Summary

This thesis is on the extension of Lagois connections for secure information flow from two actors to an arbitrary amount. A Lagois connection, due to Melton et al., is a poset system (P, f, g, Q) such that

- $p \leq (g \circ f)(p)$ for all $p : P$ (**LC1**),
- $q \leq (f \circ g)(q)$ for all $q : Q$ (**LC2**),
- $(f \circ g \circ f)(p) = f(p)$ for all $p : P$ (**LC3**), and
- $(g \circ f \circ g)(q) = g(q)$ for all $q : Q$ (**LC4**).

For the purpose of studying information flow we limited ourselves to poset systems over finite inhabited lattices as prescribed by Denning. In regards to information flow, a poset system (P, f, g, Q) is interpreted in the following manner, as prescribed by Bhardwaj and Prasad: P is a collection of labels belonging to some organization O_P (analogous for Q). Information is allowed to flow via the lattice structure of P and Q, i.e., information with label $p : P$ can flow to something/someone with label/clearance $p' : P$ if $p \leq p'$ (analogous for Q). Additionally, information is allowed to flow from $p : P$ to $q : Q$ if $f(p) \leq q$ (analogous for g) and via the transitive closure of the relations just described. Then **LC1** and **LC2** ensure that the connection is secure, and **LC3** and **LC4** give the connection some convenient properties.

The current method for extending Lagois connections to an arbitrary number of organizations, due to Bhardwaj and Prasad, is achieved by chaining several connections together. This, in some situations incurs a loss of label precision. Further, the proposition that these chains are secure is stated in terms of the endpoints of the chain and not the chain as a whole. This motivates us to study arbitrary networks of Lagois connections.

We formulate several definitions of security for these networks. In particular, we formulate security in a manner similar to Nielson et al: Like the case with two organizations the network induces a flow relation $\rightrightarrows$ such that information can flow from p to q if $p \rightrightarrows q$ where p and q might reside in separate lattices in the network. We then say that a network is secure whenever given a lattice P in the network and for all pairs of labels within this lattice $p\ p' : P$ if $p \rightrightarrows p'$ then $p \leq p'$. Said in a more informal manner: A network of organizations is secure if organizations within the network never observe a violation of their policies. The other definitions are more technical but logically equivalent, to keep the summary brief we do not cover them here. We develop several useful tools for reasoning about networks of the kind discussed above, notably, an induction principle on derived self connections within the network.

We show that several subtypes of networks are always secure, in particular: Networks that behave as if they had forest topology are secure, and thus as a corollary to this fact networks with forest topology are also secure. Further, we also show that networks that are forests of secure subnetworks are also secure, essentially a higher order version of the prequel.

We connect this framework of secure networks to an operation model similar to the one that Nielson et al. presents, i.e., a small steps semantics for a distributed system over the networks discussed herein with interleaved execution and synchronized communication. We define noninterference and progress-insensitive noninterference for this operational model in a manner similar to Askarov and Chong: The definition is parametric in an attacker located at one of the organizations. The attacker has access to the code of the programs running at all organizations, and can observe the value of variables that he as access to at the program running at the organization where he resides. The goal of the attacker is to infer the initial configuration of the memory for the program that he is partially observing. The strength of

the attacker may vary, and thus programs can be secure against some attackers but not others. Noninterference is defined in terms of the attackers knowledge or rather what he is allowed to learn. The attacker may only learn in accordance to the local policy, i.e., they may never learn more than they would by observing the parts of the initial configuration for which they had access. We do not provide a soundness result, i.e., we do not show that a distribuited program running in a secure network provides noninterference whenever individual processes observe noninterference in isolation.

The thesis is concluded by considering the viability of the framework and by examining avenues for future work besides the prequel point. To this extend, we note the generality of the framework, i.e., the framework is generalizable to infinite networks of Lagois connection over partial orders when identification of organizations is decidable. Further, results herein do no depend on **LC3** and **LC4**, and we discuss their inclusion/exclusion in a short note. The posiblity of adapting Nielson et al. type system to our setting is conjectured to be possible. We discuss extending the framework with dynamism of organizations and dynamism of policies, i.e., the ability of organizations to drop and establish new connections and to update their policies, respectively. To model this we propose to use a special type of function, in this regard we provide a cursory note on the matter. Finally, we consider that one can arrive at different notions of noninterference by tweaking the assumptions placed on the attacker.

The theory within the thesis has been developed in a type theoretic setting and its content has largely been proved in the proof assistant Coq without assuming any additional axioms. Notably, the fact that forest behaved networks are secure and the fact that networks with forest topology are secure has been proven in Coq.

Extending Lagois Connections for Secure Information Flow to n Organizations

Casper Ståhl

Aalborg University, Aalborg, Denmark
cstahl20@student.aau.dk

Abstract

In this paper we extend the framework of Lagois connection for secure information flow to an arbitrary number of organizations. In particular, we extend the framework to arbitrary network topologies such that the communication of connected organizations respects a Lagois connection. We identify several definitions of a secure network and show that these are all logically equivalent. Not all networks are secure, in this regard we identify several subtypes of secure networks. In particular we show that: Networks with forest topology, networks that behave as if they were a forest, and networks that are a forest of secure subnetworks are secure. We attach our framework to a operational model and define noninterference and progress-insensitive noninterference in a suitable manner. We do not provide an enforcement mechanism and neither do we prove a soundness result, but we conjecture that both are possible.

1 Introduction

Lagois connections provide a framework for secure, precise and converging connections between two organizations wishing to engage in secure communication with respect to noninterference [2, 3]. The framework is fairly developed for two organizations communicating in isolation and there are extensions that allow for an arbitrary finite amount of organizations to connect securely [2, 3]. However, as we will show in Section 3, current methods have a flaw in how they are proved to be secure and more importantly they have a problem with label inflation, or from another perspective loss of label information.

In this paper we seek to rectify the problems mentioned in the prequel by studying arbitrary networks of organizations that are pairwise connected in a secure manner by a Lagois connection. The security of such networks is decidable and can generally be determined in cubic time. More interestingly, certain networks turn out to always be secure. In particular, networks with forests topology, networks that behave as if they were a forest, and secure subnetworks connect in a forest turn out to always be secure. Further, we have proved formally in Coq that forest- and forest behaved networks are secure. Definition and results that have not been formalized in Coq are marked with an "*" throughout the paper. We make strides to establish a soundness proof with respect to noninterference by connecting our framework to an operational model and establishing a suitable definition of noninterference. However, we do not provide a soundness result, i.e., we do not prove that a secure network of programs all exhibiting noninterference in isolation guarantees noninterference of the distributed program as a whole. Neither do we provide an enforcement mechanism, but we conjecture that both are possible.

In Section 2 we cover the preliminaries required to understand the rest of the paper, in particular dependent type theory, order theory and graph theory. In Section 3 we further specify the problem with current methods for extension. In Section 4 we extend Lagois connection to arbitrary networks beyond two organizations and we develop a suitable notion of security. Not all graphs turn out to be secure, and thus we develop methods for determining if a network

is secure in the same section. In Section 5 we identify subtypes of networks that are secure, specifically those discussed above. In Section 6 we connect our framework to an operational model and define a suitable definition of noninterference. In Section 7 we cover notable highlights from the Coq formalization. In Section 8 we outline related works, specifically [2, 3, 6, 8, 9]. Finally, in Section 9 we conclude the paper.

2 Preliminaries

The issues presented in the previous section are quite technical in their nature. For this reason, before we can begin to precisely state the issue and develop a solution, we need to cover some preliminaries.

2.1 Dependent Type Theory

The theory within this paper has in large been developed natively in Coq, and thus in an intensional type theoretic setting. The "informal" presentation given in this paper closely mimics the formalization, and thus we present the type theory needed to understand the paper here. The presentation given here is mainly based on chapter 1 of [10].

A type A is a collection of objects freely generated by the constructors of A. If a was freely generated by the constructors of A we denote it $a : A$ and say that a is an inhabitant of A. The constructors of a type is a collection of (dependent) functions into the type. Several types are ubiquitous in dependent type theory and need description, but before we can describe these we need to describe the universe of types $\mathcal{U}$. Although $\mathcal{U}$ is usually a hierarchy of types $\mathcal{U}(0) : \mathcal{U}(1) : \ldots$, for the purpose of understanding this paper it is enough to consider $\mathcal{U}$ as just the type of types.

If A and B are types then $A \to B$ is the type of functions from inhabitants of A to inhabitants of B. Thus if $f : A \to B$ and $x : A$ then $f(x) : B$. We call $P : A \to \mathcal{U}$ a type family. For some type A and a type family $P : A \to \mathcal{U}$ the dependent function type $\prod_{(x:A)} P(x)$ is the type of functions that maps an inhabitant $x : A$ to an inhabitant in $P(x)$. In particular, if $f : \prod_{(x:A)} P(x)$ then $f(x) : P(x)$. By the Curry–Howard correspondence, types can be seen as propositions and inhabitants of a type as a proof of that proposition, thus if $A : \mathcal{U}$ then A is a proposition and if $x : A$ then x is a proof of A. In this reading, a type family $P : A \to \mathcal{U}$ is a predicate of inhabitants in A, in particular if $P(x)$ is inhabited for some $x : A$ we might say that P is true for x. Further, we can read $\prod_{(x:A)} P(x)$ as "P is true for all x inhabiting A". Similarly, $A \to B$ can be understood as "A implies B".

For some type A and a type family $P : A \to \mathcal{U}$ we let $\sum_{(x:A)} P(x)$ denote the dependent product. The dependent product has one constructor $\mathsf{exists} : \prod_{(x:A)} P(x) \to \sum_{(x:A)} P(x)$ but we will denote $\mathsf{exists}(x, y)$ as (x, y). As the name of the constructor hints at, $\sum_{(x:A)} P(x)$ can be understood as the constructive equivalent of "there exists $x : A$ such that $P(x)$". An inhabitant $(y, z) : \sum_{(x:A)} P(x)$ is thus an object y and a proof that y satisfies P in particular the proof $z : P(y)$. The dependent product has two important functions The first projection $\mathsf{pr}_1 : (\sum_{(x:A)} P(x)) \to A$ and the second projection

$$\mathsf{pr}_2 : \prod_{(p:\sum_{(x:A)} P(x))} P(\mathsf{pr}_1(p))$$

that respectively return the first and second element of a dependent pair. For some type B, we denote $\sum_{(x:A)} B$ as $A \times B$ or $A \wedge B$. When read as a proposition $A \wedge B$ denotes "A and B".

For two types A and B we let $A + B$ or $A \vee B$ denote the coproduct type, with constructors $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$. When read as a proposition $A \vee B$ denotes "A or B".

