A Theory of Information-Flow Labels

Benoît Montagu[1] Benjamin C. Pierce[1] Randy Pollack[2] Adrien Surée[3]

[1]University of Pennsylvania, [2]Harvard University [3]École Normale Supérieure

Abstract

Each one of the numerous calculi, languages, and systems for information-flow control includes a tiny domain-specific language of "micro security policies" called *labels*. The question of how to compare the relative expressive power of these different notions of labels arises naturally, and the design of a common framework for label models is a first step in that direction. One familiar abstract label framework is a simple lattice structure, but this lacks a notion of *authority*, which is used in practice to relax the rules on flow of information under carefully controlled situations.

To study the design space of information-flow labels and their relative expressiveness, we introduce *label algebras*, an abstract interface for information-flow labels equipped with a notion of authority. We define a generic notion of semantics for label algebras in terms of observations by a family of observers, and we consider *embeddings*, semantics-preserving maps between label algebras. We give two instances of this generic notion, based respectively on the observation of values and of programs, and these give rise to two natural notions of embeddings. Using this framework, we define and compare a number of concrete label algebras, including realizations of the familiar *taint*, *endorsement*, *readers*, and *distrust* models and label algebras based on several existing programming languages and operating systems.

General Terms Security, Languages, Design, Theory

Keywords Information flow control (IFC), DIFC, label models, decentralized label model (DLM), JIF, LIO, disjunction category model, Flume, HiStar, Asbestos

1. Introduction

Information-flow control systems (IFC)[3] run the gamut from static type systems to run-time monitors and from core calculi to full-blown languages and operating systems. One critical component of each of these is a *label model*—a notation for writing down information-flow labels, together with rules for when one label *flows to* another in the sense that data labeled with the first is allowed to flow to contexts labeled with the second. These labels can be thought of as low-level "micro-policies." They do not directly describe system-level security policies that end users might care about ("my banking information will never be sent to `evil.com`"), but they capture information-flow invariants on specific sensitive values ("this integer and values derived from it should only be visible to the principal named *Bank*"), and these invariants can be used by programmers to reason about higher-level security properties.

Label models come in a bewildering variety of shapes and forms. A theoretical discussion might use a very simple model with just two or three labels ($\bot \sqsubseteq \top$ or $public \sqsubseteq secret \sqsubseteq topsecret$), or it might assume an arbitrary lattice of labels. Or, more concretely, we might define a label to be a set of *principals*, interpreting this set either as the set of entities that are allowed to read a given value or as the set of entities that trust or endorse it, or perhaps as the set of entities that may have tainted it. More complex systems use sets of sets, logical formulae, or other structures as labels. Some systems—e.g., those based on the Decentralized Label Model (DLM) [9]—include a notion of *policy owners*, distinct from readers, tainters, or endorsers, that controls not who can *use* particular values but who has the power to *change* the policy by *declassification*. Some systems focus on protecting secrecy, others integrity, and still others incorporate both. The list of possible variations seems endless.

Which label models are best for which purposes? What common structure might we expect *every* label model to have, beyond a simple flows-to ordering? Can all the label models used in real systems be viewed as instances of this common structure, or are there deep differences between them? What generic operations can be performed on arbitrary label models? What high-level properties (e.g., non-interference) are guaranteed by the structure of the label model? Can the common dictum that "integrity is formally dual to secrecy" be given a rigorous explanation? Is this label model "more expressive" than that one, in the sense that, given a program written in terms of this one, can we obtain a program in terms of that one (that behaves the same!) by rewriting labels in some systematic way? Such questions seem essential to a thorough understanding of IFC, but they are rarely discussed.

Our goal in this paper is to initiate the comparative study of label models by providing a concrete mathematical framework and investigating how it applies to label models found in the wild.

1. We define *label algebras*, a precise abstract characterization of the structure common to many label models, with an abstract notion of authority (§2). On top of this, we define a simple programming language—an untyped lambda-calculus with dynamic information-flow tracking and declassification—parametrized by an arbitrary label algebra (§3). We give a generic proof of (an authority-enriched generalization of) the standard non-interference property.
2. We propose a generic notion of *embedding* between label algebras, formalizing the intuition that one label model is "more expressive" than another (§4). As usual, judgments about expressiveness depend on the power of the observer; accordingly, we study two different concrete forms of embeddings: *value embeddings*, which preserve observations on labeled data structures, and *evaluation embeddings*, which also take into account the behavior of computations over labeled data. We prove *characterization theorems*, giving necessary and sufficient conditions for checking whether a given map between label algebras is an embedding of either form.
3. We define a number of concrete label algebras, including simple examples that illustrate dimensions of the design space of label models, realizations of the familiar *taint*, *endorsement*, *readers*, and *distrust* models (§5), and more complex examples based on real-world languages and operating systems (§6). In particular, we define label algebras corresponding to the DC model [13], the DLM [9] (with and without principal hierarchies), Asbestos [4],

HiStar [15, 16], Flume [5], and Laminar [10]. We settle the existence or non-existence of embeddings among all of the simple examples and some of the real-world ones—in particular, we show that (the secrecy parts of) the DLM with no principal hierarchy and the DC model cannot be embedded in each other (while, when authorities are not considered, embeddings between the underlying label lattices exist in both directions).

We survey related work in §7 and sketch directions for future work in §8.

There are two important caveats. First, our definition of label algebra covers just a small set of core features—labels, label ordering, authority, and defaults—omitting some of the interesting complexities associated with real-world label models. In particular, we do not address dynamic generation of principals and authorities. Second, since evaluation embeddings are defined in terms of computation, their properties necessarily depend on the details of the programming language under consideration; adding other features such as first-class labels will change some of our results. We leave a detailed investigation to future work.

2. Label Algebras

2.1 Basic definitions

Recall that a pre-lattice is a preorder with meet and join operations. There may be cycles in a preorder ($x \leq y$ and $y \leq x$ with $x \neq y$), but the cycles form classes of an equivalence relation, and the lifted order on these equivalence classes is a partial order. In a pre-lattice, this partial order is a lattice with the lifted meet and join operations.

2.1.1 Definition: A *label algebra* $\mathcal{M}$ comprises:

- a pre-lattice of *labels* $(\mathcal{L}, \sqsubseteq, \sqcap, \sqcup)$
- a lower-bounded join semilattice of *authorities* $(\mathcal{A}, \leq, \vee, 0)$,
- for each authority A, a *flows-to* relation on $\mathcal{L}$, $\sqsubseteq_A$, such that
  1. $\sqsubseteq_0 = \sqsubseteq$
  2. if $A \leq A'$ and $L_1 \sqsubseteq_A L_2$, then $L_1 \sqsubseteq_{A'} L_2$
  3. each $(\mathcal{L}, \sqsubseteq_A, \sqcap, \sqcup)$ is a pre-lattice
- a designated *default label* L^{def}.

We write $L_1 \equiv_A L_2$ when $L_1 \sqsubseteq_A L_2$ and $L_2 \sqsubseteq_A L_1$; we write $\equiv$ to denote $\equiv_0$.

Label algebras include a bounded join-semilattice of *authorities*. One can understand authorities as permissions to bend the rules, allowing more flows between labels.[1] The least (or empty) authority, written 0, carries no privilege: the relation $\sqsubseteq_0$ is exactly the flows-to relation of the underlying pre-lattice (axiom 1 about authorities).

The indexed flows-to relations are compatible with the ordering on authorities (axiom 2): increasing authority makes it easier to flow from one label to another [9, 13]. *Declassification* (or *downgrading*) is precisely the exercise of authority to permit a flow that would not otherwise be allowed. The 0-authority flows-to relation describes flows that are always allowed.

Axiom 3 requires that the join and meet operators of the underlying lattice remain least upper bound and greatest lower bound for the family of authority-sanctioned flows. This uniformity allows the following reasoning: if it is possible, using authority A, to declassify from L_1 to L and from L_2 to L, then it is possible to declassify from $L_1 \sqcup L_2$ to L using the same authority A.

Although many label models do have bottom and top labels, we do not impose this as a requirement.

The last bullet in the definition specifies that there should be a designated *default label*; the intention is that all data values should be annotated with this label—or perhaps an even higher (more restrictive) one—unless they have been downgraded. This is useful when labels are used for *endorsement*: by default, a data value starts out life endorsed by no one (i.e., with a high label), and its label gets lowered only by the explicit exercise of authority. In the evaluation semantics defined in §4.4, this is achieved by using the default label as the starting value of the *pc label* when a term is evaluated.

[1] It would be natural to consider a *pre*-semilattice of authorities. We believe this works, at the cost of somewhat more complex statements and proofs.

2.2 Examples

We will see many examples of label algebras in §5 and §6, but let's look at a few simple ones now.

Most label algebras are defined over some enumerable set of *principals*, written $\mathbb{P}$. We write p for specific principals, P for sets of principals, and $\mathcal{P}(\mathbb{P})$ for the set of sets of principals. We sometimes abuse notation and consider $\mathcal{P}(\mathbb{P})$ as a lattice, with intersection and union corresponding to meet and join. Similarly, $\mathcal{P}^{fin}(\mathbb{P})$ is the set or lattice whose elements are finite sets of principals. **1** is the unit lattice; its one element is written either $\bot$ or 0.

One very simple label algebra is the *public / secret* model, which we call **2** for short (it is also sometimes called the *binary* model [6]). Its set of labels is the two-point lattice, and it has only one authority.

2: Public / Secret Model

$$\mathcal{L} = \{\top, \bot\} \qquad \bot \sqsubseteq \top \qquad L^{def} = \bot \qquad \mathcal{A} = \mathbf{1}$$

A more interesting label algebra is the *reader model* (written **CR**, rather than just **R**, for uniformity with a group of related label algebras that we will encounter in §5.3; some related real-world models are discussed in §6).

CR: Reader Model

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad L^{def} = \mathbb{P}$$
$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad A_1 \leq A_2 = A_1 \subseteq A_2$$
$$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2$$

Its labels are either the full set of principals or one of its finite subsets, ordered by reverse inclusion. Intuitively, the principals in a label are the ones who *may read* some piece of data. Its default label is $\mathbb{P}$ (which is the bottom element of the label lattice)—i.e., anybody is allowed read data with the default label. A value labeled with some set of principals can freely be relabeled with a smaller set (fewer allowed readers)—in particular, in the labeled lambda-calculus in §3, it will always be legal to take a value with the default label and relabel it (restrict its readership) to some finite set of principals. Authorities are sets of principals, and an authority containing a principal p permits flows from labels not including p (i.e. data that p cannot read) to labels where p is allowed as a reader. For example, it will be legal to relabel a value labeled $\{q, r\}$ into one labeled $\{p, q, r\}$ using the authority $\{p, s\}$.

Another common label algebra is the *endorsement model* (**CE**). It differs from the reader model only in its default label, which is the top element. The principals in a label indicate who has *endorsed* some data value. By default, nobody endorses anything. Authorities are used to add endorsements.

