Formalisation and Extension of Lagois Connections for Secure Information Flow

Casper Ståhl
Aalborg University, Denmark

Levs Gondelman
Aalborg University CPH, Copenhagen, Denmark

René Rydhof Hansen
Aalborg University, Denmark

Danny Bøgsted Poulsen
Aalborg University, Denmark

Abstract

Lagois connections have been proposed as a way to formulate and formalise a secure information flow policy between two organisations each with their own information flow policies that must also be enforced. By composing Lagois connections in a chain, it is possible to compose the information flow policies of several organisations. However, chains are not always suitable. For example, if one of the "links" in the chain enforces a coarser policy than the surrounding links, this corresponds to taking low security information as input and reclassifying it as high security, preventing further communication of (formerly low) information at the low security level. This is an instance of the well-known problem of "label creep".

In this paper we extend the compositionality results of Lagois connections to *forests*, and graphs that "behave" like forests, of Lagois connections. In addition to the extra flexibility gained in the communication topology, it also solves the label creep problem: Intuitively, to bypass the problematic intermediary, we can verify that the subgraph bypassing the problematic intermediary is located in a forest of secure sub-graphs. We show that this is sound with respect to noninterference, by proving that if information in a program flows according to a secure graph then an attacker can never observe a policy violation in the system.

As a further contribution of this paper, all of the above has been formalised in the Rocq proof assistant, including a novel formalisation of a substantial part of the literature of Lagois connections, providing a strong foundation for future work and implementations.

2012 ACM Subject Classification Security and privacy → Formal methods and theory of security; Theory of computation → Type theory

Keywords and phrases Lagois connection, information flow, security

Digital Object Identifier 10.4230/LIPIcs.TYPES.2025.6

Supplementary Material

Software (Source code): `https://github.com/CasperStaahl/LC4S-Rocq` [11]
archived at `swh:1:dir:828cd84332eae467073735e4aee31198815adf58`

1 Introduction

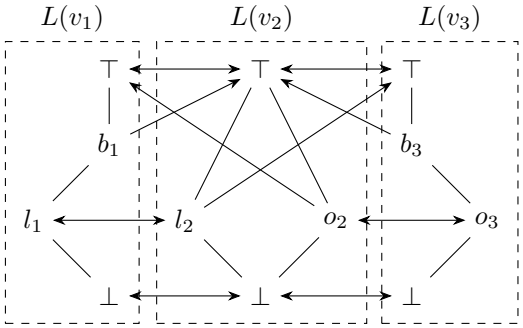
The distributed and interconnected nature of modern computing remains a challenge for the traditional approach to information flow security which often relies on a central notion or policy of how information ought to flow[1]. In practice, organisations seldom have the resources or access to verify such a property for the totality of an entire system. Therefore, there is a need for compositional methods and theories of security, i.e., how does one combine secure

[1] This paper builds on and expands the work of the first author's MSc thesis [10].

subsystems such that security is consequent in the aggregate system. One such theory is that of the Lagois connection for secure information flow, as originally presented by Bhardwaj and Prasad [2, 3]. But this theory is not without its own challenges, one of which we seek to address in this paper.

To illustrate, imagine a scenario in which two systems v_1 and v_2 wish to communicate securely with regards to information flow. Within systems v_1 and v_2, the flow of information is governed by the respective security lattices $L(v_1)$ and $L(v_2)$ [4]. Now how do we model and enforce the flow of information when communication between v_1 and v_2 has to be taken into account? As shown by Bhardwaj and Prasad [2, 3] a *Lagois connection* between the two security lattices, $(L(v_1), f, g, L(v_2))$ provides a natural foundation for formalising and studying information flow security in such a context. Most importantly the Lagois connection describes communication that is secure in the sense that an organisation can never observe a violation of its own policy. For example in the case of v_1 one has $p \leq g(f(p))$ for all $p \in L(v_1)$ (analogous for v_2). Thus, the functions f and g capture the flow of information between v_1 and v_2, while $L(v_1)$ and $L(v_2)$ continue to represent the internal flow of information within each organization.