Within this paper we usually denote constructors as inductive definitions. For example, for some types A, B and type families $P : A \to \mathcal{U}$, $Q : B \to \mathcal{U}$ and $R : A \to B \to \mathcal{U}$

$$\frac{P(x) \qquad Q(y)}{R(x,y)} \quad (\mathsf{cons})$$

would give rise to a constructor $\mathsf{cons} : \prod_{(x:A)} \prod_{(y:B)} P(x) \to Q(y) \to R(x,y)$.

We do not assume the axiom of functional extensionality, and thus we need to rely on the notion of *homotopy* of functions instead:

Definition 1 (Homotopic)**.** $f : A \to B$ and $g : A \to B$ are said to be *homotopic* whenever $\prod_{(x:A)} f(x) = g(x)$, denoted $f \sim g$.

We do not assume the axiom of exclude middle, or any variation thereof, but at times it is useful to assume it conditionally. Thus for some proposition P we say that P is decidable if $P \vee \neg P$ where $\neg P :\equiv P \to \mathbf{0}$ and $\mathbf{0}$ denotes falsehood, i.e., the type with no constructors.

We regard a set over some type A to be a type family $S : A \to \mathcal{U}$, such that for $x : A$ we have $x \in S$ if $S(x)$ is inhabited. Thus $\{m \mid E\}$ denotes $\lambda m.E$, and subset, union, and intersection are defined in the expected manner.

2.2 Order Theory

The definition of a Lagois connection relies on several concepts from order theory, in particular partial orders, monotonic functions and poset systems, thus we describe these here.

Definition 2 (Partial order)**.** A type A is a partial order when it is equipped with a decidable relation $\leq$ such that

- $\prod_{(x:A)} x \leq x$ ($\leq$ is reflexive),
- $\prod_{(x\ y\ z:A)} x \leq y \to y \leq z \to x \leq z$ ($\leq$ is transitive), and
- $\prod_{(x\ y:A)} x \leq y \to y \leq x \to x = y$ ($\leq$ is antisymmetric).

Definition 3 (Monotonic function)**.** For two partial orders P and Q, a function $f : P \to Q$ is monotone whenever $p \leq p'$ implies $f(p) \leq f(p')$.

Definition 4 (Posest system)**.** A *poset system* is two partial orders P and Q with monotonic functions $f : P \to Q$ and $g : Q \to P$, denoted (P, f, g, Q).

Security with respect to information flow is typically described in terms of lattices, as prescribed by Denning [4].

Definition 5 (Lattice)**.** A *lattice* is a partial order A equipped with a join operator $- \sqcup - : A \to A \to A$ and a meet operator $- \sqcap - : A \to A \to A$ such that

- $\prod_{(x\ y\ z:A)} x \sqcup y \leq z \longleftrightarrow x \leq z \wedge y \leq z$, and
- $\prod_{(x\ y\ z:A)} x \leq y \sqcap z \longleftrightarrow x \leq y \wedge x \leq z$.

In regards to information flow, the lattices are assumed to be finite and inhabited, i.e., for a lattice A a least upper bound and greatest lower bound can be determined decidedly for any set over a A. Taking Dennings perspective, for a lattice A information is allowed to flow from something labeled $x : A$ to something labeled $y : A$ if $x \leq y$. This something with label x could for example be a document, and the other something labeled y could be an employee. Typically the somethings are program variables in an empirical programming language.

2.3 Graph Theory

As we explain in Section 1 we are motivated to study arbitrary network topologies. In our formalism we model network topologies as graphs and we rely one some "standard" definitions, therefore we cover these here.

Definition* 6 (Graph)**.** A type G with decidable equality is a graph if it is equipped with a decidable edge relation denoted $\nrightarrow$ where:

- $v \nrightarrow v$ is false for all $v : G$,
- if $v \nrightarrow v'$ then $v' \nrightarrow v$ for all $v\ v' : G$,

,i.e, the edge relation is irreflexive and symmetric.

If $v : G$, we will say that v is a vertex of G. Further, if $v \nrightarrow v'$, we say that there is an edge between v and v'. The graphs under consideration are undirected in the sense that their edge relation is symmetric. Further, these graphs contain no loop edges because their associated edge relation is irreflexive.

Definition* 7 (Path)**.** The type of paths $- \rightsquigarrow - : G \to G \to \mathcal{U}$ in a graph G is inductively defined by

$$\frac{}{v \rightsquigarrow v}\ (\epsilon), \qquad \frac{v \nrightarrow v' \qquad v' \rightsquigarrow v''}{v \rightsquigarrow v''}\ (\mathsf{pcons}).$$

Equivalently $v_1 \rightsquigarrow v_n$ is the type of sequences $v_1, v_2, \ldots, v_n$ where $v_i \nrightarrow v_{i+1}$ which we will denoted $v_1 \nrightarrow v_2 \nrightarrow \ldots \nrightarrow v_n$, i.e., the proofs that $v_i \nrightarrow v_{i+1}$ are kept implicit. We will mainly be working with two operations on paths, reversal and concatenation. For some vertices $v\ v' : G$, reversal has type:

$$-^{-1} : (v \rightsquigarrow v') \to (v' \rightsquigarrow v), \tag{1}$$

and is intuitively defined as

$$(v_1 \nrightarrow v_2 \nrightarrow \ldots \nrightarrow v_n)^{-1} :\equiv v_n \nrightarrow \ldots \nrightarrow v_2 \nrightarrow v_1. \tag{2}$$

Formally it is defined by recursion on the path. For some vertices $v\ v'\ v'' : G$, concatenation has type

$$- \star - : (v \rightsquigarrow v') \to (v' \rightsquigarrow v'') \to (v \rightsquigarrow v'') \tag{3}$$

and is intuitively defined as

$$v \nrightarrow \ldots \nrightarrow v' \star v' \nrightarrow \ldots \nrightarrow v'' :\equiv v \nrightarrow \ldots \nrightarrow v' \nrightarrow \ldots \nrightarrow v'' \tag{4}$$

Formally it is defined by recursion on the left path. I will also be convenient to know the length of a path $|-| : (v \rightsquigarrow v') \to \mathbb{N}$. For some vertices $v\ v' : G$ and a path $f : v \rightsquigarrow v'$, $|f|$ is intuitively determined by counting the number of edges appearing in f.

Penultimate, certain paths turn out to be useful, namely simple paths and loops.

Definition* 8 (Simple path)**.** A path $f : v \rightsquigarrow v'$ is *simple* if each vertex within is distinct.

For all $v : G$, we call a path $f : v \rightsquigarrow v$ a loop. The only simple loop in $v \rightsquigarrow v$ is ϵ, as otherwise v would appear twice: At the beginning of the sequence and at the end.

Finally, certain graphs, in particular forests, turn out to be of special interest.

Definition* 9 (Forest)**.** A graph G is called a forest if for all two points all simple paths between are unique, i.e., if $f\ g : v \rightsquigarrow v'$ are simple then $f = g$ for all $v\ v' : G$.

Remark 1. As explained in Section 7, the Coq formalization does not contain a direct representation of the mathematical object presented within this subsection. Instead the formalization only contains their extension to Lagois graphs as presented in Section 4.

3 Motivation

With the preliminaries covered, we can now state the problem at hand in a precise manner. Imagine two organizations v and v'. Both v and v' employ a security lattices, $L(v)$ and $L(v')$ respectively, to enforce their local policy for secure information flow as prescribed by Denning [4]. Now v and v' wish to communicate in a secure manner as not to violate their local policy with back and forth communication. Lagois connections, as presented by Bhardwaj and Prasad [2, 3], provide such a framework.

Definition 10 (Lagois connection [6])**.** A poset system (P, f, g, Q) is called a Lagois connection whenever:

$$p \leq (g \circ f)(p) \text{ for all } p : P, \tag{LC1}$$
$$q \leq (f \circ g)(q) \text{ for all } q : Q, \tag{LC2}$$
$$f \circ g \circ f \sim f, \text{ and} \tag{LC3}$$
$$g \circ f \circ g \sim g. \tag{LC4}$$

We can view a Lagois connection $(L(v), f, g, L(v'))$ as a secure, precise and converging transformation/translation between the labels of the two lattices, that is, we can use $f : L(v) \to L(v')$ and $g : L(v') \to L(v)$ to move somewhat freely between $L(v)$ and $L(v')$ [2, 3]. Equivalently we can see a Lagois connection $(L(v), f, g, L(v'))$ as a policy for cross communication between v and v'. In particular, information in v labeled $p : L(v)$ is allowed to flow as specified by the lattice $L(v)$ but also from p to $q : L(v')$ whenever $f(p) \leq q$ (analogous for g), and via the transitive closure of the two prequel relations. **LC1** and **LC2** ensure that the connection is secure:

$$p \leq (g \circ f)(p) \text{ for all } p : P \tag{SC1}$$
$$q \leq (f \circ g)(q) \text{ for all } q : Q \tag{SC2}$$

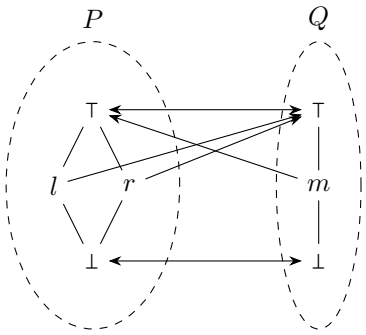
Intuitively, translating a label back and forth will not violate the local policy or equivalently sending a piece of information back and forth will never accidentally declassify it. Our development is strictly concerned with security and does not depend on **LC3** and **LC4**, therefore for the purpose of this paper they are of little interest. We include and justify them here for completeness. Appendix A contains a discussion of the inclusion/exclusion of **LC3** and **LC4**. The addition of **LC3** and **LC4** ensures that the connection is precise:

$$f(p) = \bigsqcup g^{-1}(p) \text{ for all } p \in g[Q] \tag{PC1}$$
$$g(q) = \bigsqcup f^{-1}(q) \text{ for all } q \in f[P] \tag{PC2}$$

Where $g^{-1}(p)$ denotes the preimage of p with respect to g and $g[Q]$ denotes the image of Q with respect to g. Further, **LC3** and **LC4** ensures that the connection is converging:

$$g(f(p)) = p' \to g(f(p')) = p' \text{ for all } p\, p' : P \tag{CC1}$$
$$f(g(q)) = q' \to f(g(q')) = q' \text{ for all } q\, q' : Q \tag{CC2}$$

Example 1. The figure below displays a poset system (P, f, g, Q) that is a Lagois connection. P and Q are shown as their Hasse diagram. f is indicated by arrows going from P to Q and vice versa for g.



Communication rarely occurs between two organizations in isolation, therefore we wish to extend the framework of Lagois connection for secure information flow to an arbitrary (finite) number of organizations. Bhardwaj and Prasad propose composition of Lagois connections as a method for such an extension [3]. Composition of Lagois connections is defined as:

$$(P, f, g, Q) \circ (Q, \hat{f}, \hat{g}, R) \coloneqq (P, \hat{f} \circ f, \hat{g} \circ g, R) \tag{5}$$

In this regard Bhardwaj and Prasad presents the following theorem originally derived by Melton et al. [3, 6].