CE: Endorsement Model

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad L^{def} = \emptyset$$
$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad A_1 \leq A_2 = A_1 \subseteq A_2$$
$$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2$$

2.3 Operations on label algebras

The space of label algebras is closed under some simple operations, including dualization and product; these can be useful for describing examples compactly.

2.3.1 Definition: Suppose $\mathcal{M}$ is a label algebra. Its *dual*, $\mathcal{M}^{op}$, is obtained by reversing the $\sqsubseteq_A$ relations:

$\mathcal{M}^{op}$: Dual of $\mathcal{M}$

$$\mathcal{L}^{op} = \mathcal{L} \qquad \mathcal{A}^{op} = \mathcal{A}$$
$$L_1 \sqsubseteq^{op} L_2 = L_2 \sqsubseteq L_1 \qquad \sqcup^{op} = \sqcap \qquad \sqcap^{op} = \sqcup$$
$$L_1 \sqsubseteq_A^{op} L_2 = L_2 \sqsubseteq_A L_1 \qquad (L^{def})^{op} = L^{def}$$

Note that this definition is essentially forced. Because authorities have no top element in general, we cannot invert the authority structure. Moreover, because we have no canonical way of choosing a "complement" for the default element, there is no choice but to keep the default the same.

2.3.2 Definition: Suppose $\mathcal{M}_1$ and $\mathcal{M}_2$ are two label algebras. We define their product $\mathcal{M}_1 \times \mathcal{M}_2$ as follows:

$\mathcal{M}_1 \times \mathcal{M}_2$: Product of $\mathcal{M}_1$ and $\mathcal{M}_2$

$$\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 \qquad \mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \qquad L^{def} = (L_1^{def}, L_2^{def})$$
$$(L_1, L_2) \sqsubseteq_{(A_1,A_2)} (L_1', L_2') = L_1 \sqsubseteq_{A_1} L_1' \text{ and } L_2 \sqsubseteq_{A_2} L_2'$$

$\mathcal{M}_1 \times \mathcal{M}_2$ is also a label algebra.

Real world systems often combine secrecy and integrity into a single model by taking a label algebra of the form $\mathcal{M} \times \mathcal{M}^{op}$ for some $\mathcal{M}$ (see §6).

We can also remove the authority part of an arbitrary label algebra. We will use this operation several times in §5 and §6.

2.3.3 Definition: Suppose $\mathcal{M}$ is a label algebra. We define its 0-*authority projection*, written $\mathcal{M}_0$, as follows:

$\mathcal{M}_0$: 0-authority projection of $\mathcal{M}$

$$\mathcal{L}_0 = \mathcal{L} \qquad \mathcal{A}_0 = \mathbf{1} \qquad L_1 \sqsubseteq_A^0 L_2 = L_1 \sqsubseteq L_2 \qquad L_{\mathbf{0}}^{def} = L^{def}$$

2.4 Label algebra maps

To define what it means for a label algebra to be more expressive than another one, we begin here with a very loose notion of maps between label algebras—we do not even require that they preserve any of the structure of a label algebra—and later (§4) give additional conditions they should enjoy to be considered as "good encodings," which we call *embeddings*.

2.4.1 Definition: Given two label algebras $\mathcal{M}_1 = (\mathcal{L}_1, \mathcal{A}_1, \ldots)$ and $\mathcal{M}_2 = (\mathcal{L}_2, \mathcal{A}_2, \ldots)$, a *label algebra map* m from $\mathcal{M}_1$ to $\mathcal{M}_2$, written $m \in \mathcal{M}_1 \to \mathcal{M}_2$, comprises

- a function (also written m) from $\mathcal{L}_1$ to $\mathcal{L}_2$, and
- an order-preserving function (also written m) from $\mathcal{A}_1$ to $\mathcal{A}_2$.

The reason why we require maps to preserve the order on authorities is that we want that a map between two label algebras preserves the inclusions of pre-lattices induced by axiom 2 of Definition 2.1.1.

3. Labeled Lambda-Calculus

We next define a programming language $\lambda_{\mathcal{M}}$ parametrized by a label algebra $\mathcal{M}$. For simplicity, we choose an untyped language with dynamic information-flow tracking, similar to that of [1].

3.1 Syntax and semantics

The syntax of $\lambda_{\mathcal{M}}$, together with the sets of values and labeled values (atoms), is shown in Figure 1. Its syntax comprises two constants, variables, λ-abstractions, applications and a construct $t \updownarrow_A L$ to *relabel* the result of the evaluation of t with the constant label L using authority A. Note that labels and authorities are not first-class: they can only occur in the relabeling construct.

The big-step operational semantics of $\lambda_{\mathcal{M}}$ is given in Figure 2. The evaluation judgment has the form $pc, \rho \vdash t \Downarrow a$. Evaluation

t	::=			terms
	|	tt		constant true
	|	ff		constant false
	|	x		variable
	|	$\lambda x.t$	bind x in t	abstraction
	|	$t_1\ t_2$		application
	|	$t_1 \updownarrow_A L_2$		relabel value
	|	(t)		
v	::=			values
	|	tt		true
	|	ff		false
	|	$\langle \rho,\ \lambda x.\ t \rangle$	bind x in t	closures
a	::=			atoms
	|	$v@L$		labelled values
ρ	::=			environments
	|	$\bullet$		empty
	|	$\rho, x = a$		binding
	|	(ρ)		

Figure 1. Syntax of terms, values, atoms and environments.

$pc, \rho \vdash t \Downarrow a$

$$\frac{}{pc, \rho \vdash tt \Downarrow tt@pc}\ \textsc{Eval\_True}$$

$$\frac{}{pc, \rho \vdash ff \Downarrow ff@pc}\ \textsc{Eval\_False}$$

$$\frac{\rho(x) = v@L}{pc, \rho \vdash x \Downarrow v@(pc \sqcup L)}\ \textsc{Eval\_Var}$$

$$\frac{}{pc, \rho \vdash (\lambda x.t) \Downarrow \langle \rho,\ \lambda x.\ t \rangle@pc}\ \textsc{Eval\_Abs}$$

$$\frac{pc, \rho \vdash t_1 \Downarrow \langle \rho_1,\ \lambda x.\ t \rangle@L_1 \quad pc, \rho \vdash t_2 \Downarrow a_2 \quad L_1, (\rho_1, x = a_2) \vdash t \Downarrow a_3}{pc, \rho \vdash (t_1\ t_2) \Downarrow a_3}\ \textsc{Eval\_App}$$

$$\frac{pc, \rho \vdash t_1 \Downarrow v_1@L_1 \quad L_1 \sqsubseteq_A L_2}{pc, \rho \vdash t_1 \updownarrow_A L_2 \Downarrow v_1@L_2}\ \textsc{Eval\_Relabel}$$

Figure 2. Big-step semantics.

produces *atoms*, denoted by a, which are labeled values. The environment ρ maps variables to closed atoms.

The label pc is the *program counter* label—the label of the control state of the program. In the variable lookup rule, the label on the variable's value from the environment is joined with the current pc label to form the label on the result, reflecting the fact that the choice to look up this variable (as opposed to another one, for example) may have been influenced by sensitive information. This detail is crucial for the non-interference theorem (3.2.4).

The reader may wonder why there is a construct for changing the label on a value but no analogous construct for changing the pc label. The reason is that it is encodable. The rule for function application replaces the pc with the label of the computed closure. This allows us to express pc relabeling as follows

$$t \updownarrow_A^{\text{pc}} L = ((\lambda x.t) \updownarrow_A L)\ tt$$

where x is fresh for t. If authority A permits it, $t \updownarrow^{\mathrm{pc}}_{A} L$ will change the current pc label to L while evaluating t.

3.1.1 Definition [Lifted label algebra maps]: Let m be a map from $\mathcal{M}_1$ to $\mathcal{M}_2$.

1. Define $\hat{m} \in \lambda_{\mathcal{M}_1} \to \lambda_{\mathcal{M}_2}$ as the term homomorphism that transforms label constants and authorities using m and copies everything else unchanged.
2. Define $\hat{m}$ from atoms, values, and environments of $\lambda_{\mathcal{M}_1}$ to atoms, values, and environments of $\lambda_{\mathcal{M}_2}$ as the pointwise extension of $\hat{m}$.

The following simple properties will be useful below for working with lifted maps.

3.1.2 Lemma:

1. $\widehat{id_{\mathcal{M}}} = id$, where $id_{\mathcal{M}}$ is the identity map from $\mathcal{M}$ to itself and id is the identity function on programs over $\mathcal{M}$.
2. If $m_{12} \in \mathcal{M}_1 \to \mathcal{M}_2$ and $m_{23} \in \mathcal{M}_2 \to \mathcal{M}_3$, then $\widehat{m_{23}} \circ \widehat{m_{12}} = \widehat{m_{23} \circ m_{12}}$.

3.2 Basic properties

We now establish some fundamental results about $\lambda_{\mathcal{M}}$—in particular, a standard non-interference property. This is mainly a sanity check on our labeled lambda-calculus: the only part of this development that is used in later sections is Definition 3.2.5. The rest can be skimmed on a first reading.

3.2.1 Definition: The *authority of a program* t, written $Auth(t)$, is the join of all the authorities that occur in t:

$$\begin{aligned} Auth(tt) = Auth(ff) = Auth(x) &= 0 \\ Auth(\lambda x.t) &= Auth(t) \\ Auth(t_1\ t_2) &= Auth(t_1) \vee Auth(t_2) \\ Auth(t \updownarrow_A L) &= Auth(t) \vee A \end{aligned}$$

This definition lifts naturally to atoms, values and environments.

The following technical lemma is used in the proof of non-interference. Intuitively, the first part says that authority only decreases during evaluation, while the second says that the label on the result of evaluating a term is always above the starting pc label (modulo any authority that the program might have used to downgrade the result). In particular, when $A = 0$, the final label is guaranteed to be above the starting pc in the label lattice.

3.2.2 Lemma:

1. If $pc, \rho \vdash t \Downarrow a$ then $Auth(a) \leq (Auth(\rho) \vee Auth(t))$.
2. If $pc, \rho \vdash t \Downarrow v@L$ and $Auth(\rho) \vee Auth(t) \leq A$ then $pc \sqsubseteq_A L$.

The relation $\approx^L_A$ expresses *equivalence* (or *indistinguishability*) for observers at label L and authority A.

3.2.3 Definition: $b_1@L_1 \approx^L_A b_2@L_2$ holds iff either

- $L_1 \not\sqsubseteq_A L$ and $L_2 \not\sqsubseteq_A L$, or
- $L_1 \sqsubseteq_A L$ and $L_1 \equiv L_2$ and $b_1 = b_2$.