Generalising the above to n systems $v_1, v_2, \ldots, v_n$ who wish to communicate securely, Bhardwaj and Prasad have shown that secure Lagois connections can be *chained* together [2, 3]. To illustrate, consider three systems v_1, v_2 and v_3 with security lattices and Lagois connections as depicted below:



Intuitively we would like for v_1 to be able to share information with label b with members of v_3 with clearance b, and vice versa. But this is not possible because $L(v_2)$ acts as a lossy intermediate with respects to labels, swapping the order of the chain is no help either as $L(v_1)$ or $L(v_3)$ will become the lossy intermediate. This paper seeks to expand the use of Lagois connections exactly to overcome this limitation.

In particular we consider *graphs* of lattices connected by Lagois connections such that lossy intermediates can be bypassed. Not all such graphs are secure and we therefore identify graphs that are secure and ways in which secure graphs may be constructed. This model seems especially well suited for describing secure information flow in distributed systems and thus we contribute to consolidating the theory for such a setting by giving a semantic condition for when the flow of information in a distributed system respects a graph, and by showing that if the flow of information in a distributed system respects a secure graph then the system observes a suitable notion of noninterference. The preceding contributions have all been formalized in the Rocq proof assistant, providing a rigorous foundation for future work. In summary, the contributions of this paper are:

- A Rocq formalisation of Lagois connections and substantial parts of Melton et al. [6],
- A Rocq formalisation of the information flow framework of Bhardwaj and Prasad [2],
- An extension of the composability of Lagois-based information flow policies to more complex topologies, e.g., virtual Lagois forests, also formalised in Rocq.

2 Lagois Connections

In this section we introduce Lagois connection, but first re-visit and define several concepts from order theory, in particular partial orders, monotonic functions and poset systems, that underlie Lagois connections.

▶ **Definition 1** (Partial order). *A partial order is a tuple $(A, \leq)$ where A is a set and $\leq$ is a relation on A such that*

- $x \leq x$ *for all* $x \in A$ *($\leq$ is reflexive),*
- $x \leq y \implies y \leq z \implies x \leq z$ *for all* $x, y, z \in A$ *($\leq$ is transitive), and*
- $x \leq y \implies y \leq x \implies x = y$ *for all* $x, y \in A$ *($\leq$ is antisymmetric).*

For a partial order $P = (A, \leq)$ we will omit explicitly mentioning A or P when the context is clear. Further, we will overload the symbol $\leq$ when multiple orders are present and the context is clear.

▶ **Definition 2** (Monotone 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 3** (Poset system). *A* poset system *is a tuple (P, f, g, Q) of two partial orders P and Q and monotonic functions $f : P \to Q$ and $g : Q \to P$.*

Security with respect to information flow is typically formulated as *information flow policies* formalised in terms of (security-)lattices, as prescribed by Denning [4].

▶ **Definition 4** (Lattice). *A* lattice *is tuple $(A, \sqcup, \sqcap)$ where A is a partial order, $- \sqcup - : A \to A \to A$ is a join operator and $- \sqcap - : A \to A \to A$ is a meet operator such that*

- $x \sqcup y \leq z \iff x \leq z \wedge y \leq z$ *for all* $x, y, z \in A$*, and*
- $x \leq y \sqcap z \iff x \leq y \wedge x \leq z$ *for all* $x, y, z \in A$*.*

For information flow purposes, the lattices are assumed to be finite and inhabited. Taking Denning's perspective, given a lattice A information is allowed to flow from an object labeled $x \in A$ to an object labeled $y \in A$ if $x \leq y$, i.e., information is only allowed to flow from lower levels to higher levels. This object with label x could for example be a document, and the other object labeled y could be an employee. Typically the objects are program variables in an imperative programming language.

With the preliminaries covered we now introduce the Lagois connection formally.

