g., [26, 23, 13, 9]) and *relative secrecy* [40] seek to measure or bound the *amount* of information that is declassified. This work is largely orthogonal to the declassification policies of this paper, which (in this context) are concerned only with possibilistic security assurances.

Zheng and Myers [44] show that noninterference can be achieved in the presence of dynamic labels. Dynamic labels have a close connection to declassification and erasure policies, since the conditions for declassification and erasure may depend on runtime data. In particular, both control the security policies of data at runtime, and may themselves depend on runtime data, and thus, both may be used to modulate covert channels. Most of the semantic security conditions of Section 4 require the policy of the input location to be condition-independent; it should be possible to use Zheng and Myers' techniques for reasoning about and controlling information flow from dynamic labels to prove noninterference results that hold even when the policy of the input location is not condition independent.

The *decentralized label model* [28, 29] is a security policy framework that permits mutually distrusting owners of information to specify who is permitted to read that information; only information owners may declassify the information they own. The decentralized labels form a lattice, which can be used as the base lattice of the security policies of this paper. Recent work has generalized decentralized labels to *owned policies* [6]; the security policies of this paper (instantiated with a base lattice of sets of readers) could be used as the policies that are owned by security principals.

The use of conditions to determine when declassification is permitted and erasure required adds a temporal element to the information security policies. As such, there is a connection between the policies of this paper and temporal logics, such as LTL [24] and CTL [8]. In particular, if information has a declassification policy $p \overset{c}{\rightsquigarrow} q$ enforced on it, then a policy-enforcing system ensures that at all times, if the information is declassified from p to q, then the condition c is true. (The condition c could itself be a temporal logic formula, if the framework is so instantiated.) Given sufficient predicates to reason about declassification, this guarantee could be formally stated in a temporal logic. Similarly, if information has an erasure policy $p \not\nearrow q$ enforced on it, then a policy-enforcing system ensures that at all times, if c is true then the information is either removed from the system, or has both p and q enforced on it. Again, given sufficient predicates, this guarantee could be formally stated in a temporal logic. Barthe, D'Argenio and Rezk [4] use the technique of *self-composition* to state noninterference as a temporal logic formula; the same technique may allow noninterference according to policy p to be stated as a temporal logic formula.

6. Conclusion

There has been a great deal of work on enriching information flow policies to support information release, but we are not aware of any prior work on information erasure, even though erasure policies appear to be an important aspect of information security requirements. This paper presents a framework for strong erasure policies, including support for both declassification and erasure.

The policy language allows the specification of policies that combine lattice levels, declassification, and erasure in complex ways. The may-flow relation supports static or dynamic reasoning about flows of information annotated with the policies. We have also given a formal semantics to these policies and shown that this semantics is consistent with the may-flow relation and a notion of observational level. A formal definition of what it means for a trace-based system to enforce a policy has been given; this definition then makes it possible to show that any policy-enforcing system satisfies various useful generalizations of noninterference.

Basing information security on information flow policies offers the promise of strong, end-to-end security assurance. However, information flow policies need to be much more expressive to capture the security requirements of real systems. In fact, this work was motivated by an attempt to cap-

ture the security requirements of a web-based voting system. This paper makes a step toward greater expressiveness, but much work remains to be done; one obvious next step is to develop enforcement mechanisms for erasure.

Acknowledgments

Thanks to Nate Nystrom and the anonymous reviewers for providing helpful feedback.

References

[1] J. Agat. Transforming out timing leaks. In *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL)*, pages 40–53, Boston, MA, Jan. 2000.

[2] R. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. In *IWSP: International Workshop on Security Protocols, LNCS*, 1997.

[3] A. Banerjee and D. A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *IEEE Computer Security Foundations Workshop (CSFW)*, June 2002.

[4] G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *Proc. 17th IEEE Computer Security Foundations Workshop*, pages 100–114, June 2004.

[5] E. Bertino, C. Bettini, and P. Samarati. A temporal authorization model. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 126–135, 1994.