Here is the main theorem of this section:

3.2.4 Theorem [Non-interference]: If

- $pc, \rho_1 \vdash t \Downarrow b_1@L_1$ and $pc, \rho_2 \vdash t \Downarrow b_2@L_2$,
- $\mathrm{dom}\,\rho_1 = \mathrm{dom}\,\rho_2$ and for any $x \in \mathrm{dom}\,\rho_1$, $\rho_1(x) \approx^L_A \rho_2(x)$,
- $Auth(t) \leq A$,

then $b_1@L_1 \approx^L_A b_2@L_2$.

For the proof of 3.2.4, we need to generalize the definition of labeled equivalence from booleans to atoms (3.2.5), values and environments, then prove a stronger version of non-interference (3.2.7). Most papers on dynamic information flow use a similar strategy.

3.2.5 Definition: The *labeled equivalence* for a label L and an authority A, written $\approx^L_A$, is the smallest binary relation such that...

- On atoms, $v_1@L_1 \approx^L_A v_2@L_2$ holds when
  - $L_1 \not\sqsubseteq_A L$ and $L_2 \not\sqsubseteq_A L$, or
  - $L_1 \sqsubseteq_A L$ and $L_1 \equiv L_2$ and $v_1 \approx^L_A v_2$.
- On values, $v_1 \approx^L_A v_2$ holds when
  - $v_1 = tt$ and $v_2 = tt$, or
  - $v_1 = ff$ and $v_2 = ff$, or
  - $v_1 = \langle \rho_1,\ \lambda x_1.\, t_1 \rangle$ and $v_2 = \langle \rho_2,\ \lambda x_2.\, t_2 \rangle$, with $\rho_1 \approx^L_A \rho_2$ and $\lambda x_1.t_1 = \lambda x_2.t_2$.
- On environments, $\rho_1 \approx^L_A \rho_2$ holds when ρ_1 and ρ_2 have the same domain, and $\rho_1(x) \approx^L_A \rho_2(x)$ for all $x \in \mathrm{dom}\,\rho_1$.

The next lemma says that an observer who cannot distinguish two objects will also be unable distinguish them when viewing from a lower label or with a lower authority.

3.2.6 Lemma:

1. $\approx^L_A$ is indeed an equivalence relation.
2. If $A_1 \leq A_2$, then $(a_1 \approx^L_{A_2} a_2 \implies a_1 \approx^L_{A_1} a_2)$.
3. $L_1 \sqsubseteq_A L_2$ iff $(a_1 \approx^{L_2}_A a_2 \implies a_1 \approx^{L_1}_A a_2)$.

Proof: (2) and direction $\implies$ of (3) are each proved by induction on $a_1 \approx^L_{A_2} a_2$ (including the mutual cases for values and environments). For direction $\impliedby$ of (3), Suppose $L_1 \not\sqsubseteq_A L_2$. Then we have $tt@L_1 \approx^{L_2}_A ff@L_1$ but not $tt@L_1 \approx^{L_1}_A ff@L_1$—a contradiction. □

Note that, while direction $\implies$ of (2) and (3) are basic to our intentions, direction $\impliedby$ of (3) could be viewed as an accidental consequence of the simple language we are using in this paper.

We can now state and prove a suitable generalization of the main non-interference theorem (3.2.4); the main theorem follows as a special case.

3.2.7 Lemma: If

- $pc_1, \rho_1 \vdash t \Downarrow a_1$ and $pc_2, \rho_2 \vdash t \Downarrow a_2$,
- $pc_1 \equiv pc_2$, and $\rho_1 \approx^L_A \rho_2$,
- $Auth(t) \vee Auth(\rho_1) \vee Auth(\rho_2) \leq A$,

then $a_1 \approx^L_A a_2$.

Proof: By induction, using Lemma 3.2.2. □

We close with one last property of $\lambda_{\mathcal{M}}$: the fact that increasing a program's authority does not change it's terminating behaviors (though it can cause a program that aborts with a security fault to succeed instead).

3.2.8 Definition [Authority-wise ordering on programs]:

$$\begin{array}{lcl} x_1 \leq x_2 & = & x_1 = x_2 \\ b_1 \leq b_2 & = & b_1 = b_2 \\ \lambda x.t_1 \leq \lambda x.t_2 & = & t_1 \leq t_2 \\ t_1\ t_1' \leq t_2\ t_2' & = & t_1 \leq t_2 \text{ and } t_1' \leq t_2' \\ t_1 \updownarrow_{A_1} L_1 \leq t_2 \updownarrow_{A_2} L_2 & = & t_1 \leq t_2 \text{ and } L_1 = L_2 \text{ and } A_1 \leq A_2 \end{array}$$

3.2.9 Lemma: If $t_1 \leq t_2$, then $Auth(t_1) \leq Auth(t_2)$.

3.2.10 Proposition: If $pc, \rho \vdash t_1 \Downarrow v_1@L_1$ and $t_1 \leq t_2$, then there exists $v_2@L_2$ such that $pc, \rho \vdash t_2 \Downarrow v_2@L_2$. Moreover, if v_1 is a boolean, then $v_1 = v_2$.

4. Semantics of Labels

Suppose someone asserts that "Label model $\mathcal{M}_2$ can encode label model $\mathcal{M}_1$." What, exactly, are they claiming? We can imagine two reasonable interpretations of such a statement. First, it might mean simply that the order structure of $\mathcal{M}_2$ is sufficiently rich that we can find an "image" of $\mathcal{M}_1$ inside it. This interpretation corresponds to

a view of labels as simply a way of annotating *data values* to tell which observers are allowed to look at them. Second, the statement might mean that *computations* involving the labels of $\mathcal{M}_1$ would behave the same if we somehow translated them to use labels from $\mathcal{M}_2$ instead.

The second interpretation is more interesting, since different languages may use labels in different ways. For example, in a language with information flow labels and references, writing a value with a high label into a reference cell with a low label will result in a security fault. (Formally, depending on the calculus this might be either a stuck state that halts the program or else a typing error that prevents it from being run in the first place.) Another information-flow system might include a notion of *clearance* that places a similar restriction on the *pc* label. Yet another might allow integrity labels to be queried programmatically to help decide whether a value is sufficiently trusted to be used in a certain way.

Thus, for this initial study we have chosen to compare label algebras in the setting of a single simple language. The labeled lambda-calculus of §3 includes only one construct (reclassification) that can cause program failure because of an information-flow fault: aside from this, there is no notion of "access control decisions" (can so-and-so *read* this value, do they *trust* it, etc.) within the language. Instead, the idea here (as in many other labeled lambda-calculi in the literature) is that access control decisions will be made externally: run a labeled program, and when the program terminates give its result to an observer, who then may or may not be able to do anything with it, depending on the result's label and the observer's power to distinguish (also represented as a label).

Both interpretations of statements like "$\mathcal{M}_2$ can encode $\mathcal{M}_1$" involve notions of "embedding." It is technically convenient to derive both kinds of embedding from a common framework. To this end, in this section we define a general notion of semantics for labels, from which arises a generic notion of embedding. Then, we present two concrete semantics for labels—a *value semantics* corresponding to the labels-only interpretation, and an *evaluation semantics* corresponding to the labels-in-programs interpretation—and characterize the two notions of embeddings that they induce.

4.1 Basic definitions

We write $\mathcal{R}_1 \precsim \mathcal{R}_2$ when $\mathcal{R}_1$ is a coarser relation than $\mathcal{R}_2$—that is, when $R_2 \subseteq R_1$.

4.1.1 Definition [Label semantics]: A *label semantics* assigns to each label algebra a (label, authority)-indexed binary relation. More precisely, if $\mathcal{M} = (\mathcal{L}, \mathcal{A}, \ldots)$ is a label algebra, and σ is a label semantics, then ${}^{\mathcal{M}}\sigma^L_A$ (with $L \in \mathcal{L}$ and $A \in \mathcal{A}$) is a binary relation over a set $\mathcal{X}_{\mathcal{M}}$. (We will usually leave the label algebra $\mathcal{M}$ implicit).

For example, given a label algebra $\mathcal{M} = (\mathcal{L}, \mathcal{A}, \ldots)$, $L \in \mathcal{L}$ and $A \in \mathcal{A}$, the *value semantics* mentioned above is given by the binary relation $\approx^L_A$ over $\mathcal{X}_{\mathcal{M}}$, the set of atoms (labeled values) of $\lambda_{\mathcal{M}}$. In this example, maps $m \in \mathcal{M}_1 \to \mathcal{M}_2$ are lifted to maps $\hat{m} \in \mathcal{X}_{\mathcal{M}_1} \to \mathcal{X}_{\mathcal{M}_2}$ with the properties of Lemma 3.1.2. These properties are needed in general for Proposition 4.2.3, that enables transitive reasoning.

4.1.2 Definition [Label semantics (Continued)]: For each $m \in \mathcal{M}_1 \to \mathcal{M}_2$ there is a function $\hat{m} \in \mathcal{X}_{\mathcal{M}_1} \to \mathcal{X}_{\mathcal{M}_2}$ s.t.

- $\widehat{id_{\mathcal{M}}} = id_{\mathcal{X}_{\mathcal{M}}}$.
- For $m_{12} \in \mathcal{M}_1 \to \mathcal{M}_2$ and $m_{23} \in \mathcal{M}_2 \to \mathcal{M}_3$, have $\widehat{m_{23}} \circ \widehat{m_{12}} = \widehat{m_{23} \circ m_{12}}$

The value semantics example satisfies some monotonicity properties as shown in Lemma 3.2.6. In general we require authority and label monotonicity properties of label semantics:

4.1.3 Definition [Label semantics (Continued)]: For each label algebra $\mathcal{M} = (\mathcal{L}, \mathcal{A}, \ldots)$, $L \in \mathcal{L}$ and $A \in \mathcal{A}$,

- If $A_1 \leq A_2$, then $\sigma^L_{A_1} \precsim \sigma^L_{A_2}$.
- If $L_1 \sqsubseteq_A L_2$ then $\sigma^{L_1}_A \precsim \sigma^{L_2}_A$.

Intuitively, these requirements mean that giving more power to the observer—i.e. increasing his authority or his clearance—permits him to perform *finer* observations.

By lemmas 3.1.2 and 3.2.6, the value semantics, written $[\![L]\!]^{\mathrm{v}}_A$, really is a label semantics.

4.2 Label algebra embeddings

We now define some properties that characterize label algebra maps that behave well with respect to some semantics.

4.2.1 Definition [Sound, Complete, Embedding]: Suppose $\mathcal{M}_1$ and $\mathcal{M}_2$ are label algebras, and $m \in \mathcal{M}_1 \to \mathcal{M}_2$.