▶ **Definition 5** (Lagois connection [6]). *A poset system (P, f, g, Q) is called a Lagois connection whenever:*

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

For two organisations v and v', we can view a Lagois connection $(L(v), f, g, L(v'))$ as a secure, reciprocal and converging 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 securely translate between $L(v)$ and $L(v')$ [2, 3].[2] Equivalently we can see a Lagois connection $(L(v), f, g, L(v'))$ as a policy for

[2] By "reciprocal" we are referring to what is called "precision" in [2, 3].

cross communication between v and v'. In particular, information in v labeled $p \in L(v)$ is allowed to flow as specified by the lattice $L(v)$ but also from p to $q \in L(v')$ whenever $f(p) \leq q$ (analogous for g), and via the transitive closure of the two ordering relations. Properties **LC1** and **LC2** ensure that the connection is secure:

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

Intuitively, translating a label back and forth will not violate the local policy nor will sending a piece of information back and forth ever accidentally declassify it.

When labels are translated back and forth between systems, properties **LC3** and **LC4** ensure *precision* and *convergence*, i.e., that the translation does not give rise to *label creep* and unnecessary over-approximation. This is mainly relevant for enforcement of security, e.g., through type systems or static analysis, but not for the purposes of this paper. We therefore only include them here for completeness and refer to [10, Appendix A] for a more detailed discussion. The properties below, following from **LC3** and **LC4**, ensure precision of the Lagois connection:

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

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:

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

▶ **Example 6.** Figure 1 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.

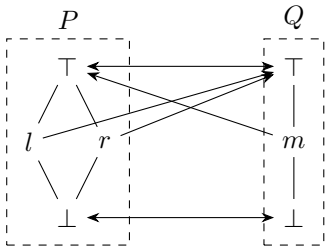


Figure 1 The Lagois connection (P, f, g, Q).

Communication rarely occurs between two system in isolation, therefore it makes sense to extend the framework of Lagois connection for secure information flow to an arbitrary (finite) number of systems. Bhardwaj and Prasad propose composition of Lagois connections as a method for such extensions [3] and define composition of Lagois connections as follows:

$$(P, f, g, Q) \circ (Q, \hat{f}, \hat{g}, R) = (P, \hat{f} \circ f, \hat{g} \circ g, R) \tag{1}$$

To formalise and show that security is preserved by composing Lagois connections, Bhardwaj and Prasad present the following theorem by Melton et al. [6]:

▶ **Theorem* 7** (Theorem 11 in [6]). *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 a Lagois connection iff* $\hat{g} \circ \hat{f} \circ f[P] \subseteq f[P]$ *and* $f \circ g \circ \hat{g}[R] \subseteq \hat{g}[R]$.

Slightly disappointing is the fact that not all Lagois connections compose to form Lagois connections. However, the composition of secure connections (as in satisfying **LC1** and **LC2**) is always secure:

▶ **Proposition* 8.** *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. ◀

More importantly as pointed out in section 1, this way of chaining Lagois connections to connect multiple systems securely also has an unintended side effect, as each translation of a label may introduce imprecision, e.g., by merging labels. Essentially we want to bypass the intermediate lattice when possible. This motivates us to study graphs where each vertex has an associated lattice and when two vertices are connected in the graph their underlying lattices are connected by some Lagois connection. Continuing the example from section 1, Figure 2 shows such a graph. Here $L(v_1), L(v_2)$ and $L(v_3)$ can communicate directly with one another, without introducing imprecision in the label information potentially caused by an intermediate. Is this secure[3]? In this case the pattern of communication is secure, but generally this is not the case[4].

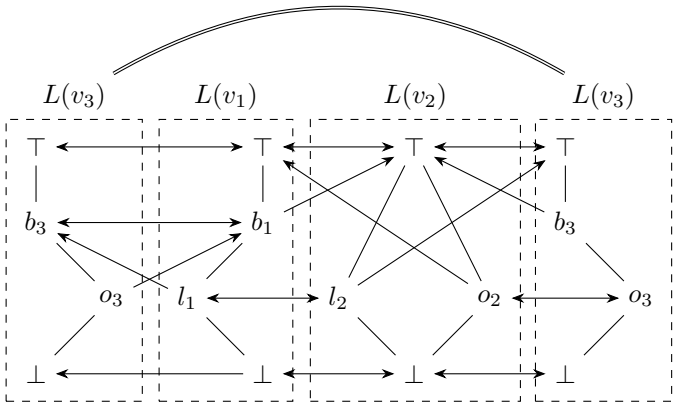


Figure 2 A Lagois graph.