[6] H. Chen and S. Chong. Owned policies for information security. In *Proc. 17th IEEE Computer Security Foundations Workshop*, June 2004.

[7] S. Chong and A. C. Myers. Security policies for downgrading. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 198–209, Oct. 2004.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Belief in information flow. In *Proc. 18th IEEE Computer Security Foundations Workshop*, June 2005.

[10] F. Cuppens and A. Gabillon. Modelling a multilevel database with temporal downgrading functionalities. In *Proceedings of the ninth annual IFIP TC11 WG11.3 working conference on Database security IX : status and prospects*, pages 145–164, 1996.

[11] D. E. Denning. A lattice model of secure information flow. *Comm. of the ACM*, 19(5):236–243, 1976.

[12] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.

[13] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate noninterference. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 1–15, June 2002.

[14] D. Gay and A. Aiken. Memory management with explicit regions. In *Proc. of the '98 SIGPLAN Conference on Programming Language Design*, pages 313–323. ACM Press, 1998.

[15] R. Giacobazzi and I. Mastroeni. Abstract non-interference: parameterizing non-interference by abstract interpretation. In *POPL31*, pages 186–197. ACM Press, 2004.

[16] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.

[17] D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney. Region-based memory management in cyclone. In *Proc. of the '02 SIGPLAN Conference on Programming Language Design*, pages 282–293. ACM Press, 2002.

[18] P. Gutmann. Secure deletion of data from magnetic and solid-state memory. In *The Sixth USENIX Security Symposium Proceedings*, pages 77–90, 1996.

[19] P. Gutmann. Data remanence in semiconductor devices. In *The Tenth USENIX Security Symposium Proceedings*, pages 39–54, 2001.

[20] N. Heintze and J. G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Proc. 25th ACM Symp. on Principles of Programming Languages (POPL)*, pages 365–377, San Diego, California, Jan. 1998.

[21] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. Generalized temporal role based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1), Jan. 2005.

[22] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *POPL32*, Long Beach, CA, Jan. 2005.

[23] G. Lowe. Quantifying information flow. In *Proc. 15th IEEE Computer Security Foundations Workshop*, June 2002.

[24] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[25] H. Mantel and D. Sands. Controlled Declassification based on Intransitive Noninterference. In *Proceedings of the 2nd ASIAN Symposium on Programming Languages and Systems, APLAS 2004*, LNCS 3303, pages 129–145, Taipei, Taiwan, Nov. 2004. Springer-Verlag.

[26] J. K. Millen. Covert channel capacity. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1987.

[27] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. 26th ACM Symp. on Principles of Programming Languages (POPL)*, pages 228–241, San Antonio, TX, Jan. 1999.

[28] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *Proc. 17th ACM Symp. on Operating System Principles (SOSP)*, pages 129–142, Saint-Malo, France, 1997.

[29] A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *Proc. IEEE Symposium on Security and Privacy*, pages 186–197, Oakland, CA, USA, May 1998.

[30] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proc. 17th IEEE Computer Security Foundations Workshop*, pages 172–186, June 2004.

[31] S. Pinsky. Absorbing covers and intransitive non-interference. In *Proc. IEEE Symposium on Security and Privacy*, pages 102–113, 1995.

[32] F. Pottier and S. Conchon. Information flow inference for free. In *Proc. 5nd ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 46–57, 2000.

[33] F. Pottier and V. Simonet. Information flow inference for ML. In *Proc. 29th ACM Symp. on Principles of Programming Languages (POPL)*, pages 319–330, 2002.

[34] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proc. 12th IEEE Computer Security Foundations Workshop*, 1999.

[35] J. Rushby. Noninterference, transitivity and channel-control security policies. Technical Report CSL-92-02, SRI, Dec. 1992.

[36] A. Sabelfeld and A. C. Myers. A model for delimited release. In *Proceedings of the 2003 International Symposium on Software Security*, number 3233 in Lecture Notes in Computer Science, pages 174–191. Springer-Verlag, 2004.