Theorem* 1 (Theorem 11 in [3])**.** *Let* (P, f, g, Q) *and* $(Q, \hat{f}, \hat{g}, R)$ *be Lagois connections. Then* $(P, \hat{f} \circ f, \hat{g} \circ g, R)$ *is secure iff* $\hat{g} \circ \hat{f} \circ f[P] \subseteq f[P]$ *and* $f \circ g \circ \hat{g}[R] \subseteq \hat{g}[R]$

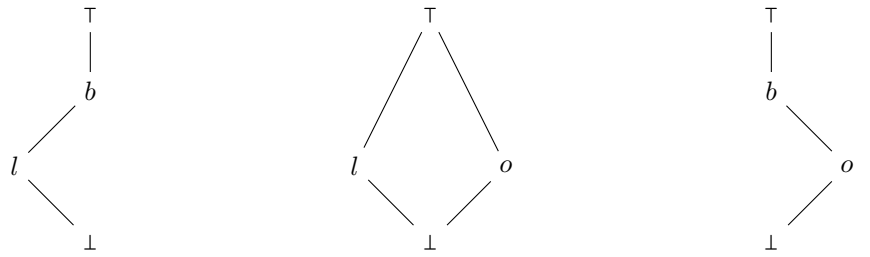
Bhardwaj and Prasad mischaracterize the original theorem (theorem 3.22 in [6]): The underlined "secure" originally stated "Lagois connection" in the version by Melton et al. While technically correct this misses an important point, indeed, the composition of secure connections is always secure (at least in the sense that Bhardwaj and Prasad present):

Proposition* 1. *Let* (P, f, g, Q) *and* $(Q, \hat{f}, \hat{g}, R)$ *satisfy* ***LC1*** *and* ***LC2****. Then* $(P, \hat{f} \circ f, g \circ \hat{g}, R)$ *satisfies* ***LC1*** *and* ***LC2****.*

Proof. Fix a $p : P$. Have $f(p) \leq (\hat{g} \circ \hat{f} \circ f)(p)$ by **LC1** on $(Q, \hat{f}, \hat{g}, R)$. Then, by monotonicity of g and **LC1** on (P, f, g, Q) we have $p \leq (g \circ f)(p) \leq (g \circ \hat{g} \circ \hat{f} \circ f)(p)$ i.e. **LC1**. The proof of **LC2** is analogous. □

The notion that $(P, \hat{f} \circ f, g \circ \hat{g}, R)$ is secure if (P, f, g, Q) and $(Q, \hat{f}, \hat{g}, R)$ are secure is a bit dubious, at least in the way it is presented in Theorem 1 and Proposition 1: Consider that Q's order might be violated by the traffic it mediates. This turns out not to be the case, but the results stated in the prequel do not directly reflect this.

More importantly, this way of chaining Lagois connections to connect mulitple organizations securely also has an unintended side effect as each translation of a label has the potential to incur a loss of label information. Consider the three lattices P, Q and R seen in Figure 1.



(a) Lattice P. (b) Lattice Q. (c) Lattice R.

Figure 1: Three lattices with similar structure.

If we establish the most precise Lagois connection between P, Q and between Q, R, denoted (P, f, g, Q) and $(Q, \hat{f}, \hat{g}, R)$ respectively, i.e., the ones where $f[P]$ and $\hat{f}[Q]$ (or equivalently $g[Q]$ and $\hat{g}[R]$) are as large as possible. Then, the derived poset system $(P, \hat{f} \circ f, g \circ \hat{g}, R)$ depicted in Figure 2, is not as precise as possible, essentially we would want $(\hat{f} \circ f)(b) = b$ and $(g \circ \hat{g})(b) = b$ but this is lost due to having R as an intermediate and we get $(\hat{f} \circ f)(b) = \top$ and $(g \circ \hat{g})(b) = \top$ instead.

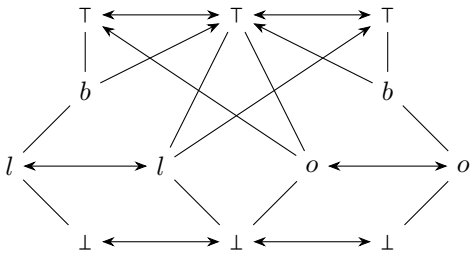


Figure 2: A misbehaved chain of Lagois connections.

For the lattices P, Q and R discussed here, swapping the order of the chain does not help either, observe Figure 3. Here the same problem occurs but for the point l.

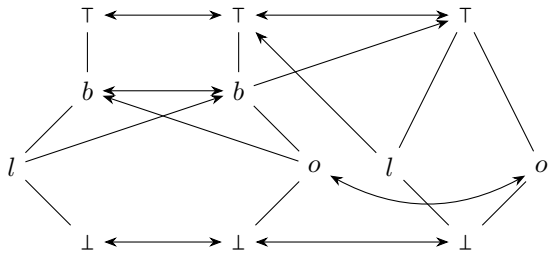


Figure 3: Another misbehaved chain of Lagois connections.

Essentially we want to bypass the intermediate lattice when possible, this motivates us to study graphs where each point has an associated lattice, and when two points are connected in the graph their underlying lattices are connected by some Lagois connection. Continuing

our example, Figure 4 shows such a graph. Here P, Q and R can communicate directly with each other, without incurring a loss of label information caused by an intermediate. Is this secure[1]? In this case the pattern of communication is secure, but generally this is not the case[2]. Therefore we seek to identify which graphs are secure.

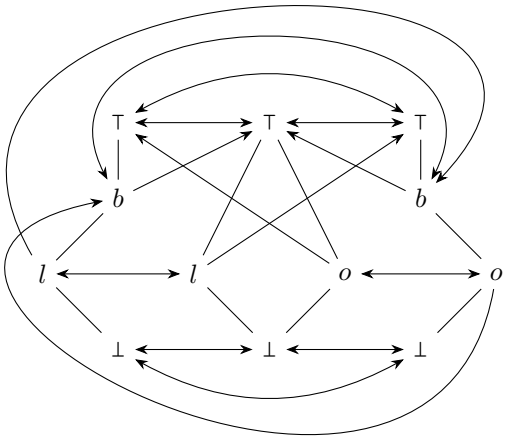


Figure 4: A Lagois graph.

4 Lagois Graphs

Here we define formally the concept described at the end of Section 3.

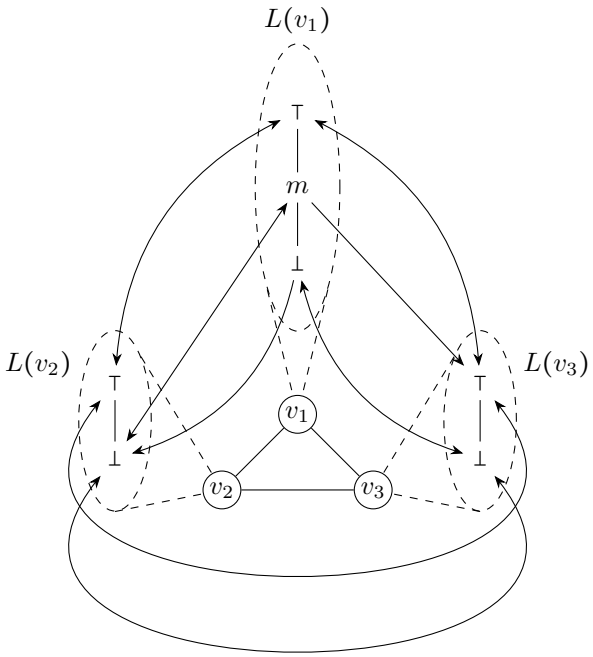
Definition 11 (Lagois graph)**.** A *Lagois graph* is a finite graph G where:

- Each $v : G$ vertex has an associated finite inhabited lattice denoted $L(v)$,
- each edge $v \nrightarrow v'$ has an associated monotonic function $f : L(v) \to L(v')$ denoted $v \overset{f}{\nrightarrow} v'$, and
- for each pair of symmetric edges $v \overset{f}{\nrightarrow} v'$, $v' \overset{g}{\nrightarrow} v$ it is the case that $(L(v), f, g, L(v'))$ is a Lagois connection.

For some Lagois graph G we can view each $v : G$ as an organization, actor, system, etc. who wishes to communicate securely with other $v' : G$. Each organization $v : G$ has an associated security lattice $L(v)$ that specifies their local policy. When two organizations v $v' : G$ are connected in the graph we assume that they have negotiated a policy for secure information exchange forming a Lagois connection $(L(v), f, g, L(v'))$ for some $f : L(v) \to L(v')$ and $g : L(v') \to L(v)$.

Example 2. The diagram below displays a Lagois graph G. The graph contains three elements v_1 v_2 $v_3 : G$ which are drawn as the innermost three nodes in the diagram. The lattices $L(v_1), L(v_2), L(v_3)$ are displayed around the graph, and the functions inducing the Lagois connections are indicated by the arrows.

[1] See Definition 13 for a definition of security.
[2] See Example 2 for a graph that is not secure.



For a Lagois graph two notions of security arises: One akin to the one presented in [8, 9]. And one akin to **SC1** (or **SC2**). As we will later show, these two notions turn out to be logically equivalent.

The first notion of security is described in terms of a flow relation that is induced by the lattices and by the connections that the graph is equipped with.

Definition 12 (Flow relation)**.** Each Lagois graph G induces a flow relation for $v\ v' : G$ and $p : L(v), q : L(v')$ with judgments $(v,p) \rightrightarrows (v',q)$ which is inductively defined as follows:

$$\frac{p \leq q}{(v,p) \rightrightarrows (v,q)}\ \textsf{(flowle)}, \qquad \frac{v \overset{f}{\mapsto} v'}{(v,p) \rightrightarrows (v', f(p))}\ \textsf{(flowlc)}$$

$$\frac{(v,p) \rightrightarrows (v',r) \qquad (v',r) \rightrightarrows (v'',q)}{(v,p) \rightrightarrows (v'',q)}\ \textsf{(flowtran)}.$$

For $(v,p) \rightrightarrows (v',q)$ we drop v, v' when they can be inferred from the context, giving judgments $p \rightrightarrows q$.

Rule flowle tells us that information is allowed to flow according to local policies. Rule flowlc allows information to flow via the Lagois connections established by the Lagois graph. And rule flowtran is the transitive closure. We can now define a notion of security similar to [8, 9].

Definition 13 (Flow secure)**.** A graph G is said to be *flow secure* if for all $v : G$ and $p\ p' : L(v)$ it is the case that $p \rightrightarrows p'$ implies $p \leq p'$.