3 Lagois Graphs and Security

In this section, we define the notion of *Lagois graph* to formalise the communication between systems with different security policies. We further define and investigate what it means for communication to be secure in such a system defined by a Lagois graph. In particular, we develop several notions of security capturing different intentions and intuitions about secure communication, but conclude by showing they are all logically equivalent. We start by reviewing and formalising essential concepts from graph theory.

[3] See Definition 15 for a definition of security.
[4] See Figure 3 for a graph that is not secure.

▶ **Definition 9** (Graph). *A graph is a tuple $G = (V, E)$ where V is a finite inhabited set of vertices and E is a edge relation on V such that:*

- *$(v, v) \notin E$ for all $v \in V$ (E is irreflexive),*
- *if $(v, v') \in E$ then $(v', v) \in E$ for all $v, v' \in G$ (E is symmetric).*

For convenience, the basic graphs we work with are undirected (symmetric) graphs with no self-loops (irreflexive), as defined above. For a graph $G = (V, E)$ we let G denote both the graph and V when this causes no confusion.

▶ **Definition 10** (Path). *For a graph $G = (V, E)$ a* path *is either the empty path, denoted ϵ, or a sequence of edges $(v_1, v_2), (v_2, v_3) \ldots, (v_n, v_{n+1}) \in E^*$.*

We will mainly be working with two operations on paths, reversal $(-)^{-1}$ and concatenation $- \star -$. For a path p we let $|p|$ denotes its length. Certain paths turn out to be useful, namely simple paths and loops.

▶ **Definition 11** (Simple path). *A* simple path *is either the empty path ϵ or a path (v_1, v_2), $(v_2, v_3), \ldots, (v_n, v_{n+1})$ where each vertex is distinct $i \neq j \implies v_i \neq v_j$.*

▶ **Definition 12** (Loop). *A path $(v_1, v_2), (v_2, v_3) \ldots, (v_n, v_{n+1})$ is a* loop *if $v_1 = v_{n+1}$.*

Note that this definition allows for nested loops. Furthermore, the only simple loop from v to v is the empty path ϵ, as otherwise v would appear twice: at the beginning of the sequence and at the end.

We can now formally define the notion of *Lagois graph* as a graph of Lagois connections.

▶ **Definition 13** (Lagois graph). *A* Lagois graph *is a tuple (G, L, F) where*

- *$G = (V, E)$ is a graph,*
- *L is a function associating each vertex $v \in V$ to a finite inhabited lattice $L(v)$,*
- *F is a function associating each edge $e = (v, v') \in E$ to a monotone function $F(e) : L(v) \to L(v')$, and*
- *for each pair of symmetric edges $e = (v, v')$ and $e' = (v', v)$ in E we have that $(L(v), F(e), F(e'), L(v'))$ is a Lagois connection.*

We will denote a Lagois graph (G, L, F) by G when it does not cause confusion. For some Lagois graph $G = (V, E)$ we can view each $v \in V$ as a system, actor, organisation, etc. who wishes to communicate securely with other $v' \in V$. Each system $v \in V$ has an associated security lattice $L(v)$ that specifies its local policy. When two systems $v, v' \in V$ are connected in the graph, i.e., we have $e = (v, v') \in E$ and consequently $e' = (v', v) \in E$, we assume that a policy for secure information exchange has been negotiated forming a Lagois connection $(L(v), F(e), F(e'), L(v'))$. Information is then allowed to flow via a flow relation that is induced by the structure of the graph and its underlying lattices and Lagois connections.