- m is *sound* with respect to a semantics σ when, for any label algebra $\mathcal{M}$, authority A, label L, x and y,
$$\text{if} \quad x \; {}^{\mathcal{M}}\sigma^L_A \; y \quad \text{then} \quad \hat{m}(x) \; {}^{m(\mathcal{M})}\sigma^{m(L)}_{m(A)} \; \hat{m}(y).$$
- m is *complete* with respect to a semantics σ when for any label algebra $\mathcal{M}$, authority A, label L, x and y,
$$\text{if} \quad \hat{m}(x) \; {}^{m(\mathcal{M})}\sigma^{m(L)}_{m(A)} \; \hat{m}(y) \quad \text{then} \quad x \; {}^{\mathcal{M}}\sigma^L_A \; y.$$
- m is an *embedding* with respect to a semantics σ (written $m \in \mathcal{M}_1 \overset{\sigma}{\hookrightarrow} \mathcal{M}_2$) when it is both sound and complete.
- $\mathcal{M}_1$ *embeds in* $\mathcal{M}_2$ for a semantics σ, (written $\mathcal{M}_1 \overset{\sigma}{\hookrightarrow} \mathcal{M}_2$) if there exists a function m such that $m \in \mathcal{M}_1 \overset{\sigma}{\hookrightarrow} \mathcal{M}_2$.

Intuitively, a sound map decreases the power of an observer to distinguish, whereas a complete map increases her power to distinguish. Embeddings do not change the power of the observer. Embedding is the notion of *good encoding* that we use in the rest of the paper. We'll see that different label semantics lead to different kinds of embeddings.

As a first example we have the identity embedding from the 0-authority projection of a model back into it.

4.2.2 Proposition: For any label algebra $\mathcal{M}$ and any label semantics σ, $\mathcal{M}_0$ σ-embeds into $\mathcal{M}$.

Embeddings allow some modular reasoning, as they form a category.

4.2.3 Proposition: Label algebras and embeddings form a category:

- $id_{\mathcal{M}} \in \mathcal{M} \hookrightarrow \mathcal{M}$, where $id_{\mathcal{M}}$ is the identity function on the label algebra $\mathcal{M}$;
- if $m_{12} \in \mathcal{M}_1 \hookrightarrow \mathcal{M}_2$ and $m_{23} \in \mathcal{M}_2 \hookrightarrow \mathcal{M}_3$, then $m_{23} \circ m_{12} \in \mathcal{M}_1 \hookrightarrow \mathcal{M}_3$.

The category property of embeddings gives rise to the preorder of embeddability, that enables transitive reasoning on embeddability.

4.2.4 Proposition: For any label semantics σ, the relation $\overset{\sigma}{\hookrightarrow}$ on label algebras is a preorder.

4.3 Value semantics of labels

In the rest of the document, we write $\overset{\mathrm{v}}{\hookrightarrow}$ to denote embeddability w.r.t. the value semantics. We can characterize value embeddings, which is useful to prove or disprove the existence of embeddings between label algebras.

4.3.1 Theorem [Characterization of value embeddings]: A label algebra map $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is a value embedding iff for any $L_1, L_2 \in \mathcal{L}_1$ and $A \in \mathcal{A}_1$:

1. $L_1 \equiv L_2$ iff $m(L_1) \equiv m(L_2)$, and

2. $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$.

Note that (1) is not implied by (2), since m might not map authority 0 in $\mathcal{M}_1$ to 0 of $\mathcal{M}_2$.

The default label does not occur in the characterization of value embeddings because it is not used in the definition of the value semantics. Thus, value embeddings enjoy the following property.

4.3.2 Proposition: Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two label algebras differing only in their default labels. Then, $\mathcal{M}_1 \overset{\text{v}}{\hookrightarrow} \mathcal{M}_2$.

Proof: The identity map satisfies Theorem 4.3.1. □

4.4 Evaluation semantics of labels

The second label semantics that we study relates to the evaluation of programs; thus, it is language dependent. We call it the *evaluation semantics* of labels. With this semantics, we interpret labels as observation policies on *programs*, i.e. as the accuracy of an observer that observes the behavior of programs: two programs obey the same *evaluation* policy if they both evaluate to results, which themselves obey the corresponding *value* policy.

4.4.1 Definition [Evaluation semantics of labels]: $[\![L]\!]^{\text{e}}_A$ is the relation on programs:

$t_1 \ [\![L]\!]^{\text{e}}_A \ t_2 =$

$\exists a_1 a_2.\ L^{def}, \bullet \vdash t_1 \Downarrow a_1$ and $L^{def}, \bullet \vdash t_2 \Downarrow a_2$ and $a_1 \approx^L_A a_2$.

This relation is a label semantics. In the rest of the document, we write $\overset{\text{e}}{\hookrightarrow}$ to denote embeddability w.r.t. the evaluation semantics.

The definition of evaluation semantics is the first time the default label L^{def} plays a role: it is the pc label we start evaluation with. As we will see, its use has consequences on the characterization of evaluation embeddings. Naturally, we start evaluation of programs in the empty environment.

The evaluation semantics is a partial equivalence relation on programs. Considering a definition that is closed under evaluation contexts is left as future work. It will be particularly interesting to see how its embeddings differ from the embeddings of the current evaluation semantics.

We now give a characterization for evaluation embeddings.

4.4.2 Theorem [Characterization of evaluation embeddings]: A map $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is an evaluation embedding iff:

- $m(L_1^{def}) \equiv L_2^{def}$
- if $L_1^{def} \sqsubseteq_A L_1$ and $L_1^{def} \sqsubseteq_A L_2$, then $m(L_1 \sqcup L_2) \equiv m(L_1) \sqcup m(L_2)$
- if $L_1^{def} \sqsubseteq_A L_1$, then $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$
- if $L_2^{def} \sqsubseteq_{m(A)} m(L_1) \sqsubseteq_{m(A)} m(L_2)$, then $L_1 \sqsubseteq_A L_2$
- if $L_1^{def} \sqsubseteq_A L_1 \equiv L_2$, then $m(L_1) \equiv m(L_2)$
- if $L_2^{def} \sqsubseteq_{m(A)} m(L_1) \equiv m(L_2)$, then $L_1 \equiv L_2$

Value- and evaluation-embeddings are incomparable notions. In evaluation embeddings, unlike value embeddings, defaults must be mapped to defaults (up to equivalence). Conversely other properties, which recall the characterization of value embeddings, are only required to hold for labels above the defaults.

Many real world examples of label algebras have a bottom label, used as L^{def}. For that case, the characterization simplifies as follows.

4.4.3 Corollary: Assume $\mathcal{M}_1$ and $\mathcal{M}_2$ such that L_1^{def} (resp. L_2^{def}) is a bottom element for $\mathcal{L}_{\mathcal{M}_1}$ (resp. for $\mathcal{L}_{\mathcal{M}_2}$). Then, $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is an evaluation embedding iff:

- $m(L_1^{def}) \equiv L_2^{def}$
- $m(L_1 \sqcup L_2) \equiv m(L_1) \sqcup m(L_2)$
- $L_1 \equiv L_2$ iff $m(L_1) \equiv m(L_2)$
- $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$

In this special case, evaluation embeddings are also value embeddings.

4.4.4 Proposition: Assume $\mathcal{M}_1 \overset{\text{e}}{\hookrightarrow} \mathcal{M}_2$ and that L_1^{def} (resp. L_2^{def}) is a bottom element for $\mathcal{L}_{\mathcal{M}_1}$ (resp. for $\mathcal{L}_{\mathcal{M}_2}$). Then $\mathcal{M}_1 \overset{\text{v}}{\hookrightarrow} \mathcal{M}_2$.

Proof: By Theorems 4.4.2 and 4.3.1. □

One can wonder under which conditions an evaluation embedding can change the default label. It turns out that if the default label changes from $\bot$ to $\top$ (or from $\top$ to $\bot$), then the input of the embedding must be a trivial label algebra. This result is used to show the non-existence of evaluation embeddings.

4.4.5 Corollary: Assume $\mathcal{M}_1 \overset{\text{e}}{\hookrightarrow} \mathcal{M}_2$.

- If $\mathcal{M}_1$ has a bottom element as default and $\mathcal{M}_2$ has a top element as default, then for any label $L \in \mathcal{L}_1$, $L \equiv \bot$.
- Dually, if $\mathcal{M}_1$ has a top element as default, and $\mathcal{M}_2$ has a bottom element as default, then for any label $L \in \mathcal{L}_1$, $L \equiv \top$.

Proof: Let m be an evaluation embedding. We have $m(\bot) = \top$ and for any L_1, L_2 and A, $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$. In particular, for any L, $m(L) \sqsubseteq_{m(0)} m(\bot)$; therefore $L \sqsubseteq_0 \bot$ and $L \equiv \bot$. Dually, If $m(\top) = \bot$, consider $\mathcal{M}_1^{op}$ and $\mathcal{M}_2^{op}$. □

If the default label of a non-degenerate model is $\bot$ or $\top$ then, since dualization changes bottoms to tops (or conversely), there is no evaluation embedding between the model and its dual. However, it is not true that if there is an evaluation embedding from a model to its dual then this model is degenerate. It is indeed easy to verify that, for all label model $\mathcal{M}$, there is an evaluation embedding from $\mathcal{M} \times \mathcal{M}^{op}$ to $(\mathcal{M} \times \mathcal{M}^{op})^{op}$.

5. Simple Examples

In this section, we define a number of label algebras, ranging from very simple finite lattices with no authorities (§5.1) to infinite label algebras with richer authority structures (§5.2 and §5.3), and investigate embeddings among them. Our goals are twofold: (1) to catalog common examples from the literature, and (2) to explore some interesting symmetries among these examples that become easy to see when they are all presented in a common framework. In particular, we can define four familiar label models—the *Taint*, *Endorsement*, *Readers*, *Distrust* models—by varying the lattice order and default label of the same basic structure, where labels are sets of principals. We use the characterization theorems to exhaustively settle the existence or non-existence of embeddings among all of these simple models (Figures 4 and 6).

We present the definitions by giving the underlying set of labels and authorities and the flows-to ordering for all authorities including the empty one. Authorities are either the singleton set **1** or else sets of principals ordered by inclusion. Where unspecified, the default label is the bottom element of $\mathcal{L}$.

5.1 Basic Models

We start with some very simple finite label algebras with trivial authority structures ($\mathcal{A} = \mathbf{1}$); these models are defined in Figure 3. The characterization theorems make it easy to show existence or non-existence of embeddings, summarized in Figure 4.

As a sanity check, note that the trivial model **1** can be embedded in any label model, using the function that maps the only label in **1** to the default label in the target model. As a second sanity check, note that the public / secret model **2** defined in §2 *cannot* be value-embedded in **1**, since Theorem 4.3.1 demands that value embeddings be injective on labels. Moreover, **2** cannot be evaluation-embedded in **1**, by Proposition 4.4.4 (noting that the default element of **1** is

1: Trivial Model

$\mathcal{L} = \mathbf{1}$

3: Three-Point Linear Model

$\mathcal{L} = \{\top, 0, \bot\} \quad \bot \sqsubseteq 0 \sqsubseteq \top$

And similarly **4**, **5**, etc.