Essentially, a graph G is flow secure if for all vertices $v : G$ all flows that start and end at v respects v's local policy as specified by $L(v)$. As stated in Section 1, not all graphs are flow

secure, the graph displayed in Example 2 is a witness to this fact. Observe that by rule flowlc $m_{v_1} \rightrightarrows \bot_{v_2}, \bot_{v_2} \rightrightarrows \bot_{v_3}, \bot_{v_3} \rightrightarrows \bot_{v_1}$ and thus by rule flowtran $m_{v_1} \rightrightarrows \bot_{v_1}$ which violates the local policy of v_1 because $m_{v_1} \not\leq \bot_{v_1}$. By a similar argument as the one put forward in [8], we can determine if a graph is flow secure in cubic time.

We will now develop the second notion of security which is similar to **SC1** (or **SC2**). This notion of security turns out to be a stepping stone towards further results and as we will show is also logically equivalent to flow security. Before we can define this type of security we need extend the notion of a path inside a Lagois graph. A path in a Lagois graph $v_1 \twoheadrightarrow v_2 \twoheadrightarrow \ldots \twoheadrightarrow v_n : v_1 \rightsquigarrow v_n$ is additionally extended with the monotonic functions that appear along its edges denoted $v_1 \overset{f}{\twoheadrightarrow} v_2 \overset{g}{\twoheadrightarrow} \ldots \overset{h}{\twoheadrightarrow} v_n : v_1 \rightsquigarrow v_n$ A path $v_1 \overset{f}{\twoheadrightarrow} v_2 \overset{g}{\twoheadrightarrow} \ldots \overset{h}{\twoheadrightarrow} v_n : v_1 \rightsquigarrow v_n$ can thus be coerced to a function $h \circ \cdots \circ g \circ f : L(v_1) \to L(v_n)$ with the empty path $\epsilon : v \rightsquigarrow v$ being coerced to the identity function $\mathsf{id} : L(v) \to L(v)$. Formally the coercion is defined by recursion on the path. From now on the coercion will be implicit when this causes no confusion. We can now state the second notions of security.

Definition 14 (Loop secure)**.** A loop $f : v \rightsquigarrow v$ is *loop secure* whenever $p \leq f(p)$ for all $p : L(v)$. A vertex $v : G$ is *loop secure* if for all $f : v \rightsquigarrow v$, f is loop secure. A Lagois graph G is *loop secure* if for all $v : G$, v is loop secure.

As mentioned previously a graph being flow secure is logically equivalent to it being loop secure, as the following lemma and proposition shows.

Lemma 1. *If $p \rightrightarrows q$ for some $p : L(v)$, $q : L(v')$, then there exists $f : v \rightsquigarrow v'$ such that $f(p) \leq q$.*

Proof. By induction on the derivation of $p \rightrightarrows q$. □

Proposition 2. *A graph is flow secure iff it is loop secure.*

Proof.

→: Fix $v : G, f : v \rightsquigarrow v$ and $p : L(v)$. Because there is a loop $f : v \rightsquigarrow v$ there is a flow $p \rightrightarrows f(p)$. Thus by our assumption that flow is secure we have $p \leq f(p)$.

←: Fix $v : G, p\ p' : L(v)$ and assume that $p \rightrightarrows p'$. By Lemma 1 there exists $f : v \rightsquigarrow v$ such that $f(p) \leq p'$. Thus we have $p \leq f(p) \leq p'$ by our assumption that all loops are secure.

□

From now on, when it does not cause confusion, we will refer to a graph as secure if it is either flow secure or loop secure, as they are logically equivalent. This notion of secure loops is not very nice for practical use, as a non-trivial graph induces an infinite number of loops. However it allows us to proceed by a particular type of induction, induction on loops.

Theorem 2 (Loop Induction)**.** *For $P : \prod_{(v:G)}(v \rightsquigarrow v) \to \mathcal{U}$ to prove $P(v, f)$ for all $v : G$ and $f : v \rightsquigarrow v$ it suffices to prove:*

1. *$P(v, \epsilon)$ for all $v : G$,*
2. *$P(v, \mathsf{pcons}(f, g))$ for all $v\ v' : G, f : v \overset{f'}{\twoheadrightarrow} v', g : v' \rightsquigarrow v$ such that g is simple, and*
3. *$P(v, f_1 \star f_2)$ and $P(v', g)$ implies $P(v, f_1 \star g \star f_2)$ for all $v\ v' : G, f_1 : v \rightsquigarrow v', g : v' \rightsquigarrow v', f_2 : v' \rightsquigarrow v$.*

Proof. Fix v and $f : v \rightsquigarrow v$. Either $f = \epsilon$ or $f = \mathsf{pcons}(g, h)$ for some $g : v \overset{g'}{\mapsto} v'$ and $h : v' \rightsquigarrow v$.

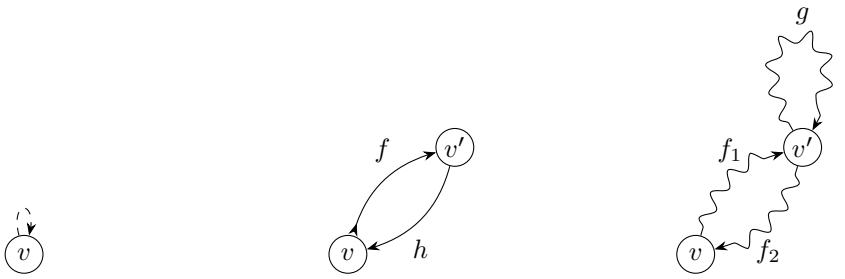
Case $f = \epsilon$: By assumption 1.

Case $f = \mathsf{pcons}(g, h)$: Either h is simple or it is not.

Case h is simple: By assumption 2.

Case h is not simple: In this case we can decompose $h = h_1 \star h_2 \star h_3$ such that $h_2 : v' \rightsquigarrow v'$ is a nonempty loop, thus f can be decomposed into $f = \mathsf{pcons}(g, h_1) \star h_2 \star h_3$. Clearly $|\mathsf{pcons}(g, h_1) \star h_3| \leq |f|$ and $|h_2| \leq |f|$ and thus by mathematical induction we can assume $P(v, \mathsf{pcons}(g, h_1) \star h_3)$ and $P(v', h_2)$ and therefore by assumption 3 we have $P(v, f)$.

□



(a) Case 1. (b) Case 2. (c) Case 3.

Figure 5: The three different cases of loop induction.

The three different cases appearing in loop induction are depicted diagrammatically in Figure 5. Empty paths are indicated by a dashed edge, paths that are a single edge are indicated by a smooth arrow with a tail, simple paths are indicated by a smooth arrow, and general paths are indicated by a squiggly arrow. We can now derive a third notion of security, that again will turn out to be logically equivalent to our other notions of security.

Definition 15 (Simply secure)**.** A Lagois graph G is *simply secure* whenever for all pairs of vertices $v\ v' : G$ and for all simple paths $f : v \rightsquigarrow v'$ and $g : v' \rightsquigarrow v$ between these it is the case that $f \star g$ is secure.

This notion might seem superfluous, but in the next section we will use it to prove that certain graph topologies are always secure without resorting to loop induction. The following lemma and proposition, as we hinted at in the prequel, show that security and simple security are logically equivalent.

Lemma 2. *A path $f : v \rightsquigarrow v'$ is always monotonic.*

Proof. By induction on f. □

Proposition 3. *A graph is simply secure iff it is secure.*

Proof.

←: A secure graph is simply secure because all of its loops are secure.

$\rightarrow$: Fix a loop f. We proceed by loop induction on f.

Case $f = \epsilon$: ϵ is trivially secure.

Case $f = \mathsf{pcons}(g, h)$ such that h is simple: If h is simple then f is secure because g is simple and because the graph is simply secure.

Case $f = f_1 \star g \star f_2$: By the inductive hypothesis we can assume g is secure, from which we can derive $f_1(p) \leq (f_1 \star g)(p)$. Then by monotonicity of paths and the inductive hypothesis that $f_1 \star f_2$ is secure we have $p \leq (f_1 \star f_2)(p) \leq (f_1 \star g \star f_2)(p)$.

□

It is probably the case that Proposition 3 can be derived without resorting to loop induction, even if this is the case we believe that loop induction might be useful for proving other properties of Lagois graphs and similar structures. We now have the tools at hand to show that a variety of graphs are secure, which we will do in the next section.

5 Secure Types of Graphs

We will now define the first type of Lagois graph that is always secure.

Definition 16 (Virtual Lagois forest)**.** A Lagois graph G is a *virtual forest* whenever for all vertices $v\ v' : G$ and for all simple paths $f : v \rightsquigarrow v'$ and $g : v \rightsquigarrow v'$ it is the case that $f \sim g$.

Essentially a virtual Lagois forest is a Lagois graph in which for all $v\ v' : G$, all simple paths from v to v' have the same extensional behavior when seen as functions. Said otherwise, if you go from v to v' via a simple path it doesn't matter which simple path you take. Before we can prove that such graphs are secure, we need to prove the following lemma.

Lemma 3. *$f \star f^{-1}$ is secure for all $f : v \rightsquigarrow v'$ and all $v\ v' : G$.*

Proof. By induction of f: □

Lemma 3 intuitively says that a flow of information is secure as long as it returns exactly the way it came. We can now prove that virtual Lagois forests are secure.

Theorem 3. *A virtual Lagois forest is secure.*

Proof. Fix a graph G, by Proposition 3 it is enough to consider all pairs of simple paths $f : v \rightsquigarrow v'$, $g : v' \rightsquigarrow v$ for all pairs of points $v\ v' : G$. We have $f^{-1} \sim g$ because G is a virtual forest. $f \star f^{-1}$ is secure by Lemma 3 and therefore $f \star g$ is also secure because $f^{-1} \sim g$. □

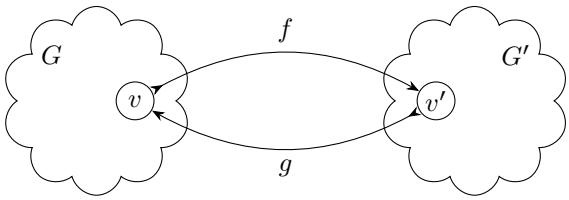
It immediately follows as a corollary that Lagois forests are secure, i.e., a Lagois graph that is a forest is secure.

Corollary 1. *A Lagois forest is secure.*

Proof. A virtual Lagois forest are secure by Theorem 3 and a Lagois forest is a virtual Lagois forest because $f = g$ implies $f \sim g$. □

We can now prove a kind of stronger version of Proposition 1 by observing that a chain of composed Lagois connections is essentially a Lagois graph that is a path graph, and that a path graph is a forest which is secure as we have just showed. Further, Corollary 1 is interesting because it tells us that a Lagois graph can be determined to be secure by looking at the structure of the graph alone. Taking another view, security of an entire network can emerge from security ensured entirely between organizations in isolation.