▶ **Definition 14** (Flow relation). *Each Lagois graph $G = (V, E)$ induces a flow relation for $v, v' \in V$ and $p \in L(v), q \in L(v')$ with judgments $(v, p) \rightrightarrows (v', q)$ which is inductively defined as follows:*

$$\frac{p \leq q}{(v, p) \rightrightarrows (v, q)} \quad \textsf{(flowle)}, \qquad \frac{e = (v, v') \in E}{(v, p) \rightrightarrows (v', F(e)(p))} \quad \textsf{(flowlc)}$$

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

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.

Figure 3 displays a Lagois graph $G = (V, E)$. The graph contains three elements $v_1, v_2, v_3 \in V$ 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 arrows. Notice that flowlc $(v_1, m) \rightrightarrows (v_2, \bot) \rightrightarrows (v_3, \bot) \rightrightarrows (v_1, \bot)$, i.e., $(v_1, m) \rightrightarrows (v_1, \bot)$ which *violates* the local policy of v_1 because $m \not\leq \bot$. Worryingly, it is intuitive to see that not all Lagois graphs are secure.

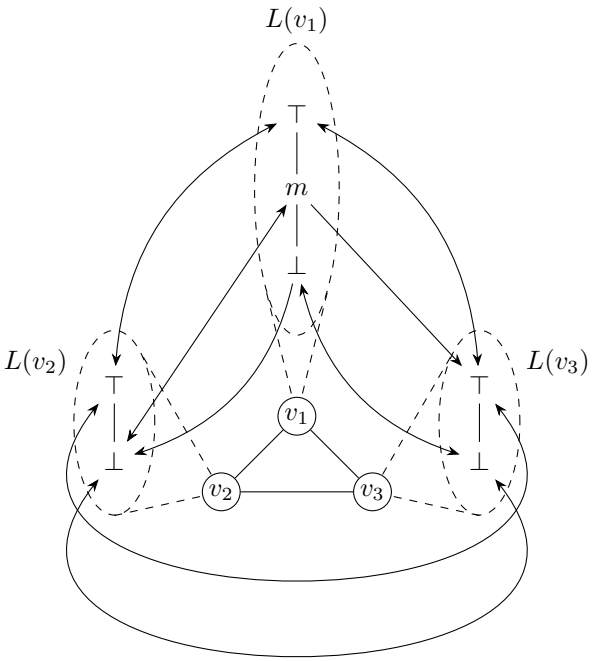


Figure 3 An insecure Lagois graph.

This example gives rise to our first notion of security, similar to the one presented in [8, 9].

▶ **Definition 15** (Flow secure). *A graph G is said to be* flow secure *if for all $v \in G$ and $p, p' \in L(v)$ it is the case that $(v, p) \rightrightarrows (v, p')$ implies $p \leq p'$.*

Essentially, a graph G is flow secure if for all vertices $v \in G$ all flows that start and end at v respects v's local policy as specified by $L(v)$. This setup is somewhat similar to that of [8] and we conjecture that the cubic time algorithm (in the number of communicating nodes) presented there for verifying flow security of communication between programs, can be adapted to determine if a graph is flow secure.

We will now develop a second definition of security which is somewhat similar to **SC1** (or **SC2**). This definition 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 $f = (v_1, v_2), (v_2, v_3), \ldots, (v_n, v_{n+1})$ in a Lagois graph can similar to edges be treated as a function $\hat{f} : L(v_1) \to L(v_{n+1})$. $\hat{f}$ is intuitively computed as $\hat{f} = F(e_n) \circ \cdots \circ F(e_2) \circ F(e_1)$ for $f = e_1, e_2, \ldots, e_n$. Formally it is computed by induction on the sequence, with the base case $\hat{\epsilon} = \mathsf{id}$. We can now state the second notion of security.

▶ **Definition 16** (Loop secure). *Loop security is defined in three levels:*

- *A loop* $f = (v, \_), \ldots, (\_, v)$ *is* loop secure *whenever* $p \leq \hat{f}(p)$ *for all* $p \in L(v)$.
- *A vertex* $v \in G$ *is* loop secure *if for all loops* $f = (v, \_), \ldots, (\_, v)$, f *is loop secure.*
- *A Lagois graph* G *is* loop secure *if for all* $v \in G$, 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 show.