4D: Diamond Model

$\mathcal{L} = \{\top, p, q, \bot\} \quad \bot \sqsubseteq p \sqsubseteq \top$ and $\bot \sqsubseteq q \sqsubseteq \top$

$\mathbf{T_0}$: Basic Taint Model

$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \quad L_1 \sqsubseteq L_2 = L_1 \subseteq L_2$

$\mathbf{CR_0}$: Basic Readers Model

$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \quad L_1 \sqsubseteq L_2 = L_1 \supseteq L_2$

Figure 3. Basic models.

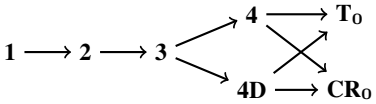

Figure 4. Embeddings among basic models. Solid arrows denote existence of both value and evaluation embeddings. No arrow means absence of both kinds of embeddings.

also a bottom element). Similarly, **2** evaluation- and value-embeds in **3**, but **3** does not embed in **2**. There is no value (or evaluation) embedding from **4** to **4D** or vice versa, because there is no injective map between the underlying label sets that preserves the $\sqsubseteq$ relation (**4** is totally ordered, while **4D** is not).

The labels of the basic taint model $\mathbf{T_0}$ are sets of principals, representing the principals who may have tainted (influenced) some piece of labeled data. Everything is untainted by default. Alternatively, $\mathbf{T_0}$ can be viewed as a basic *secrecy* model, where a label represents a set of principals whose secret information has been used while computing some value. It might also be called a *conjunctive model*, since the authority of *all* the principals in a label is required to declassify a labeled value to be completely public. The basic readers model $\mathbf{CR_0}$ is similar to the readers model (**CR**) from §2.2, except that authorities are trivial. Labels represent sets of principals who are permitted to read some data. This could also be called a *disjunctive model*, since the authority of just *one* of the principals in a label is sufficient to declassify a labeled value to $\bot$. (The terminology in this area is not very consistent: both the taint and the reader model have been referred to as "reader" models in the literature, the former more commonly in the OS community and the latter in the language community.)

5.2 Universal models

We next consider four *universal* models: the Universal Endorsement, Taint, Readers, and Distrust models. These are all set-based models in the style of $\mathbf{T_0}$ and $\mathbf{CR_0}$, with sets of principals as both labels and authorities. They are "universal" in the sense that these sets can be arbitrary subsets of the whole set of principals. All four can be specified by varying two parameters: the default label and the direction of the flows-to relation (Figure 5).

The universal endorsement model **UE** generalizes the endorsement model **CE** from §2.2: it has more labels and authorities, but behaves the same otherwise.

UE: Universal Endorsement Model

$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2 \qquad L^{def} = \emptyset$

UT: Universal Taint Model

$L_1 \sqsubseteq_A L_2 = L_1 \subseteq L_2 \cup A \qquad L^{def} = \emptyset$

UR: Universal Readers Model

$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2 \qquad L^{def} = \mathbb{P}$

UD: Universal Distrust Model

$L_1 \sqsubseteq_A L_2 = L_1 \subseteq L_2 \cup A \qquad L^{def} = \mathbb{P}$

Figure 5. Universal models. In this figure, $\mathcal{L} = \mathcal{A} = \mathcal{P}(\mathbb{P})$. Note that **UE** and **UR** have the same flows-to relation. The same is true of **UT** and **UD**.

The universal taint model **UT** generalizes the basic taint model $\mathbf{T_0}$. A label represents a set of principals who have tainted some piece of data, and data is untainted by default. Authorities are used to remove taint. That model is very similar to the universal endorsement model **UE**: the only difference is that more endorsement decreases the sensitivity (restrictiveness) of a label whereas more taint increases it. **UT** and **UE** are duals in the sense of Definition 2.3.1.

The universal readers model **UR** extends **CR** with larger labels—arbitrary sets of principals. This model and **UT** are evaluation equivalent: using set complementation as a map, they can be evaluation-embedded one in each other.

The universal distrust model **UD** is the same as **UT**, but with a top default label. Labels in this model can be interpreted as sets of principals who *dis*trust some piece of data (this terminology was proposed by [6]). The default label is $\mathbb{P}$, meaning that, by default, everyone distrusts some given piece of data. Authority can be used to remove a principal, that is to make a principal trust the data. This model is the dual of **UR**. It is also evaluation-equivalent to **UE** (using complementation as a map): principals who distrust a piece of data are the ones who did *not* endorse it.

5.3 Syntactic models

The universal models are pleasantly simple, but they are not suitable for real languages or systems because they are not *syntactic*—in general, their labels have no finite representation. However, we can define two operators that "cut down" universal models to syntactic ones.

5.3.1 Definition: Let **L** be a label model whose underlying set of labels and authorities is $\mathcal{P}(\mathbb{P})$. We write $Fin(\mathcal{M})$ for the label algebra that restricts the sets of labels and authorities of $\mathcal{M}$ to $\mathcal{P}^{fin}(\mathbb{P})$, and we write $Fin^+(\mathcal{M})$ for the label algebra that restricts the sets of labels and authorities of $\mathcal{M}$ to $\mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\}$.

The reason for having two operators is that Fin removes $\mathbb{P}$, which is sometimes needed as the default label.

We write **CT**, **CE**, **CR** and **CD** for the images under Fin^+ of the universal models **UT**, **UE**, **UR** and **UD**. (The **C** stands for "Completed": these models contain all finite sets of principals plus the limit element $\mathbb{P}$.) We write **T** and **E** for the images under Fin of **UT** and **UE**. We cannot apply Fin to **UR** and **UD**, since the default label $\mathbb{P}$ would not make sense. **CR** and **CE** were already defined in §2; this is just an alternate definition.

The main consequence of removing (almost all of the) non-finite labels is that the taint model does not embed into the reader model anymore (as we shall see below). Another consequence is that the completed distrust model **CD** is degenerate, since the labels start as

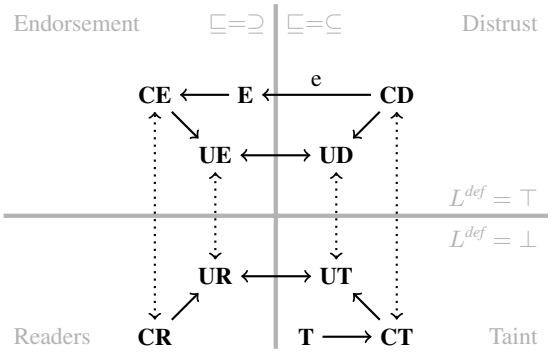

Figure 6. Embeddings among universal and syntactic models. Dotted arrows mean only value embeddings.

high by default and either they stay high—when the authority is not $\mathbb{P}$—or else the whole lattice collapses into the trivial one—when the $\mathbb{P}$ authority is used. This is formalized by the following proposition.

2A: Public / Secret Model With Authorities

$\mathcal{L} = \{\bot, \top\}$ $\quad \mathcal{A} = \{\bot, \top\}$
$\bot \sqsubseteq_\bot \top$ $\quad \top \sqsubseteq_\top \bot$ $\quad \bot \sqsubseteq_\top \top$ $\quad L^{def} = \top$

5.3.2 Proposition: $\mathbf{CD} \overset{e}{\hookrightarrow} \mathbf{2A}$.

5.4 Embeddings among universal and syntactic models

The relative expressiveness of the universal and syntactic models is summarized in Figure 6. Most of the embeddings are defined using the identity map, and it is very easy to prove that they are indeed embeddings. (The evaluation embeddings between universal models, such as the one from **UR** to **UT**, are exceptions to this rule, since they are not the identity but complementation.) Proving non-existence of embeddings is significantly harder.

Note that there can be no evaluation embedding between the upper and lower parts of the figure because of Corollary 4.4.5.

We can prove the non-existence of value embeddings by considering the number of labels that are greater than an infinite number of labels and the number that are smaller than an infinite number of labels. Value embedding increase those numbers and the only arrows that would respect this property are the one in the figure. This also gives us the results of non-existence of evaluation embeddings for the lower part of the label because of Proposition 4.4.4.

For the proofs of non-existence of evaluation embeddings in the upper part of the diagram, we rely on the following technical lemma.

5.4.1 Lemma: There is no value embedding from **E** to a finite model with a high default.

Proof: Suppose, for a contradiction, that m is an evaluation embedding from **E** to some finite model **F** with a high default. Recall Theorem 4.4.2. Using the first bullet we have $m(\emptyset) = \top$. **F** is finite, so there must be some p and q with $m(\{p\}) = m(\{q\})$. But $\{p\} \sqsubseteq_{\{p,q\}} \emptyset$ and $\{q\} \sqsubseteq_{\{p,q\}} \emptyset$ in **E**, so using the second bullet we have $m(\emptyset) = m(\{p\} \sqcup \{q\}) = m(\{p\}) \sqcup m(\{q\}) = m(\{p\})$. Using the sixth bullet we then have $\emptyset = \{p\}$—a contradiction. □

We can simplify the proofs of non-existence of evaluation embeddings in the upper part of the diagram by noting that **CD** evaluation-embeds into every label algebra in the upper part of the diagram and no other label algebra in this part of the diagram evaluation-embeds into **CD**, since **E** embeds into all of them and does not embed into **CD** (Lemma 5.4.1). Also note that **UD** is evaluation-equivalent to **UE**, which leaves us with only **E**, **CE** and **UE**.

To prove that **CE** does not evaluation-embed into **E**, we consider the (finite) image of authority $\mathbb{P}$. Using this, we prove that the image of **CE** is finite, which contradicts Lemma 5.4.1. To prove that **UE** does not evaluation-embed into **CE**, we consider a putative embedding m. It cannot be injective because of cardinality, so we consider two labels L_1 and L_2 such that $L_1 \not\sqsubseteq L_2$ and $m(L_1) = m(L_2)$. Using the third bullet of Theorem 4.4.2 we have that $m(\emptyset) \sqsubseteq_{m(L_1)} m(L_1)$ and $m(\emptyset) \not\sqsubseteq_{m(L_1)} m(L_2)$, a contradiction.

5.5 Apropos integrity

The observation that integrity can be treated as a formal dual to confidentiality goes back at least to Biba [2]. This agrees with the fact that **UE** is the formal dual of **UT** and and **UR** the formal dual of **UD**. Many real-world systems (e.g., [5, 14, 15]) have relied on this observation to provide unified mechanisms that serve both goals.