[37] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. 18th IEEE Computer Security Foundations Workshop*, June 2005.

[38] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

[39] D. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of the 7th International Joint Conference on the Theory and Practice of Software Development*, pages 607–621, 1997.

[40] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL)*, pages 268–276, Boston, MA, Jan. 2000.

[41] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[42] S. Zdancewic. A type system for robust declassification. In *Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics*, Electronic Notes in Theoretical Computer Science, Mar. 2003.

[43] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 15–23, Cape Breton, Nova Scotia, Canada, June 2001.

[44] L. Zheng and A. C. Myers. Dynamic security labels and noninterference. In *Proc. 2nd Workshop on Formal Aspects in Security and Trust, IFIP TC1 WG1.7*. Springer, Aug. 2004.

A. Proof of Theorem 3.4

Proof: By induction on the judgment $p \leq_\tau q$. The inductive hypothesis is that for any premise of the form $p' \leq_{\tau'} q'$, we have $[\![p']\!]_{\tau'} \supseteq [\![q']\!]_{\tau'}$.

(MF-LATTICE), (MF-TRANS). Trivial.

(MF-DECL). Here $p \equiv p'' \overset{c}{\rightsquigarrow} q$, and $\tau \vDash c$. We have $[\![p'' \overset{c}{\rightsquigarrow} q]\!]_\tau = [\![p'']\!]_\tau \cup \{(\tau'', \ell) \in [\![q]\!]_{\tau'} \mid \tau' \succeq \tau \text{ and } \tau' \vDash c\} \supseteq [\![q]\!]_\tau$, since $\tau \vDash c$.

(MF-DECL-I). Here $q \equiv q' \overset{d}{\rightsquigarrow} q''$, and $p \leq_\tau q'$, and $p \leq_\tau q''$, and for all $\tau' \succeq \tau$, if $\tau' \vDash d$ then q'' is valid for all traces between τ and τ'. We have $[\![q' \overset{d}{\rightsquigarrow} q'']\!]_\tau = [\![q']\!]_\tau \cup \{(\tau'', \ell) \in [\![q'']\!]_{\tau'} \mid \tau' \succeq \tau \text{ and } \tau' \vDash d\}$. By the inductive hypothesis, we have $[\![p]\!]_\tau \supseteq [\![q']\!]_\tau$, and $[\![p]\!]_\tau \supseteq [\![q'']\!]_\tau$. Now, if $(\tau'', \ell) \in [\![q'']\!]_{\tau'}$ for some τ' extending τ such that $\tau' \vDash d$, then we know that $\mathsf{valid}(q'', [\tau, \tau'))$, and thus, by Property 3.2 $[\![q'']\!]_\tau \supseteq [\![q'']\!]_{\tau'}$. Therefore, $[\![p]\!]_\tau \supseteq [\![q'']\!]_{\tau'}$, and so $(\tau'', \ell) \in [\![p]\!]_\tau$, and $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$ as required.

(MF-DECL-E). Here $p \equiv q \overset{c}{\rightsquigarrow} p'$. Clearly, $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$.

(MF-DECL-DECL). Here $p \equiv p' \overset{c}{\rightsquigarrow} p''$ and $q \equiv q' \overset{d}{\rightsquigarrow} q''$, and for all $\tau' \succeq \tau$, if $\tau' \vDash d$ then $\tau' \vDash c$ and $p'' \leq_{\tau'} q''$. By the inductive hypothesis, we have $[\![p']\!]_\tau \supseteq [\![q']\!]_\tau$. Also, for any $\tau' \succeq \tau$ such that $\tau' \vDash d$, we have $\tau' \vDash c$, and $p'' \leq_{\tau'} q''$, so by the inductive hypothesis, $[\![p'']\!]_{\tau'} \supseteq [\![q'']\!]_{\tau'}$. So we have $[\![p' \overset{c}{\rightsquigarrow} p'']\!]_\tau \supseteq [\![q' \overset{d}{\rightsquigarrow} q'']\!]_\tau$.