▶ **Lemma 17.** *If* $p \rightrightarrows q$ *for some* $p \in L(v)$, $q \in L(v')$, *then there exists a path* $f = (v, \_), \ldots, (\_, v')$ *such that* $\hat{f}(p) \leq q$.

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

▶ **Proposition 18.** *A graph is flow secure iff it is loop secure.*

Proof.

$\rightarrow$ Fix $v \in G$ a path $f = (v, \_) \ldots, (\_, v)$ and $p \in L(v)$. Because there is a loop f there is a flow $p \rightrightarrows \hat{f}(p)$. Thus by our assumption that flow is secure we have $p \leq \hat{f}(p)$.

$\leftarrow$ Fix $v \in G$ and $p, p' \in L(v)$ and assume that $p \rightrightarrows p'$. By Lemma 17 there exists a path $f = (v, \_), \ldots, (\_, v)$ such that $\hat{f}(p) \leq p'$. Thus we have $p \leq \hat{f}(p) \leq p'$ by our assumption that all loops are secure. ◀

For convenience, 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.

We can now derive the final notion of security, one that is very close to **SC1** (or **SC2**). This definition of security will again turn out to be logically equivalent to our other definitions of security.

▶ **Definition 19** (Simply secure). *A Lagois graph* G *is* simply secure *whenever for all pairs of vertices* $v, v' \in G$ *and for all simple paths* $f = (v, \_), \ldots, (\_, v')$ *and* $g = (v', \_), \ldots, (\_, v)$ *it is the case that* $f \star g$ *is secure.*

This notion might seem superfluous, especially given the results below showing that security and simple security are logically equivalent, but in the next section we will use it to prove that certain graphs are always secure without having to resort to strong induction in the future.

▶ **Lemma 20.** *For all paths* f *in a Lagois graph* G, $\hat{f}$ *is always monotone.*

Proof. By induction on the length of f. ◀

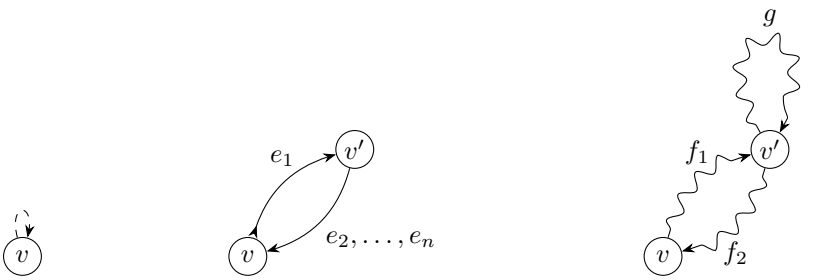
▶ **Proposition 21.** *A graph is simply secure iff it is secure.*

Proof.

$\Longrightarrow$ A secure graph is simply secure because all of its loops are secure.

$\Longleftarrow$ Fix a loop $f = e_1, e_2, \ldots, e_n$. We proceed by strong induction over the length of f, Figure 4 depicts all three cases.

- $|f| = 0$ The empty path ϵ is trivially secure.
- $|f| > 0$, $e_2, \ldots, e_n$ **is simple** f is secure because the path e_1 is simple, $e_2, \ldots, e_n$ is simple and because the graph is simply secure.
- $|f| > 0$, $e_2, \ldots, e_n$ **is not simple** We have $f = f_1 \star g \star f_2$ for some f_1, g, f_2 such that g and $f_1 \star f_2$ are loops strictly shorter than f. By strong induction we can assume g is secure, from which we can derive $\hat{f}_1(p) \leq \widehat{(f_1 \star g)}(p)$. By strong induction we can assume $f_1 \star f_2$ is secure, and then by monotonicity of paths we have $p \leq \widehat{(f_1 \star f_2)}(p) \leq \widehat{(f_1 \star g \star f_2)}(p)$. ◀



(a) $|f| = 0$. **(b)** $|f| > 0$, $e_2, \ldots, e_n$ is simple. **(c)** $|f| > 0$, $e_2, \ldots, e_n$ is not simple.