The terms *secrecy* (or *confidentiality*) *model* and *integrity model* are used somewhat informally (and not entirely consistently) in the literature. In our framework, we can offer a precise distinction: models with a $\top$ default are integrity models, while models with a $\bot$ default are secrecy models. That is consistent with the dualization operator, which turns models with low defaults into models with high defaults and vice versa. Our **R**- and **T**-models fall in the secrecy category and our **E**- and **D**-models in the integrity category. This seems consistent with the fact that **CR** is often called a *reader model* (which seems more related to secrecy than to integrity) in the language-based security community and that **T** is called a reader model in the OS community. Models similar to **E** are generally called integrity models. We are not aware of any system making use of **D**-models, though they are discussed in [6].

6. Real-World Examples

We now turn our attention to formalizing the label models of several existing systems. While some do not quite fit the formal structure of label algebras, even imperfect descriptions in a common framework will hopefully clarify their similarities and differences. Moreover, we can use these formalizations to study the existence and non-existence of embeddings, as we did in the previous section. We do not settle the question for all pairs of examples, but we do establish several results involving variants of the DLM and DC models.

6.1 Disjunction-Category (DC) labels

Disjunction Category labels [13] come from a Haskell security library called LIO [14], part of the HAILS system. **DC** labels are made of a secrecy part and an integrity part. Let us first focus on the secrecy part ($\mathbf{DC_S}$); the full model will be defined later on.

Labels of $\mathbf{DC_S}$ are finite boolean formulae in conjunctive normal form, containing no negation—i.e. finite conjunctions of finite disjunctions of principals. We write $\mathcal{F}$ to denote the set of these formulae. The flows-to relation is the reverse logical entailment, written $\Leftarrow$. Intuitively, these formulae tell who can observe some data. The relation $\sqsubseteq$ allows to make conservative approximations about who is allowed to observe the data. *True* is the empty conjunction $\emptyset$ and is the $\bot$ element—it means that any principal can observe the data; *False* is the empty disjunction $\{\emptyset\}$ and is the $\top$ element—it means nobody is allowed to observe the data.

For example, the $\mathbf{DC_S}$ label $L = p_1 \wedge (p_2 \vee p_3)$ can be read as "the data can be read by somebody that has p_1's credentials and either p_2's or p_3's." It flows to $p_1 \wedge p_2$, because somebody that has p_1's and p_2's credentials respects the policy of L.

Authorities are also formulae: $L_1 \sqsubseteq_A L_2$ means that $L_1 \Leftarrow (L_2 \wedge A)$. The label L can flow to $p_2 \vee p_3$ using authority p_1, because somebody that has either p_2's or p_3's credentials and the ability to locally use p_1's credentials respects the policy of L.

$\mathbf{DC_S}$: DC Labels (secrecy part)

$\mathcal{L} = \mathcal{F} \qquad L^{def} = True \qquad \mathcal{A} = \mathcal{F} \qquad 0 = True$
$A_1 \leq A_2 = A_1 \Leftarrow A_2 \qquad A_1 \vee A_2 = A_1 \wedge A_2$
$S_1 \sqsubseteq_A S_2 = S_1 \Leftarrow (S_2 \wedge A)$
$L_1 \sqcup L_2 = L_1 \wedge L_2 \qquad L_1 \sqcap L_2 = L_1 \vee L_2$

Each formula is in normal form, so the definitions of join and meet are up to renormalization. In §6.3, we consider the secrecy part of **DC** as our subject of study for embeddability.

The definition of the full **DC** model follows: it only adds an integrity component to $\mathbf{DC_S}$.

DC: Full DC Labels (LIO)

$\mathcal{L} = \mathcal{F} \times \mathcal{F} \qquad L^{def} = (True, True)$
$\mathcal{A} = \mathcal{F} \qquad 0 = True$
$A_1 \leq A_2 = A_1 \Leftarrow A_2 \qquad A_1 \vee A_2 = A_1 \wedge A_2$
$(S_1, I_1) \sqsubseteq_A (S_2, I_2) =$
$\quad S_1 \Leftarrow (S_2 \wedge A) \quad$ and $\quad I_2 \Leftarrow (I_1 \wedge A)$
$(S_1, I_1) \sqcup (S_2, I_2) = (S_1 \wedge S_2, I_1 \vee I_2)$
$(S_1, I_1) \sqcap (S_2, I_2) = (S_1 \vee S_2, I_1 \wedge I_2)$

Note that the default label is neither a top nor a bottom element: it is the pair of the bottom for secrecy and the top for integrity, i.e. the "most public" and the "least endorsed". Authorities in **DC** are the same formulae as in $\mathbf{DC_S}$. Note that **DC** is isomorphic to $\mathbf{DC_S} \times \mathbf{DC_S}^{op}$.

Note that some of the basic label models can also be given natural logical readings, as some of the following embeddings will illustrate. There leads one to wonder about label models where the labels and authorities are arbitrary formulae in some logic (perhaps with implication, quantification, ...). We leave this for future investigation.

6.2 Simplified DLM (with no principal hierarchy)

We describe a stripped down version of the Decentralized Label Model [9]: for the sake of simplicity, we focus on its secrecy part only, and don't yet model its principal hierarchy (*acts-for* relation). Discussing the extension with *acts-for* is the purpose of §6.4. We name the current version $\mathbf{DLM_S}$, and the extended one $\mathbf{DLM_S}$**-H**.

Labels in $\mathbf{DLM_S}$ are sets of policies, where policies are drawn from the set $Pol = \{p \to P \mid p \in \mathbb{P}, P \in \mathcal{P}^{fin}(\mathbb{P})\}$. The sets on the right hand side are called reader sets; they are akin to labels of **CR** in that they decrease as we go up in the lattice of labels. For instance, the label $L_1 = \{p \to \{p_1, p_2\}\}$ says that the principal p allows only principals p_1 and p_2 to read some data. Label $L_2 = \{p \to \{p_1\}\}$ is strictly more secure than L_1—i.e. $L_1 \sqsubseteq L_2$—since L_2 allows less possible readers.

When $(p \to P) \in L$, we say that principal p *owns* the policy $p \to P$ in L. Principals can own multiple policies. That label model is called *decentralized*, because several principals can independently own different policies, that is to say, ask for different security requirements. In label $L_3 = \{p \to \{p_1, p_2\}, q \to \{p_1\}\}$, two principals—$p$ and q—express a policy. The resulting policy is, according to [9], the *intersection* of p's and q's policies: all the stated policies must be enforced. Note that $L_1 \sqsubseteq L_3$ and that $\{q \to \{p_1\}\} \sqsubseteq L_3$.

Authorities are sets of *owners*. They specify which policies can be modified: for any p in the authority, one can arbitrarily change or remove policies that are owned by p in a label but the other policies can only be changed to more restrictive ones. For instance, $L_3 \sqsubseteq_{\{p\}} \{q \to \{p_1\}\}$ and $L_3 \sqsubseteq_{\{p\}} \{p \to \{p_1, p_2, p_3\}, q \to \{p_1\}\}$. However, $L_3 \not\sqsubseteq_{\{p\}} \{q \to \{p_1, p_2\}\}$, because in the latter label, the security policy owned by q is more permissive than the one in L_3.

The complete definition of the $\mathbf{DLM_S}$ label algebra follows.

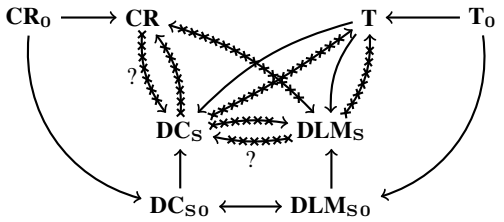

Figure 7. Embeddings with $\mathbf{DLM_S}$ and $\mathbf{DC_S}$. Plain arrows are value embeddings; crossed arrows denote non-existence of value embeddings; '?' means conjecture.

$\mathbf{DLM_S}$: Simplified DLM (secrecy, no acts-for)

$\mathcal{L} = \mathcal{P}^{fin}(Pol) \qquad L^{def} = \emptyset \qquad \mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \qquad 0 = \emptyset$

$L_1 \sqcup L_2 = L_1 \cup L_2$
$L_1 \sqcap L_2 = \{p \to P_1 \cup P_2 \mid (p \to P_1) \in L_1, (p \to P_2) \in L_2\}$

$L_1 \sqsubseteq L_2 = \forall (p \to P_1) \in L_1.\ \exists (p \to P_2) \in L_2.\ P_2 \subseteq P_1$
$L_1 \sqsubseteq_A L_2 = L_1 \sqsubseteq L_2 \sqcup L_A$ where $L_A = \{p \to \emptyset \mid p \in A\}$

Note that a given principal can own more than one policy. For instance, $L_1 \sqsubseteq \{p \to \{p_1\}, p \to \{p_2\}\}$, and the converse does not hold. Interestingly, the explanation based on reader sets does not scale to the case of principals owning several policies. For instance, one could expect that $\{p \to \{p_1\}, p \to \{p_2\}\}$ and $\{p \to \emptyset\}$ express the same requirement, i.e. "p says that nobody can read the data". However, the former label is *strictly* lesser than the latter. Here is how we understand such labels: the sets of the right hand side express disjunctions of principals, whereas the juxtaposition of two policies means their conjunction. We conjecture that one can represent $\mathbf{DLM_S}$ labels as finite maps from principals to $\mathcal{F}$ (conjunctions of disjunctions of principals).

Li *et al.* [6] claim some encodability results, namely that the two point model, the writer model, the endorsement model and the distrust model can all be encoded in the Decentralized Label Model. Unfortunately, while they describe the lattice structures of all the models they consider, they do not explain what are their default labels, and they ignore authority. We will see in §6.3 that the presence of authorities plays an interesting role.

6.3 Some embeddability results

We have gathered in Figure 7 some relative embeddability results about $\mathbf{DC_S}$, $\mathbf{DLM_S}$ and some of the models of §5.3. Note that it is preliminary work, hence we have not achieved an exhaustive exploration of the area. For instance, we are definitely interested in considering the full models, that is to say with secrecy *and* integrity and, in the case of the **DLM**, the extension with an *acts-for* relation presented in §6.4.

The first thing to notice about that diagram is that neither **CR** nor **T** are expressive enough to express $\mathbf{DC_S}$ of $\mathbf{DLM_S}$ labels. That is not a surprise.

Then, the most noticeable result is that the presence of authorities sometimes *precludes* embeddability. For instance, the models $\mathbf{DC_{S0}}$ and $\mathbf{DLM_{S0}}$, that don't have authorities, embed in each other, but it is not true for their authority-enriched versions (we proved one way, conjectured the other): intuitively, there is indeed no notion of *owner* of a policy in $\mathbf{DC_S}$, and conversely there is no disjunction of authorities in $\mathbf{DLM_S}$.

Another instance of that phenomenon is the embeddability of **CR**: it does not embed in $\mathbf{DC_S}$ or $\mathbf{DLM_S}$, but its 0-authority version does. By contrast, the behavior of **T** is not influenced by the presence of authorities.