Certain composite graphs can also be shown to be secure assuming its parts are secure. In particular, consider for two Lagois graphs G and G', vertices $v : G$ and $v' : G'$ and a Lagois connection $(L(v), f, g, L(v')$ the composite graph $G \overset{f}{v}\!\rightleftharpoons\!\overset{v'}{g} G'$ that is defined as depicted below.

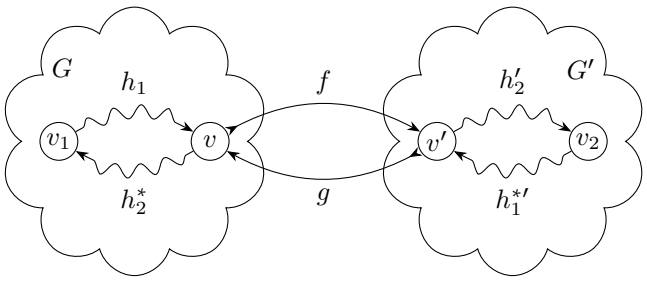


Formally we define $G \overset{f}{v}\!\rightleftharpoons\!\overset{v'}{g} G'$ as a graph with vertices $G + G'$ where $L(\mathsf{inl}(v_1)) = L(v_1)$ for all $v_1 : G$ (analogous for all $v_2 : G'$), and an edge relations such that

- $\mathsf{inl}(v_1) \overset{h}{\rightarrowtail} \mathsf{inl}(v_2) \longleftrightarrow v_1 \overset{h}{\rightarrowtail} v_2$ for all h and $v_1\ v_2 : G$,
- $\mathsf{inr}(v_1') \overset{h'}{\rightarrowtail} \mathsf{inr}(v_2') \longleftrightarrow v_1' \overset{h}{\rightarrowtail} v_2'$ for all h' and $v_1'\ v_2 : G'$, and
- $\mathsf{inl}(v) \overset{f}{\rightarrowtail} \mathsf{inr}(v')$ and $\mathsf{inr}(v') \overset{g}{\rightarrowtail} \mathsf{inl}(v)$ with no other edges across.

Theorem* 4. *If G and G' are secure graphs then for all $v : G$, $v' : G$ and f, g such that $(L(v), f, g, L(v'))$ is a Lagois connection then $G \overset{f}{v}\!\rightleftharpoons\!\overset{v'}{g} G'$ is secure.*

Proof. By Proposition 3 it is enough to consider all pairs of simple paths $h : v_1 \rightsquigarrow v_2$, $h^* : v_2 \rightsquigarrow v_1$ for all pairs of points $v_1\ v_2 : G \overset{f}{v}\!\rightleftharpoons\!\overset{v'}{g} G'$. If each point resides exclusively within G then h and h^* must also be contained within G, because if either h or h^* where to pass over into G' it must have been through $e_f : \mathsf{inl}(v) \overset{f}{\rightarrowtail} \mathsf{inr}(v')$ (and $e_g : \mathsf{inr}(v') \overset{g}{\rightarrowtail} \mathsf{inl}(v)$ on the way back) contradicting the fact that they are simple (analogous for vertices residing exclusively within G'). If v_1 resides within G and v_2 resides within G' we can decompose $h = h_1 \star e_f \star h_2'$ and $h^* = h_1^{*\prime} \star e_g \star h_2^*$ such that h_1, h_2^* are contained within G and $h_2', h_1^{*\prime}$ are contained within G', as depicted below.



Fix a $p : L(v_1)$, now $(h_1 \star e_f)(p) \leq (h_1 \star e_f \star h_2' \star h_1^{*\prime})(p)$ because $h_2' \star h_1^{*\prime}$ is secure by virtue of being a loop contained within G'. Then $h_1(p) \leq (h_1 \star e_f \star e_g)(p) \leq (h_1 \star e_f \star h_2' \star h_1^{*\prime} \star e_g)(p)$

by security of $e_f \star e_g$ and monotonocity of e_g. Finally,

$$\begin{aligned} p &\leq (h_1 \star h_2^*)(p) \\ &\leq (h_1 \star e_f \star h_2' \star h_1^{*\prime} \star e_g \star h_2^*)(p) \\ &= (h \star h^*)(p) \end{aligned}$$

because $h_1 \star h_2^*$ is secure by virtue of being contained within G and because h_2^* is monotone (analogous when v_1 resides in G' and v_2 resides in G). □

Essentially, Theorem 1 tells us that if a graph G is a forest of secure subgraphs $G_1, G_2, \ldots G_n$ then G is a secure graph. Theorem 4 is thus a higher order version of Corollary 1, in this regard we conjecture that Corollary 1 can be derived from Theorem 4 instead of Theorem 3.

6 Towards Soundness (with respect to Noninterference)

Our current framework is entirely detached from any kind of operational model. Although we consider this a strength of the approach, it also means that we can not establish a soundness result in regards to noninterference. In this section we will work towards such a result. We could proceed in a manner similar to Bhardwaj and Prasad [2, 3], but as we will also mention in Section 8 the programs that they consider are rather restricted, that is to say that we wish to consider several programs that communicate in a synchronized but nondeterministic manner and that communication can occur within arbitrary program constructs. Further, we wish to express soundness in general terms, i.e., not conditioned on the approval of some enforcement mechanism. For the operational model our approach is to define a network of programs over a Lagois graph G by a function $S : G \to \mathsf{Stmt}$ where Stmt are the type of programs under consideration. For $v : G$ we say that $S(v)$ is the program running at v. We will define the semantics of this network of programs in a manner similar to [8] minus the dynamism of relocating agents. Our notion of noninterference is parametric in the attacker, and is defined in terms of what an attacker may or may not learn as prescribed by Askarov and Chong [1]. We will not derive a soundness result and neither will we provide an enforcement mechanism, however we conjecture that both are possible. To be clear, by soundness we mean that a distributed system can be show to exhibit noninterference when communication respects the underlying Lagois graph and when each program in isolation exhibits noninterference. By enforcement mechanism we mean a type system, monitor, etc. that would ensure noninterference. Of course such an enforcement mechanism could also be shown to be sound. Soundness and enforcement mechanism are further discussed in Section 9.

6.1 Abstract Syntax

In this section we define the abstract syntax for the programs that will run at each node in a network. Take n to range over the natural numbers $\mathbb{N}$, take x to ranger over a type of variables Var equivalent but distinct to $\mathbb{N}$, and take c to range over a type of channels Ch equivalent but distinct to $\mathbb{N}$ and Var. We take e to range over expressions Exp, and we take s to range over statements Stmt which we define in the following manner.

$$e ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 = e_2 \mid \neg\, e \tag{6}$$

$$s ::= \texttt{skip} \mid x := e \mid c\,!\,e \mid c\,?\,x \mid \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while}\ e\ \texttt{do}\ s \mid s_1; s_2 \tag{7}$$

Like [8, 9] we take $c\,!\,e$ to denote that a program wishes to send e on channel c, and $c\,?\,x$ denotes that a program wishes to receive on channel c and store the result in variable x.

6.2 Small Step Semantics

In this section we will define the small step semantics of a distributed network with a network topology as described by some Lagois graph G. Generally the execution of individual programs are interleaved and communication is synchronized but nondeterministic. In a manner similar to [8, 9], we define the type of local signals Sig ranged over by φ as

$$\varphi ::= \varepsilon \mid u(x) \mid c\,!\,n \mid c\,?_n\,x \tag{8}$$

ε indicates no signal, $u(x)$ indicates a signal that variable x has been assigned to, $c\,!\,n$ signals that the program has sent value n on channel c, and $c\,?_n\,x$ signals that a program has received value n on channel c and is assigning the value to x.

We define $\mathsf{Mem} :\equiv \mathsf{Var} \to \mathbb{N}$ to be the type of memories ranged over by m. For an expression e we let $m(e) = n$ denote that e evaluates to n in environment m. We take st to range over local state i.e. the type $\mathsf{Stmt} \times \mathsf{Env}$. Further, we take $f[x \mapsto n]$ to be the function mapping x to n but all other values y to $f(y)$. We then define the semantics of local execution with judgments $st \Rightarrow_\varphi st'$ inductively below. The judgment $st \Rightarrow_\varphi st'$ should be read as: State st can in one step go to st' emitting signal φ.

$$\frac{m(e) = n}{(x := e, m) \Rightarrow_{u(x)} (\texttt{skip}, m[x \mapsto n])}\quad (\mathsf{ex\_assign}),$$

$$\frac{m(e) = n}{(c\,!\,e, m) \Rightarrow_{c\,!\,n} (\texttt{skip}, m)}\quad (\mathsf{ex\_out}),$$

$$\frac{}{(c\,?\,x, m) \Rightarrow_{c\,?_n\,x} (\texttt{skip}, m[x \mapsto n])}\quad (\mathsf{ex\_in}),$$

$$\frac{m(e) = 0}{(\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2, m) \Rightarrow_\varepsilon (s_2, m)}\quad (\mathsf{ex\_ifelse\_ff}),$$

$$\frac{0 \leq m(e)}{(\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2, m) \Rightarrow_\varepsilon (s_1, m)}\quad (\mathsf{ex\_ifelse\_tt}),$$

$$\frac{}{(\texttt{while}\ e\ \texttt{do}\ s, m) \Rightarrow_\varepsilon (\texttt{if}\ e\ \texttt{then}\ (s; \texttt{while}\ e\ \texttt{do}\ s)\ \texttt{else}\ \texttt{skip}, m)}\quad (\mathsf{ex\_while}),$$

$$\frac{(s_1, m) \Rightarrow_\varphi (s_1', m')}{(s_1; s_2, m) \Rightarrow_\varphi (s_1'; s_2, m')}\quad (\mathsf{ex\_seque\_cont}),$$

$$\frac{}{(\mathit{skip}; s, m) \Rightarrow_\varepsilon (s, m)}\quad (\mathsf{ex\_seque\_skip}).$$

In a manner similar to [1], we will define the type of events Ev ranged over by α as:

$$\alpha ::= \varepsilon \mid u(v, x) \tag{9}$$

ε is regarded as no event occurring and $u(v,x)$ as the event that x was assigned to in the program at v.