Figure 4 The three different cases in the proof of Proposition 21. The empty paths ϵ are depicted with a little dashed arrow, simple paths are depicted with straight arrow, paths that consists of a single edge are depicted as a straight arrow with a tail, and general paths are depicted with a squiggly arrow.

As shown above, our notion of Lagois graph addresses the problem of label information loss that may occur when communicating through an intermediary with a coarse(er) flow policy, by allowing more general communication patterns than chains. However, this also removes the "automatic" guarantee of security that follows from simple chaining. In the following section, we adress this problem and identify communication patterns that can restore the "automatic" security guarantee as well as provide tools for analysing and arguing about security of yet other patterns.

4 Secure Graphs

In this section we identify a class of secure Lagois graphs that expand and generalise over the basic communication chains. We further prove results showing how Lagois graphs can be built in a way that guarantees security. For this purpose, we will build on the intuition provided by the following lemma.

▶ **Lemma 22.** *$f \star f^{-1}$ is secure for all paths f.*

Proof. By induction on the length of f. ◀

Lemma 22 intuitively says that flow of information is secure as long as it returns along the same path it came from. It will be fruitful to explore less strict versions of this principle. However, an object that immediately embodies this principle is that of a forest, thus we reiterate the definition here.

▶ **Definition 23** (Forest). *A graph G is called a forest if all simple paths between all pairs of vertices are unique.*

The first subset of graphs that turns out to be secure is the one containing only virtual Lagois forests.

▶ **Definition 24** (Virtual Lagois forest). *A Lagois graph G is a* virtual forest *whenever for all vertices $v, v' \in G$ it is the case that for all simple paths $f = (v, \_), \ldots, (\_, v')$ and $g = (v, \_), \ldots, (\_, v')$ one has $\hat{f} = \hat{g}$.*

Essentially a virtual Lagois forest is a Lagois graph in which for all $v, v' \in G$, all simple paths from v to v' have the same behavior when seen as functions. Said otherwise, if you go from v to v' via a simple path it does not matter which simple path you take in regards

to label translation. Practically speaking, this would correspond to a network in which information with the same initial label would always receive the same label after arriving at some other organization assuming it took a direct path.

▶ **Theorem 25.** *A virtual Lagois forest is secure.*

Proof. Fix a graph G, by Proposition 21 it is enough to consider all pairs of simple paths $f = (v, \_), \ldots, (\_, v')$, $g = (v', \_), \ldots, (\_, v)$ for all pairs of points $v, v' \in G$. $f \star f^{-1}$ is secure by Lemma 22 and therefore $f \star g$ is also secure because $\widehat{f^{-1}} = \hat{g}$ by the fact that G is a virtual forest. ◀

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

▶ **Corollary 26.** *A Lagois forest is secure.*

Proof. A virtual Lagois forest are secure by Theorem 25 and a Lagois forest is a virtual Lagois forest because $f = g$ implies $\hat{f} = \hat{g}$. ◀

We can now effectively prove a stronger version of Proposition 8 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 shown. Further, Corollary 26 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.

Although (virtual) forests are secure, we have arrived at the same problem that we set out to solve in the first place: A lossy intermediate might still be present. In the case of a forest the lossy intermediate cannot be bypassed as there is only one direct path, and any indirection will intuitively only cause further loss of label information. In the case of the virtual forest it does not matter which path information takes as they all have the same effect: if one path from v to v' incurs some unwanted loss of label information then all paths from v to v' do.

Again employing the intuition from Lemma 22, the idea is to securely bypass intermediates locally in subgraphs and then stitch together these secure graphs in such a manner that information can only return to the subgraph the way it came. This notion of stitching graphs together is depicted in Figure 5, and we make it precise in the following manner.