For the sake of concreteness, we detail one embeddability and one non-embeddability arrows of Figure 7

6.3.1 Proposition: $\mathbf{T} \overset{e}{\hookrightarrow} \mathbf{DLM_S}$.

Proof: Define $m(S) = \bigcup_{p \in S} \{p \to \emptyset\}$ and $m(A) = A$. The map m verifies the conditions of Corollary 4.4.3. □

Thanks to Proposition 4.4.4, that evaluation embedding is also a value embedding.

6.3.2 Proposition: **CR** does not value-embed in $\mathbf{DLM_S}$.

Proof: Assume that such a value embedding exists; let's call it m.

Let us first prove (1): there exists A_0 such that for any label $L \in \mathbf{DC_S}$, $\mathrm{dom}\, m(L) \subseteq A_0$. For any label $L \in \mathbf{CR}$, $L \sqsubseteq_0 \emptyset$, therefore $m(L) \sqsubseteq_{m(0)} m(\emptyset)$ by Theorem 4.3.1. Then, $\mathrm{dom}\, m(L) \subseteq m(0) \cup \mathrm{dom}\, m(\emptyset)$ by definition.

Then, let us show (2): for any $A \in \mathcal{A}_{\mathbf{CR}}$ and $L_1, L_2 \in \mathcal{L}_{\mathbf{CR}}$, $m(L_1) \sqsubseteq_{m(A) \cap A_0} m(L_2)$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$. The direct way holds by properties of label algebras, since $m(A) \cap A_0 \leq m(A)$. Let us show the converse: assume $m(L_1) \sqsubseteq_{m(A)} m(L_2)$. Assume $p \to P_1 \in m(L_1)$. If $p \in m(A) \cap A_0$, then $(p \to \emptyset) \in m(L_2) \sqcup L_{m(A) \cap A_0}$, which concludes the proof. Assume now, that $p \notin m(A) \cap A_0$. We know that $p \in A_0$ by (1), thus $p \notin m(A)$. Therefore, there exists P_2 such that $p \to P_2 \in m(L_2)$ and $P_2 \subseteq P_1$, by definition of $\sqsubseteq$ in $\mathbf{DLM_S}$. Then, $p \to P_2 \in m(L_2) \sqcup L_{m(A) \cap A_0}$, which concludes the proof that $m(L_1) \sqsubseteq_{m(A) \cap A_0} m(L_2)$.

Let us consider the function $\lambda A.\ A \cap A_0$. It has an infinite domain and a finite range, therefore it cannot be injective. Thus, there exists A_1 and A_2 such that $A_1 \neq A_2$ and $m(A_1) \cap A_0 = m(A_2) \cap A_0$ (3). Therefore, for any L_1 and L_2,

	$L_1 \sqsubseteq_{A_1} L_2$	
iff	$m(L_1) \sqsubseteq_{m(A_1)} m(L_2)$	by Theorem 4.3.1
iff	$m(L_1) \sqsubseteq_{m(A_1) \cap A_0} m(L_2)$	by (2)
iff	$m(L_1) \sqsubseteq_{m(A_2) \cap A_0} m(L_2)$	by (3)
iff	$m(L_1) \sqsubseteq_{m(A_2)} m(L_2)$	by (2)
iff	$L_1 \sqsubseteq_{A_2} L_2$	by Theorem 4.3.1.

Then, since $\emptyset \sqsubseteq_{A_2} A_2$, we have $\emptyset \sqsubseteq_{A_1} A_2$, i.e. $A_2 \subseteq A_1$. Similarly, since $\emptyset \sqsubseteq_{A_1} A_1$, we have $\emptyset \sqsubseteq_{A_2} A_1$, i.e. $A_1 \subseteq A_2$. We proved $A_1 = A_2$—a contradiction. □

Using Proposition 4.4.4, there is no evaluation embedding either.

We hope to have convinced the reader that including authorities in the reasoning about encodability makes a lot of sense, as it can lead to surprising results. Thus, informal claims about the expressive power of label models really need to be taken with a grain of salt.

6.4 DLM with principal hierarchy

In this section, we extend the $\mathbf{DLM_S}$ label algebra of §6.2 with principal hierarchies: we end up with the secrecy component of Jif [9]. A principal hierarchy is a partial order on principals. The idea is to consider that the principal hierarchy is one component of the authorities: its role is to enlarge the flows-to relation. In Jif, the principal hierarchy is a system-wide centralized piece of state. As noticed by [8], the meet operation is not in general the *greatest* lower bound for any principal hierarchy: therefore, the description that follows is *not* a label algebra. One could think that might be a good reason to drop meets from the definition of label algebras, but Jif programs do make use of meets in the type of methods, and during typechecking. Moreover, dropping meets would prevent us from defining the dual of a label algebra.

In Jif, the set of principals $\mathbb{P}$ has two distinguished points $p_\bot$ and $p_\top$. Let $\mathcal{H}$ be the set of partial orders on principals such that $p_\bot$ is a bottom element and $p_\top$ is a top element: we call *principal hierarchies* the elements of $\mathcal{H}$. For $H \in \mathcal{H}$, $(p_1, p_2) \in H$ reads "p_2 *acts for* p_1". We define authorities as pairs of a principal hierarchy and a set of principals: that latter set represents the owners of policies that can be declassified, as in the $\mathbf{DLM_S}$ label algebra.

Formally, we also add a top element to authorities: hierarchies are partial orders, therefore they are *acyclic*. In Jif, extending the principal hierarchy is a partial operation, since it must preserve acyclicity. In label algebra, however, the join on authorities is a total function: the role of the top authority is precisely to model the possible *failure* of extending authorities: if the system reaches the top authority, it warns the user of its illegal action.

Figure 8 contains the definition of $\mathbf{DLM_S}$**-H**. On the one hand, the fact that the definition does not fit the label algebra interface can clearly be understood as a limitation of label algebras. On the other hand, one could also consider that it is an imperfection of $\mathbf{DLM_S}$: meet does not gives the best lower bound, and therefore the type analysis performed by Jif can lose some accuracy.

$\mathbf{DLM_S}$-H: not a label algebra

$\mathcal{L} = \mathcal{P}^{fin}(Pol)$
$\mathcal{A} = \{\top\} + (\mathcal{H} \times \mathcal{P}^{fin}(\mathbb{P}))$

$0 = (\{(p_\bot, p_\top)\}, \emptyset)$
$(H, P) \leq \top = \text{true}$
$(H_1, P_1) \leq (H_2, P_2) = H_1 \subseteq H_2 \text{ and } P_1 \subseteq P_2$
$(H_1, P_1) \vee (H_2, P_2) = ((H_1 \cup H_2)^+, P_1 \cup P_2)$
if $(H_1 \cup H_2)^+$ is a partial order
$A_1 \vee A_2 = \top$ otherwise
$L_1 \sqsubseteq_\top L_2 = \text{true}$
$L_1 \sqsubseteq_{(H,\emptyset)} L_2 = \forall (p_1 \to P_1) \in L_1,\ \exists (p_2 \to P_2) \in L_2,$
$(p_1, p_2) \in H$ and
$\forall p_2' \in P_2, \exists p_1' \in P_1, (p_1', p_2') \in H$
$L_1 \sqsubseteq_{(H,A)} L_2 =$
$L_1 \sqsubseteq_{(H,\emptyset)} L_2 \sqcup L_A$ where $L_A = \{p \to \emptyset \mid p \in A\}$

Figure 8. $\mathbf{DLM_S}$**-H**: $\mathbf{DLM_S}$ with principal hierarchy.

6.5 Asbestos

Asbestos [4] is a information-flow secure operating system. Labeling in Asbestos is coarse grained, in the sense that label are attached to kernel objects, such as files or processes. Its labels are *almost-constant* maps from principals to security levels. *Level* is the set $\{\star, 0, 1, 2, 3\}$ equipped with the total order $\star < 0 < 1 < 2 < 3$. Labels are composed of a finite map from principals to levels, and a default level for the principals that are not mentioned in the map. The authors use the word *category* while we use *principal*. For a label $L \in (\mathbb{P} \overset{\text{fin}}{\to} Level) \times Level$, we define

$$L(p) = \begin{cases} f(p) & \text{if } p \in \mathrm{dom}\, f \\ l & \text{otherwise.} \end{cases}$$

where $L = (f, l)$. The ordering on labels is the pointwise extension of the level ordering.

Each Asbestos process owns a set of privileges, i.e. a set of principals: if p is in that set, a process is allowed to freely change the level that is owned by p in a label L, à la **DLM**. The definition of Asbestos labels follows.

Asbestos:

$\mathcal{L} = (\mathbb{P} \overset{\text{fin}}{\to} Level) \times Level \qquad L^{def} = (\{\}, 1)$
$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \qquad 0 = \emptyset$
$L_1 \sqsubseteq_A L_2 = \forall p, p \in A \text{ or } L_1(p) \leq L_2(p)$

A particularity of Asbestos is that processes can automatically get tainted by the communications they perform, following complex

schemes that involve the use of joins and meets. That implicit taint mechanism was shown to be insecure [15]: "Asbestos changes a processs label to track information flow when it receives IPCs, which is detectable by third parties and can leak information." Later information-flow secure OSes enforce explicit changes of the processes labels.

6.6 Early HiStar

It seems that at least two versions of HiStar exist and use different labels. The earliest version [15] uses labels that are inspired by Asbestos labels. We describe the second version in the next section. There are several differences in the set of primitives that Asbestos and HiStar provide. Among the differences, thread label changes are required to be explicitly stated. In the rest of the section, we only focus on the differences with respect to labels.

The main difference compared to Asbestos is the way untainting (or reclassification) is handled. Indeed in HiStar, level $\star$ is a privileged level, that can only appear in thread labels: it confers a thread the right to untaint a category. It is low in the lattice so that threads need authority to gain untainting privileges: $\star$ is indeed strictly below the default level.

When a thread with label L_T attempts to observe an object with label L_O, the check $L_O \sqsubseteq L_T$ is usually performed. However, that does not correspond to $\star$ being an untainting privilege, since its low position in the lattice prevents flows instead of allowing more flows. The authors explain that the meaning of $\star$ is either bottom or top, depending on the situation. To better explain the mechanism in place, they introduce a special level ✪, called *high star*, that behaves like a maximum level, but is not really part of the lattice of levels: "level ✪ is only used in access rules and never appears in labels of actual objects". Now, the read check becomes $L_O \sqsubseteq L_T^{✪}$, where $L_T^{✪}$ is the label L_T in which every occurrence of $\star$ is replaced with the special top element ✪.

It seems that the fact that $\star$ occurs in a thread label is a way to express the privileges that are owned by that thread. Indeed, we can define $Auth(L) = \{p \mid L(p) = \star\}$: it is the authority of a thread that has L as a thread label. Then, under the assumption that $\star$ does not occur in L_1, $L_1 \sqsubseteq L_2^{✪}$ is equivalent to $L_1 \sqsubseteq_{Auth(L_2)} L_2$, where the indexed flows-to relation is the one of Asbestos. Note that our assumption about L_1 makes sense, since L_1 is supposed not to be a thread label.