We take St to range over the type of states of a network $G \to \mathsf{Stmt}\times\mathsf{Env}$, such that $St(v) = st$ is the state of the program running at $v : G$. Judgments take the form $St \Rightarrow_\alpha St'$ which should be read as St can in one execution-step go to St' emitting the observable event α. The judgments are inductively defined as:

$$\frac{St(v) = st \qquad st \Rightarrow_\varepsilon st'}{St \Rightarrow_\varepsilon St[v \mapsto st']} \quad (\mathsf{ex\_sp}),$$

$$\frac{St(v) = st \qquad st \Rightarrow_{u(x)} st'}{St \Rightarrow_{u(v,x)} St[v \mapsto st']} \quad (\mathsf{ex\_lp}),$$

$$\frac{v_i \overset{f}{\rightarrowtail} v_j \qquad St(v_i) = st_i \qquad St(v_j) = st_j \qquad st_i \Rightarrow_{c\,!\,n} st_i' \qquad st_j \Rightarrow_{c\,?_n\,x} st_j'}{St \Rightarrow_{u(v_j,x)} St[v_i \mapsto st_i', v_j \mapsto st_j']} \quad (\mathsf{ex\_exch}).$$

The choice that v_i and v_j must be directly connected by an edge, i.e. $v_i \overset{f}{\rightarrowtail} v_j$ for some f, is somewhat arbitrary. We could alternatively require that v_i and v_j were connect by a path $f : v_i \rightsquigarrow v_j$ or we could drop the requirement entirely and relegate it to be handle by an enforcement mechanism.

6.3 Traces

In this section we define traces, informally a trace is a sequence of distributed states observable by an omniscient observer from program execution.

Definition 17 (Trace)**.** The type of traces from St to St' denoted $St \Rightarrow^\star St'$ is inductively defined by:

$$\frac{}{St \Rightarrow^\star St} \quad (\mathsf{exs\_refl}), \qquad \frac{St \Rightarrow^\star St' \qquad St' \Rightarrow_\alpha St''}{St \Rightarrow^\star St''} \quad (\mathsf{exs\_trans}).$$

Said otherwise, an element of $t : St_1 \Rightarrow^\star St_n$ is a sequence of states St_i for $i = 1 \ldots n-1$ such that $St_i \Rightarrow_{\alpha_i} St_{i+1}$

From now on we fix $loc : G$, level $lvl : L(loc)$, and security mapping $\Lambda : \mathsf{Var} \to L(loc)$. loc should be interpreted as the location of the attacker, and lvl the maximum security label that the attacker is allowed to observe given λ. More specifically, the attacker can observe variable x if $\Lambda(x) \leq lvl$.

Take $St_1 :\equiv \lambda v.\mathsf{pr}_1(St(v))$ and $St_2 :\equiv \lambda v.\mathsf{pr}_2(St(v))$. And let $[] : \mathsf{seq}(A)$ be the empty sequence and $- :: - : A \to \mathsf{seq}(A) \to \mathsf{seq}(A)$ the sequence constructor for some type A. Further, let $[x]$ denote the sequence $x :: []$.

Let $\mathsf{PMem} :\equiv \mathsf{Var} \rightharpoonup \mathbb{N}$ denote the type of partial memories ranged over by μ. For a memory m we define the partial memory observable by the attacker $\mathsf{obs}(m) : \mathsf{PMem}$ as:

$$\mathsf{obs}(m) :\equiv \lambda x. \begin{cases} m(x) & \text{if } \Lambda(x) \leq lvl \\ \text{nothing} & \text{otherwise} \end{cases} \tag{10}$$

For a trace t we define the observable trace for the attacker $\mathsf{Obs}(t)$ to be a sequence of partial memories observable by the attacker:

$$\mathsf{Obs}(\mathsf{ex\_refl}(St)) :\equiv [\mathsf{obs}(St_2(loc))] \tag{11}$$

$$\mathsf{Obs}(\mathsf{ex\_trans}(St, St', \varepsilon, St'', t, \_)) :\equiv \mathsf{Obs}(t) \tag{12}$$

$$\mathsf{Obs}(\mathsf{ex\_trans}(St, St', u(v,x), St'', t, \_)) :\equiv \begin{cases} \mathsf{obs}(St_2''(loc)) :: \mathsf{Obs}(t) & \text{if } loc = v \wedge \Lambda(x) \leq lvl \\ \mathsf{Obs}(t) & \text{otherwise} \end{cases} \tag{13}$$

We let $St \Downarrow \mu s$ denote that St can emit the sequence of partial states μs that are observable by the attacker and formally define it by:

$$St \Downarrow \mu s :\equiv \sum_{(St')} \sum_{(t : St \Rightarrow^\star St')} \mathsf{Obs}(t) = \mu s \tag{14}$$

6.4 Attacker

As [1] our notion of noninterference is parametric in the attacker. We define an attacker in the following manner:

Definition 18 (Attacker)**.** An *attacker* is a type Σ equipped with

- an initial state $\sigma_{init} : \Sigma$, and
- a transition function $\delta : \Sigma \to \mathsf{PMem} \to \Sigma$.

Similar to [1], we denote the state of an attacker after observing a sequence of partial memories μs as $A(\mu s)$ and define it as:

$$A([\,]) :\equiv \sigma_{init} \tag{15}$$

$$A(\mu :: \mu s) :\equiv \delta(A(\mu s), \mu) \tag{16}$$

From now on fix an attacker A. We define the knowledge of an attacker as:

$$k(S, \sigma) :\equiv \left\{ m \;\middle|\; \sum_{(St)} \sum_{(\mu s)} St \Downarrow \mu s \wedge St_2(loc) = m \wedge A(t) = \sigma \wedge \left(\prod_{(v)} St_2(v) = S(v) \right) \right\} \tag{17}$$

Essentially $k(S, \sigma)$ is the set of environments A considers possible knowing that S is the program running on the distributed system and σ is A's current state.

Further we define the knowledge of an attacker that has observed progress as

$$k^+(S, \sigma) :\equiv \left\{ m \;\middle|\; \sum_{(St)} \sum_{(\mu s)} \sum_{(\mu)} St \Downarrow (\mu :: \mu s) \wedge St_2(loc) = m \wedge A(t) = \sigma \wedge \left(\prod_{(v)} St_2(v) = S(v) \right) \right\} \tag{18}$$

We define a lower bound on the environments that the attacker is allowed to deduce as $[\![m]\!] :\equiv \{m' \mid m \approx m'\}$ over the following relation:

$$m \approx m' :\equiv \prod_{(x : \mathsf{Var})} \Lambda(x) \leq lvl \to m(x) = m(x') \tag{19}$$

Said otherwise $[\![m]\!]$ is a lower bound on the confusion of the attacker: The attack must never learn such that he is unconfused enough that he can begin to distinguish between elements in $[\![m]\!]$.

6.5 Noninterference

We are now in a position to define noninterference in a manner closely resembling that of [1].

Definition 19 (Noninterference)**.** A distributed program over a Lagois graph G with initial state St exhibits *noninterference* against attacker A with location $loc : G$ and level $lvl : L(loc)$ with respect to security mapping $\Lambda : \mathsf{Var} \to L(loc)$ whenever for all μs and μ such that $St \Downarrow (\mu :: \mu s)$ one has:

$$k(St_1, A(\mu :: \mu s)) \supseteq k(St_1, A(\mu s)) \cap [\![St_2(loc)]\!]. \tag{20}$$

Essentially, an initial state St exhibits noninterference with respect to an attacker A if at each step observable to the attacker the attacker only learns in accordance to the lower bound on his confusion. Said otherwise, the attacker must never learn more than what they would by observing the parts of the initial memory for which they had access. As with [1] this notion of noninterference is progress sensitive and we can thus define a progress insensitive version (as Askarov and Chong do):

Definition 20 (Progress-insensitive noninterference)**.** A distributed program over a Lagois graph G with initial state St exhibits *noninterference* against attacker A with location $loc : G$ and level $lvl : L(loc)$ with respect to security mapping $\Lambda : \mathsf{Var} \to L(loc)$ whenever for all μs and μ such that $St \Downarrow (\mu :: \mu s)$ one has:

$$k(St_1, A(\mu :: \mu s)) \supseteq k^+(St_1, A(\mu s)) \cap [\![St_2(loc)]\!]. \tag{21}$$

Essentially, the attacker is able to learn more than the lower bound on his confusion, as long as it is due to observing progress.

7 Formalization

As mentioned in Section 1, the majority of this paper has been formalized in Coq. In particular, definitions, theorems, lemmas, etc. that have not been formalized are marked with "*". Built upon Mathematical Components (and Hierarchy Builder), mainly for its development of order theory, the formalization does not depend on any additional axioms besides uniqueness of identity proofs (**UIP**) for a very specific type of identity. However, a few points within the formalization are worthy of discussion, in particular: The Lagois graph structure, the inclusion of **UIP**, and the proof of the loop induction principle (Theorem 2). We will cover these two in this section.

The definition of a Lagois graph is of particular interest, because the formalization contains no underlying notion of a graph:

```
1  HB.mixin Record IsLagoisGraph V of Equality V := {
2    lattice : V → {d & porderType d};
3    edge v v' : option (
4        Lagois.type (projT2 (lattice v)) (projT2 (lattice v')));
5
6    edge_irefl v : edge v v = None;
7
8    edge_sym v v' fg : edge v v' = Some fg →
9        {gf | edge v' v = Some gf ∧ fg.1 = gf.2};
10 }.
11 HB.structure Definition LagoisGraph :=
```

```
    {T of IsLagoisGraph T & Equality T}.
Notation "L( v )" := (projT2 (lattice v)).
Notation "v ↣[ f ] v'" :=
  (edge v v' = Some f) (at level 70, f at level 60).
Axiom squash_edge :
  forall (G : LagoisGraph.type) (v v' : G)
      f (e e' : v ↣[ f ] v'), e = e'.
```

Essentially, a Lagois graph is a type `V` equipped with a function `lattice` that maps points in `V` to their respective lattices. `edge` is the edge relation, and should be interpreted in the following manner: If `edge v v' = None` then there is not an edge between `v` and `v'`. However, if `edge v v' = Some fg`, then `fg` is a pair of functions `fg.1` and `fg.2` that form a Lagois connection over the respective lattices belonging to `v` and `v'`. This means that there is an edge from `v` to `v'` that has an associated function `fg.1` and that the edge going in the opposite direction has an associated function `fg.2` which is enforced by `edge_sym`. In this regard a proof of `edge v v' = Some fg` for some `fg` is considered an edge which we denote $v \overset{fg}{\rightarrowtail} v'$. `edge_irefl` enforces that there are no immediate self loops. The definition in the formalization is also more general, i.e., instead of points having a associated lattice they are only required to have an associated partial order. Further, we do not require that the underlying type that represents the vertices in the graph is finite, only that it has decidable equality.