(MF-ERASE-E). Here $p \equiv p' \overset{c}{\nearrow} p''$, and, by the inductive hypothesis, $[\![p']\!]_\tau \supseteq [\![q]\!]_\tau$ and $[\![p'']\!]_\tau \supseteq [\![q]\!]_\tau$. Thus $[\![p']\!]_\tau \cap [\![p'']\!]_\tau \supseteq [\![q]\!]_\tau$. Since $\forall tau''.\ \tau \succ \tau'' \succeq \tau \Rightarrow \tau'' \nvDash c$ is trivially true, we have $[\![p]\!]_\tau \supseteq [\![p']\!]_\tau \cap [\![p'']\!]_\tau$, so $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$ as required.

(MF-ERASE-I). Here $q \equiv p \overset{d}{\nearrow} q'$. Clearly, $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$.

(MF-ERASE-ERASE). Here $p \equiv p' \overset{c}{\nearrow} p''$ and $q \equiv q' \overset{d}{\nearrow} q''$. By the inductive hypothesis, we have $[\![p']\!]_\tau \supseteq [\![q']\!]_\tau$ and for all $\tau' \succeq \tau$, if $\tau' \vDash c$ then $\tau' \vDash d$ and $[\![p'']\!]_{\tau'} \supseteq [\![q'']\!]_{\tau'}$. Conversely, for all τ' such that $\tau' \nvDash d$ we have $\tau' \nvDash c$, and thus for any τ' such that for all τ'', $\tau' \succ \tau'' \succeq \tau \Rightarrow \tau'' \nvDash d$ we have $[\![p'']\!]_{\tau'} \supseteq [\![q'']\!]_{\tau'}$. Thus $[\![p]\!]_\tau \supseteq [\![q]\!]_\tau$ as required. ∎

B. Proof of Lemma 4.5

Let S be a policy-enforcing system and p_h be a condition-independent policy. Let $s_0 \ldots s_k s_{k+1}$ and $s'_0 \ldots s'_k$ be two traces such that for all locations x, if $s_k(x) \neq s'_k(x)$ then $p_h \leq_{\epsilon \preceq s_0 \ldots s_k} \mathsf{pol}(x)$. Since S is policy enforcing, there is a state s'_{k+1} such that $s'_k \rightarrow s'_{k+1}$, such that for all locations x, if $s_{k+1}(x) \neq s'_{k+1}(x)$ then there is a location y such that $s_k(y) \neq s'_k(y)$ and $\mathsf{pol}(y) \leq_{s_0 \ldots s_{k+1}} \mathsf{pol}(x)$.

Suppose there is some location x such that $s_{k+1}(x) \neq s'_{k+1}(x)$ and $p_h \not\leq_{\epsilon \preceq s_0 \ldots s_{k+1}} \mathsf{pol}(x)$. Then there is a location y such that $s_k(y) \neq s'_k(y)$ and $\mathsf{pol}(y) \leq_{s_0 \ldots s_{k+1}} \mathsf{pol}(x)$. Since p_h is condition independent, $\mathsf{valid}(\mathsf{pol}(x), s_0 \ldots s_{k+1})$ if and only if $\mathsf{valid}(\mathsf{pol}(x), s'_0 \ldots s'_{k+1})$, and since $s_{k+1}(x) \neq s'_{k+1}(x)$, it must be the case that $\mathsf{valid}(\mathsf{pol}(x), s_0 \ldots s_{k+1})$. Similarly, it must be the case that $\mathsf{valid}(\mathsf{pol}(y), s_0 \ldots s_k)$. Therefore, $p_h \leq_{\epsilon \preceq s_0 \ldots s_{k+1}} \mathsf{pol}(x)$, a contradiction.

Therefore, for all locations x, if $s_{k+1}(x) \neq s'_{k+1}(x)$ then $p_h \leq_{\epsilon \preceq s_0 \ldots s_{k+1}} \mathsf{pol}(x)$. ∎