6.7 Later HiStar

The more recent version of HiStar [16] seems to have switched to a different kind of labels, that are closer to the models we described in §5. Their labels are indeed pairs of sets of categories: secrecy and integrity categories. As in previous works from the same authors, the word category denotes tag, or principal. Secrecy categories are disjoint from integrity categories. Secrecy categories are also called *read* categories, whereas integrity categories are called *write* categories. Their model corresponds exactly to $\mathbf{T} \times \mathbf{E}$, where $\mathbf{T}$ is the secrecy part (reader) and $\mathbf{E}$ is the integrity part (writer).

6.8 Flume

Flume [5] is another experiment in the field of information-flow secure OSes. The labels in flume are pairs of sets of tags (i.e. what we call principal); one component is used for secrecy, the other one for integrity. Similarly to latest HiStar (§6.7), it is the product of the taint model with the endorsement model.

However, Flume has a notion of authority that makes the description above inaccurate: authorities are sets of "capabilities", that are tags equipped with a polarity annotation—positive or negative. A positive capability permits to add a tag to a label (in whichever component), whereas a negative capability allows removing a tag. That behavior of authorities is captured in Figure 9.

Flume: not a label algebra

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \times \mathcal{P}^{fin}(\mathbb{P}) \qquad L^{def} = (\emptyset, \emptyset)$$
$$(S_1, I_1) \sqsubseteq (S_2, I_2) = S_1 \subseteq S_2 \text{ and } I_1 \supseteq I_2$$

$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \times \mathcal{P}^{fin}(\mathbb{P}) \qquad 0 = (\emptyset, \emptyset)$$
$$(C_1^+, C_1^-) \leq (C_2^+, C_2^-) = C_1^+ \subseteq C_2^+ \text{ and } C_1^- \subseteq C_2^-$$

$$(S_1, I_1) \sqsubseteq_{(C^+, C^-)} (S_2, I_2) =$$
$$S_2 \setminus S_1 \subseteq C^+ \text{ and } S_1 \setminus S_2 \subseteq C^-$$
$$\text{and } I_2 \setminus I_1 \subseteq C^+ \text{ and } I_1 \setminus I_2 \subseteq C^-$$

Figure 9. Flume labels and authorities.

The definition does not form a label algebra, because $\sqsubseteq_0$ is the equality relation on labels, which is different from the relation $\sqsubseteq$ of the underlying lattice. In other words, the way information is allowed to flow is strictly more permissive than the way processes can change labels when using the least authority possible. That very peculiarity makes Flume more flexible, i.e. the way thread labels can change is completely programmable: programmers have a lot of freedom in the way they can shape the lattice. A system like LIO [14] uses *clearance* as a mechanism to control how programs can change their thread label. The clearance mechanism is just a flows-to check, and therefore does not require more than what a label algebra can offer.

Inter-process communication in Flume is facilitated in the following way: "if two processes could communicate by changing their labels, sending a message using the centralized rules, and then restoring their original labels, then the model can safely allow the processes to communicate without label changes". That extra flexibility is captured by the notion of *safe message* between processes. A process P with label $L_P = (S_P, I_P)$ and authority $A_P = (C_P^+, C_P^-)$ can safely send a message to a process Q with label $L_Q = (S_Q, I_Q)$ and authority $A_Q = (C_Q^+, C_Q^-)$ when $S_P \setminus D_P \subseteq S_Q \cup D_Q$ and $I_P \cup D_P \supseteq I_Q \setminus D_Q$, where $D_P = C_P^+ \cap C_P^-$ and $D_Q = C_Q^+ \cap C_Q^-$. That is to say, processes can implicitly use capabilities that they positively *and* negatively own. That is consistent with the intention that a process would need to raise its label (using positive capabilities), perform the communication, and then lower its label back to its original value (using negative capabilities), or the converse—lower, then raise. With our notation, one could say that P can safely send a message to Q if there exists two labels L'_P and L'_Q such that $L_P \equiv_{A_P} L'_P \sqsubseteq L'_Q \equiv_{A_Q} L_Q$.

6.9 Laminar

Laminar [10] uses essentially the same labels and authorities as Flume, therefore it also does not quite fit in the label algebra framework.

7. Related Work

Sabelfeld and Sands [11] give semantic types to secure sequential programs: their types are partial equivalence relations (PERs). They develop an elegant theory that can model different kinds of security policies, including probabilistic ones. They do not consider zauthorities or exceptional flows. It is worth noticing that our value semantics is a family of equivalence relations, and our evaluation semantics is a family of PERs (§4).

In a previous paper the same authors [12] describe some aspects of declassification and what rules it should follow. One of them is *conservativity*: "Security for programs with no declassification is equivalent to noninterference". This rule corresponds to one of the axioms of label algebras, namely: $\sqsubseteq = \sqsubseteq_0$, i.e. the empty authority plays no role. Notice that according to Theorem 3.2.4, programs

with 0 authority are non-interferent in the usual sense. Another rule—*monotonicity of release*—is also directly connected to our work: "Adding further declassifications to a secure program cannot render it insecure". This corresponds to the monotonicity properties that we require on label semantics.

Mantel and Sands [7] study *intransitive non-interference*, a security property of programs that perform declassification. They use PERs that rely on bisimulations of labeled transition systems: by definition, a *strongly secure* program is related to itself. This seems akin to our evaluation semantics, although they use a more intensional point of view. Like our λ-calculus (§3), their language identifies which parts of the code perform declassification. Their small-step semantics is labeled with the authority that each step uses. (Keeping such a trace and using it in the evaluation semantics of labels is an interesting direction for future work for us.) Our treatment of authority differs from theirs: they only consider two authorities—no authority ($\bot$) and declassification authority ($\top$)—and they consider only $\sqsubseteq_\bot$, not $\sqsubseteq_\top$. Instead, the declassification relation ($\rightsquigarrow$), which allows exceptional flows, is not required to be transitive, and neither is $\sqsubseteq_\bot \cup \rightsquigarrow$ in general. It seems that, in our notation, $\sqsubseteq_\top$ should be the relation $(\sqsubseteq \cup \rightsquigarrow)^+$.

8. Future Work

This exploration of label algebras and embeddings is just a beginning. We hope to produce an exhaustive map of the known label algebras along with the existence or absence of embeddings between all pairs. Such a map could be very useful to people that approach the field for the first time, and as a guide for the design of new label algebras.

Although many examples fit the interface of label algebras, §6.4 shows that principal hierarchies require looser conditions on meets and joins: relaxing the definition of label algebras and its implications deserves further study.

The theory of label semantics lacks the ability to generate principals and authorities, a pervasive ingredient in dynamic information flow systems. We are particularly interested in the consequences on embeddings and their existence.

Since it depends on the semantics of a programming language, the evaluation semantics from §4.4 is sensitive to the set of features of the language. We are particularly interested in extending our language with first-class labels and authorities, and possibly imperative features, and seeing the implications on the characterization Theorem 4.4.2. We guess that extending the language in a way that makes use of more features of label algebras leads to embeddings that need to preserve and reflect more aspects of label algebras. It would be of primary interest to see whether we can come up with a language whose evaluation embeddings are exactly the injective label-algebra morphisms (i.e., the injective maps that preserve *all* the label-algebra structure); we conjecture that adding first-class labels may be enough for this.

We would like also to experiment with different instances of the abstract label semantics: for example, our evaluation semantics could use a more semantic notion of program equivalence. We could also consider an information-flow type system for our language, i.e. parametrized over label algebras, and use the typability judgment to define a label semantics.

Acknowledgments

We are grateful to Steve Zdancewic, David Mazières, Deian Stefan, and the members of the SAFE team for fruitful discussions. Robin Morisset helped initiate the design of label algebras. Greg Morrisett participated in early discussions and gave us essential feedback on the technical definitions. This material is based upon work supported by the DARPA CRASH program through the US Air Force Research Laboratory (AFRL) under Contract No. FA8650- 10-C-7090. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] Austin, T. H. and Flanagan, C. 2009. Efficient purely-dynamic information flow analysis. *SIGPLAN Notices 44*, 20–31.

[2] Biba, K. J. 1977. Integrity considerations for secure computer systems. Tech. Rep. ESD-TR-76-372, MTR-3154 Rev 1, Mitre. Apr.

[3] Denning, D. E. 1976. A lattice model of secure information flow. *Communications of the ACM 19*, 236–243.

[4] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., and Morris, R. 2005. Labels and event processes in the Asbestos operating system. In *Proceedings of the 20th Symposium on Operating Systems Principles*. SOSP. ACM, 17–30.

[5] Krohn, M. N., Yip, A., Brodsky, M. Z., Cliffer, N., Kaashoek, M. F., Kohler, E., and Morris, R. 2007. Information flow control for standard OS abstractions. In *Proceedings of the 21st Symposium on Operating Systems Principles*. SOSP. ACM, 321–334.

[6] Li, P., Mao, Y., and Zdancewic, S. 2003. Information integrity policies. In *Proceedings of the Workshop on Formal Aspects in Security & Trust (FAST)*.

[7] Mantel, H. and Sands, D. 2004. Controlled declassification based on intransitive noninterference. In *Proc. Asian Symp. on Programming Languages and Systems*. LNCS. Springer-Verlag, 129–145.

[8] Myers, A. C. 1999. Mostly-static decentralized information flow control. Ph.D. thesis, Massachusetts Institute of Technology.

[9] Myers, A. C. and Liskov, B. 2000. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol. 9*, 410–442.

[10] Roy, I., Porter, D. E., Bond, M. D., McKinley, K. S., and Witchel, E. 2009. Laminar: Practical fine-grained decentralized information flow control. In *Proceedings of the Conference on Programming Language Design and Implementation*. PLDI. ACM, 63–74.

[11] Sabelfeld, A. and Sands, D. 2001. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation 14,* 1, 59–91.

[12] Sabelfeld, A. and Sands, D. 2005. Dimensions and principles of declassification. In *Computer Security Foundations 18th Workshop*, IEEE, Ed. 255–269.

[13] Stefan, D., Russo, A., Mazières, D., and Mitchell, J. C. 2011. Disjunction category labels. In *16th Nordic Conference on Secure IT Systems*. NordSec. Springer, 223–239.

[14] Stefan, D., Russo, A., Mitchell, J. C., and Mazières, D. 2011. Flexible dynamic information flow control in Haskell. In *Proceedings of the 4th Symposium on Haskell*. ACM, 95–106.

[15] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. 2006. Making information flow explicit in HiStar. In *Proceedings of the 7th symposium on Operating systems design and implementation*. OSDI. USENIX Association, 263–278.

[16] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. 2011. Making information flow explicit in HiStar. *Communications of the ACM 54,* 11, 93–101.