Notice the axiom `squash_edge` which is essentially **UIP** for identity proofs of the form `edge v v' = Some fg`. We take this axiom to ensure that equality of paths behave as expected. Given the definition of a path in the formalism:

```
Reserved Notation "v ⇝ v'" (at level 80).
Reserved Notation "v ⇝ v' :> G".
Inductive path (G : LagoisGraph.type) : G → G → Type :=
  | path_empty v : v⇝v
  | path_cons v v' v'' f : v↣[f]v' → v'⇝v'' → v⇝v''
where "v ⇝ v'" := (path v v')
and   "v ⇝ v' :> G" := (@path G v v').
```

Notice that two path `f g : v ⇝v'` for some `v v' : G` might take the exact same paths but contain differing proofs of their edge relations, i.e. `f` and `g` are not equal. In this case `squash_edge` makes things right by enforcing that edge relation proofs be unique, and thus giving `f = g`. All proofs in the formalization still work out if you drop `squash_edge`, but the notion of a path that one expects becomes wonky.

The proof of the loop induction principle in the formalization is particularly scary looking and thus we cover it here:

```
Fixpoint loop_ind_aux
    (P : forall v : G, v ⇝ v → Prop)
    (bc_1 : forall v : G, P v (ε))
    (bc_2 : forall (v v' : G) g' (g : v ↣[g'] v') (h : v' ⇝ v),
        uniq h → P v (g ⋆ h))
    (ic : forall (v v': G) f1 f2 h,
        P v' h → P v (f1 ⋆ f2) → P v (f1 ⋆ h ⋆ f2))
    v (f : v ⇝ v)
    (ACC : Acc lelooplen (existT _ v f)) {struct ACC} : P v f :=
```

```
10    match loopdecomp' f with
11    | or_introl f_eq_ε => eq_rect ε _ (bc_1 v) f (esym f_eq_ε)
12    | or_intror rest =>
13        match rest with
14        | or_introl f_imm =>
15            let: ex_intro v' (
16                   ex_intro g' (
17                     ex_intro g (
18                       ex_intro h (
19                         ex_intro un_h f_imm)))) := f_imm in
20              eq_rect (g ⋆ h) _
21                  (bc_2 v v' g' g h un_h) f (esym f_imm)
22        | or_intror f_dup =>
23            let: ex_intro v' (
24                   ex_intro f1 (
25                     ex_intro g (
26                       ex_intro f2 (
27                         conj f_eq (
28                           conj glef f1f1lef))))) := f_dup in
29            let Pg :=
30                loop_ind_aux bc_1 bc_2 ic (Acc_inv ACC glef) in
31            let Pf1f2 :=
32                loop_ind_aux bc_1 bc_2 ic (Acc_inv ACC f1f1lef) in
33              eq_rect (f1 ⋆ g ⋆ f2) _
34                  (ic v v' f1 f2 g Pg Pf1f2) f (esym f_eq)
35        end
36    end.
37
38  Definition loop_ind
39      (P : forall v : G, v ⇝ v :> G → Prop)
40      (bc_1 : forall v : G, P v (ε))
41      (bc_2 : forall (v v' : G) g' (g : v -g'-> v') (h : v' ⇝ v),
42          uniq h → P v (g ⋆ h))
43      (ic : forall (v v': G) f1 f2 h,
44          P v' h → P v (f1 ⋆ f2)→ P v (f1 ⋆ h ⋆ f2))
45      v (f : v ⇝ v) : P v f :=
46    loop_ind_aux bc_1 bc_2 ic (subloopwf (existT _ v f)).
```

The function, i.e. the proof, is derived with the method prescribed by [5] and the informal proof of Theorem 2. The function is first written as a general recursive function, which is generally not allowed in Coq. Then we add the assumption that `f` is accessible through the strict ordering of loops based on their length i.e. `ACC`. In the recursive case this allows us to do structural recursion on `ACC`, which is allowed in Coq, giving us the function `loop_ind_aux`. We can get rid of the assumption `ACC` by proving that the ordering of loops based on their length is well founded, i.e., all loops are accessible through the ordering just described. This is the case, given a type A, a relation $R(x, y) :\equiv f(x) < f(y)$ with $f : A \to \mathbb{N}$ is always well founded. In this particular case, A is the type of loops in a graph and $f(x)$ is the length of loop x. With `ACC` eliminated we can finally derive the complete induction principle as seen in the function `loop_ind`.

As a final aside, the formalized proof of Theorem 3 is by loop induction instead of resorting to Proposition 3. Some of the obvious proof steps in the informal proof become quite involved when formalized, for example the fact that if $f : v \rightsquigarrow v'$ for some $v\ v' : G$ is simple then f^{-1} is also simple. On the contrary, the inductive proof takes up quite a lot of space when written out informally. This discrepancy is immaterial, insofar as we are not proving things about our proofs.

8 Related Work

The mathematical structure that is the Lagois connections is originally due to Melton et al. as presented in their seminal work [6]. In this article Melton et al. present several results that are analogous, though most not directly, to results derived from Galois connections.

The idea to use Lagois connections as a framework for secure information flow is due to Bhardwaj and Prasad as originally presented in [2]. Along with the framework, Bhardwaj and Prasad present several results from [6] that are relevant for secure information flow. Further, they develop a type system that is sound with respect to noninterference for secure information flow between two organizations. While the threat model is fairly permissive the assumptions placed on the programs are severely limiting: The two programs running at each organization are synchronized and communication can only occur outside of conditionals and loops.

In [3], Bhardwaj and Prasad extend the seminal work presented in the prequel. Firstly, the article covers how to establish and update secure connections based on the Lagois connection framework, for this purpose many results from [6] are again used. In this regard, they cover the extension from two actors to an arbitrary amount, as discussed in Section 3. Secondly, they prove that Lagois connections can be used to establish a secure connection with two organizations using the decentralized label model due to Myers [7].

In [8], Nielson et al. present an alternative to Bhardwaj- and Prasads type system. Most importantly, the type system can deal with an arbitrary number of actors/organizations out of the box. Further, actors are not restricted in the way they are connected, i.e., they do not have to be connected in a chain and can move between clusters of interconnected actors provided that they drop all information that is not already public before moving. The programs under consideration are also not limited in the fashion that those considered by Bhardwaj and Prasad are. However, the security of the system is described in terms of a condition similar to our Definition 13, it is thus unclear how strong the noninterference guarantees are.[3]

In [1], Askarov and Chong present an alternative definition of noninterference in terms of the change in knowledge of an attacker. Their definition is parametric in the attacker, and programs can be secure against some attacker but not others. In this regard they identify a collection of relevant attackers, in particular the perfect attacker that remembers everything it sees and the "i'th event only" attacker that only observes the i'th event. They show that a program that is secure against an "i'th event only" attacker for all i is also secure against the perfect attacker. They use this fact to develop enforcement mechanisms, both static and dynamic, that are secure against the perfect attacker. Askarov- and Chongs framing is quite different from ours: They are concerned with only a single program and security of input channels/streams to this program. Further, the flows that are legal can dynamically change and the attacker is only able to observe a single channel as opposed to several variables.

[3] [9], also by Nielseon et al., covers much of the same material as [8] but with a worked example. The ability for actors to move between clusters is however not covered.

9 Conclusion

As we mentioned in Section 3, the main motivation for studying arbitrary graphs of Lagois connections instead of chains of Lagois connections was to bypass connections incurring label loss. We have mitigated this problem to some degree, but it is inherently the case that the graph must be forest shaped if we want ensure security without peeking into the Lagois connections underlying the graph and in the worst case computing and checking the flow relation. Thus if we want to bypass an imprecise connection we can not determine a Lagois graph to be secure by looking at the graph structure alone, although Theorem 4 tells us that we only have to check the parts of the graph that do not have the structure of a forest. More precisely, Theorem 4 tells us that if we can securely bypass imprecise connections locally forming of secure connections, then these knots can be connected in a forest of knots that will be secure. Said otherwise, if a graph can be decomposed into a forest of secure subgraphs then it is secure.

The theory presented within this paper has several artificial constraints that are essentially imposed because they are considered sound assumptions in the "real world" of cyber security. First, we expect that networks and therefore Lagois graphs are finite because all real world distributed systems have a finite number of agents. In this regard, our theory actually extends to arbitrary graphs as long as identification of vertices is decidable. Secondly, we assume that each point in the graph has an associated finite inhabited lattice, because these represent security labels and how they behave. In this regard, our theory is actually generalizable to arbitrary partial orders. Thirdly, as noted in Section 3, our development does not depend on **LC3** and **LC4**. This is in and of itself an interesting result because as Melton et al. states: "without them (referring to **LC3** and **LC4**) the resulting concept appears to be too weak to be of general interest". It appears that we have added enough structure to make poset systems satisfying only **LC1** and **LC2** interesting. I, the author, is conflicted about the inclusion of **LC3** and **LC4** for the purposes of studying secure connections. An extended discussion of the inclusion/exclusion of **LC3** and **LC4** is included in Appendix A.

As mentioned in Section 9, we do not bridge the gap between our framework and a soundness result with respect to the operational model presented herein, however we conjecture that it is possible. In terms of an enforcement mechanism we expect that the type system presented by Nielson et al. [8, 9] could be adapted to our operational model. As mentioned in Section 8, Nielson et al. do not provide a soundness results but we conjecture that a type system similar to theirs applied to our setting could be proven sound with respect to progress-insensitive noninterference when the underlying graph is secure. One foreseeable challenge with porting over the type system presented by Nielson et al. is the labeling of communication channels. Nielson et al. assume that all agents agree on security labels, their structure, and the labeling of channels. In our framework agents only partially agree on labels, their structure, and thus the labeling of channels. At the moment it is not clear to us how one would deal with this discrepancy.

Two concepts that our framework does not account for is dynamism of agents and dynamism of policies. By dynamism of agents we mean: The ability of agents to move about in the network and in the process dropping old connections and establishing new ones. For example [9] has dynamism of agents: Agents can move between clusters of agents via a primitive `relocate`(l) command, that relocates agent v to cluster l when executed at agent v. By dynamism of policies we mean: The ability of policies to change during runtime. For example [1] has dynamism of policies: A programs can update its information flow policy via a `setPolicy`($\sqsubseteq$) primitive for

some reflexive relation $\sqsubseteq$. We propose that such dynamism can be modeled by functions of type

$$G \to_{\mathcal{G}} G' :\equiv \prod_{(v:G)} \sum_{(v:G')} L(v) \to L(v'). \tag{22}$$

For $F : G \to_{\mathcal{G}} G'$ and $v : G$ we would then interpret $F(v)$ as v moving to or becoming $\mathsf{pr}_1(F(v))$ and v's lattice transforming as specified by $\mathsf{pr}_2(F(v))$. More specifically $(\mathsf{pr}_2(F(v)))(p) = p'$ would denote label p becoming label p'. We provide a cursory discussion of the notion in Appendix B.

The notion of noninterference that we develop in Section 6 assumes an attacker A residing at v that is trying to learn about the initial memory at v beyond what he is allowed by some security labeling Λ and the local policy $L(v)$ by continually observing the state of the variables he is allowed to. Additionally, the attacker knows the code running on the entire distributed network. One could obtain different notions of noninterference by tweaking the assumptions placed on the attacker. For example, one could imagine an Attacker A_v residing at v that is trying to learn about the initial value of variables in the memory of v' for $v' \neq v$. Or, an attacker A_c could be observing a channel c instead of a set of variables and so on.

A On the inclusion/exclusion of LC3 and LC4

As mentioned in Section 3 and Section 9, **LC3** and **LC4** are never appealed to in a proof step within this development. This is somewhat natural, as none of our results deal with precision of any kind. Thus for the sake of security alone, we in the best case consider **LC3** and **LC4** nice to have and in the worse case needlessly restrictive. Consider the original justification for the inclusion of **LC3** and **LC4** as provided by Bhardwaj and Prasad [2, 3]: To enforce precision and convergence of the connection, i.e. **PC1**, **PC2** and **CC1**, **CC2** respectively. Consider now that for two very large and complex lattices P and Q with top elements $\top_P : P$ and $\top_Q : Q$ that the following Lagois connection is perfectly valid:

$$(P, f, g, Q) \text{ where } f(p) :\equiv \top_Q \text{ and } g(q) :\equiv \top_P \tag{23}$$

Precision, in the sense of the size of $(g \circ f)[P]$ (or $(f \circ g)[Q]$), is completely lost, and convergence is necessarily achieved but only because we immediately overinflate labels to the topmost level. The problem arises from the fact that in a Lagois connection (P, f, g, Q) it is the case that f and g uniquely determine each other (Theorem 3.9 in [6]), a property that is otherwise nice. This problem also occurs in the opposite direction: For a Lagois connection (P, f, g, Q) if f is precise then g is forced to assume an equivalent level of precision as specified by **PC2** (vice versa for **PC1** if g is precise). If we for two organizations v and v' with a Lagois connection $(L(v), f, g, L(v'))$ view f as v's understanding of $L(v')$ in terms of $L(v)$ and vice versa for g then **LC3** and **LC4** enforces that v and v' agree in their understanding. Or from another perspective one organization is forced to assume the worldview of the other.

B Lagois Graph Transformations

Like [8, 9] we wish to reason about agents relocating, but within the framework that we have developed. Further, similar to [1], we would like to reason about actors updating their policies. Let $\mathcal{G}$ denote the type of Lagois graphs. We model such changes as an inhabitant of the instantiation of the type family $- \to_{\mathcal{G}} - : \mathcal{G} \to \mathcal{G} \to \mathcal{U}$ defined as

$$G \to_{\mathcal{G}} G' :\equiv \prod_{(v:G)} \sum_{(v:G')} L(v) \to L(v'). \tag{24}$$

For graphs G and G' we see $F : G \to_{\mathcal{G}} G'$ as a description of an atomic transformation of the graph G into G' in the sense that $\mathsf{pr}_1(F(v)) = v'$ expresses that v in G becomes v' in G'. Further, $\mathsf{pr}_2(F(v))(p) = p'$ expresses that security level p in $L(v)$ becomes security level p' in $L(v')$.

We will now develop a notion of security of transformations that in some sense, that is to be specified, falls short of the mark. This notion of security will be used as a stepping stone towards a more satisfactory definition. For some graphs G and G' and a transformation $F : G \to_{\mathcal{G}} G'$ we define a new flow relation with judgments $(v,p) \rightrightarrows' (v',q)$ similar to the one given in Definition 12:

$$\frac{p \leq q}{(v,p) \rightrightarrows' (v,q)}\ (\mathsf{flowle}), \quad \frac{}{(v,p) \rightrightarrows' (\mathsf{pr}_1(F(v)), \mathsf{pr}_2(F(v))(p))}\ (\mathsf{flowts})$$

$$\frac{v \overset{f}{\rightarrowtail} v'}{(v,p) \rightrightarrows' (v', f(p))}\ (\mathsf{flowlc}), \quad \frac{(v,p) \rightrightarrows' (v',r) \qquad (v',r) \rightrightarrows' (v'',q)}{(v,p) \rightrightarrows' (v'',q)}\ (\mathsf{flowtran}).$$

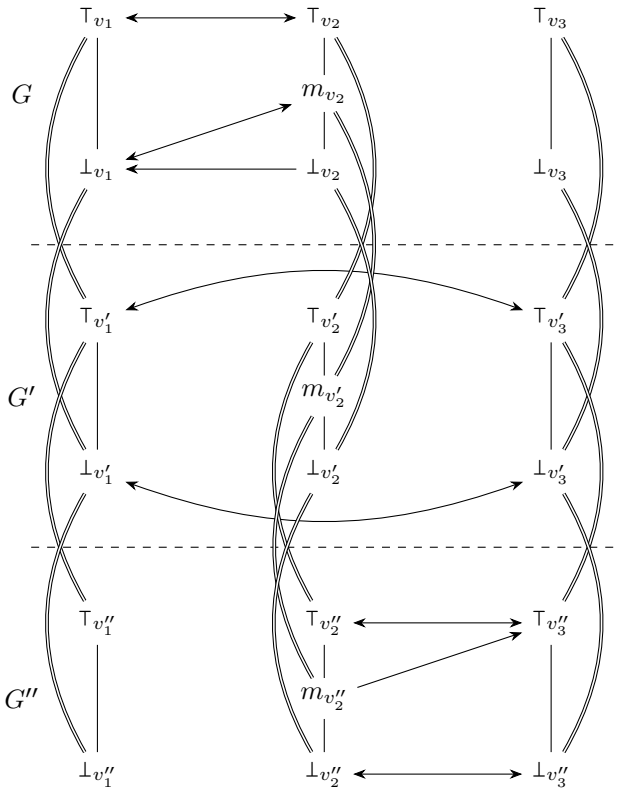
As before we drop v and v' from $(v,p) \rightrightarrows' (v',q)$ granting judgments $p \rightrightarrows' q$ when this causes no confusion. Essentially we allow flows to occur within G and G' as before, but also via the transformation F by the rule flowts. We can now define a (somewhat) suitable notion of security.

Definition 21 (Atomically secure transformation)**.** For two Lagois graphs G and G' a transformation $F : G \to_{\mathcal{G}} G'$ is *atomically secure* whenever G and G' are secure with respect to $\rightrightarrows'$ and

$$\prod_{(v:G)} \prod_{(p:L(v))} \prod_{(q:L(\mathsf{pr}_1(F(v))))} p \rightrightarrows q \to \mathsf{pr}_2(F(v))(p) \leq q. \tag{25}$$

Essentially because v in G is regarded as $\mathsf{pr}_1(F(v)))$ in G' and because $p : L(v)$ is regarded as $\mathsf{pr}_2(F(v))(p)$ we have to check that that a flow from $p : L(v)$ to $q : L(\mathsf{pr}_1(F(v)))$ is secure modulo F.

We will now show a critical flaw in this notion of security. Consider two Lagois graph transformations $F : G \to_{\mathcal{G}} G'$ and $H : G' \to_{\mathcal{G}} G''$ for $G\ G'\ G'' : \mathcal{G}$ as seen in the figure below.



The Lagois graphs G, G' and G'' are depicted as in Section 1 and separated by the dashed line. Here F and G are denoted by the double lines, the direction of the transformation is from top to bottom. In this situation F and H are both atomically secure, but it is also the case that $H \circ F : G \to_{\mathcal{G}} G''$ atomically secure if we define

$$H \circ F :\equiv \lambda v.\big((H_1 \circ F_1)(v), H(F_1(v))_2 \circ F(v)_2\big) \tag{26}$$

where $F_1 :\equiv \lambda v.\mathsf{pr}_1(F(v))$ and $F(v)_2 :\equiv \mathsf{pr}_2(F(v))$. But intuitively the composition should not be secure. See that information can flow from m_{v_2} to $\bot_{v_2''}$, which is insecure in some sense because m_{v_2} should be regarded as $m_{v_2''}$ according to the transformation $H \circ F$. Essentially normal function composition obscures the information flows that occur while G' is the current state of the system. This is why we refer to Definition 21 as atomically secure, i.e., because for residing in an intermediate state may not be secure. Instead we want to inquire about a sequence of atomic transformations $F_i : G_i \to_{\mathcal{G}} G_{i+1}$ for $i = 1 \dots n$. We thus update the rule flowts to account for this change in the following manner:

$$\frac{}{(v,p) \rightrightarrows' (\mathsf{pr}_1(F_i(v)), \mathsf{pr}_2(F_i(v))(p))} \quad (\mathsf{flowts}')$$

And define security as:

Definition 22 (Secure sequence of transformations)**.** A sequence of transformations $F_k : G_k \to_{\mathcal{G}} G_{k+1}$ for $k = 1 \dots n$ is said to be secure whenever for all $i = 1 \dots n$ and $j = i \dots n$ one has

$$\prod_{(v:G_i)} \prod_{(p:L(v))} \prod_{(q:L(\mathsf{pr}_1((F_j \circ \cdots \circ F_i)(v))))} p \rightrightarrows' q \to \mathsf{pr}_2((F_j \circ \cdots \circ F_i)(v))(p) \leq q. \tag{27}$$

References

[1] Aslan Askarov and Stephen Chong. "Learning is change in knowledge: Knowledge-based security for dynamic policies". In: *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE. 2012, pp. 308–322.

[2] Chandrika Bhardwaj and Sanjiva Prasad. "Only connect, securely". In: *Formal Techniques for Distributed Objects, Components, and Systems: 39th IFIP WG 6.1 International Conference, FORTE 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings 39*. Springer. 2019, pp. 75–92.

[3] Chandrika Bhardwaj and Sanjiva Prasad. "Secure information flow connections". In: *Journal of Logical and Algebraic Methods in Programming* 127 (2022), p. 100761.

[4] Dorothy E Denning. "A lattice model of secure information flow". In: *Communications of the ACM* 19.5 (1976), pp. 236–243.

[5] Xavier Leroy. "Well-founded recursion done right". In: *CoqPL 2024: The Tenth International Workshop on Coq for Programming Languages*. 2024.

[6] Austin Melton, Bernd SW Schröder, and George E Strecker. "Lagois connections—a counterpart to Galois connections". In: *Theoretical Computer Science* 136.1 (1994), pp. 79–107.

[7] Andrew C Myers and Barbara Liskov. "A decentralized model for information flow control". In: *ACM SIGOPS Operating Systems Review* 31.5 (1997), pp. 129–142.

[8] Flemming Nielson, René Rydhof Hansen, and Hanne Riis Nielson. "Adaptive security policies". In: *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part II 9*. Springer. 2020, pp. 280–294.

[9] Flemming Nielson, René Rydhof Hansen, and Hanne Riis Nielson. "Benign Interaction of Security Domains". In: *Protocols, Strands, and Logic: Essays Dedicated to Joshua Guttman on the Occasion of his 66.66 th Birthday*. Springer. 2021, pp. 312–331.

[10] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.