▶ **Definition 27** (Secure graph composition)**.** *We define the* secure composition *of disjoint Lagois graphs G and G' with respect to vertices $\bar{v} \in G$ and $\bar{v}' \in G'$ and Lagois connection $(L(\bar{v}), f, g, L(\bar{v}'))$ as*

$$G \overset{f}{\bar{v}} \rightleftharpoons \overset{\bar{v}'}{g} G' = ((V, E), L, F) \overset{f}{\bar{v}} \rightleftharpoons \overset{\bar{v}'}{g} ((V', E'), L', F') \tag{2}$$

$$:= ((V'', E''), L'', F''), \tag{3}$$

where $V'' = V \cup V'$, $E'' = E \cup E' \cup \{(\bar{v}, \bar{v}'), (\bar{v}', \bar{v})\}$ and

$$L''(v) = \begin{cases} L(v) & \text{if } v \in V \\ L'(v) & \text{if } v \in V', \end{cases} \qquad F''(e) = \begin{cases} F(e) & \text{if } e \in E \\ F'(e) & \text{if } e \in E' \\ f & \text{if } e = (\bar{v}, \bar{v}') \\ g & \text{if } e = (\bar{v}', \bar{v}). \end{cases} \tag{4}$$

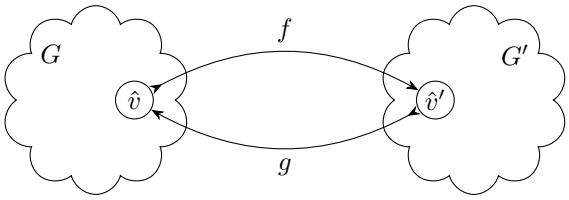
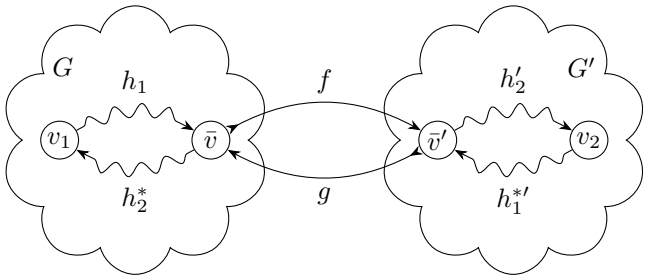


Figure 5 Secure composition of Lagois graphs.

Note that there is no need to define the operation in terms of multiple graphs, as we can chain the operation freely. We can then show that the operation $G \frac{f}{\bar{v}} \rightleftharpoons \frac{\bar{v}'}{g} G'$ preserves security.

▶ **Theorem 28.** *If G and G' are disjoint and secure graphs then for all $\bar{v} \in G$, $\bar{v}' \in G$ and f, g such that $(L(\bar{v}), f, g, L(\bar{v}'))$ is a Lagois connection then $G \frac{f}{\bar{v}} \rightleftharpoons \frac{\bar{v}'}{g} G'$ is secure.*

Proof. By Proposition 21 it is enough to consider all pairs of simple paths $h = (v_1, \_), \ldots, (\_, v_2)$, $h^* = (v_2, \_), \ldots, (\_, v_1)$ for all pairs of vertices $v_1, v_2 \in G \frac{f}{\bar{v}} \rightleftharpoons \frac{\bar{v}'}{g} G'$. If each vertex resides exclusively within G then h and h^* must also be contained within G, because if either h or h^* were to pass over into G' it must have been through $e_f = (\bar{v}, \bar{v}') \in E''$ (and $e_g = (\bar{v}', \bar{v}) \in E''$ 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 $p \in L(v_1)$, now $(\widehat{h_1 \star e_f})(p) \leq (h_1 \star \widehat{e_f \star h_2'} \star h_1^{*\prime})(p)$ because $h_2' \star h_1^{*\prime}$ is secure by virtue of being a loop contained within G'. Then

$$\hat{h}_1(p) \leq (h_1 \widehat{\star e_f \star e_g})(p) \leq (h_1 \star e_f \widehat{\star h_2' \star} h_1^{*\prime} \star e_g)(p)$$

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

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

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 7 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, as depicted for some G in Figure 6. Theorem 28 is thus a generalisation of Corollary 26, in this regard it should be easy to see that Corollary 26 can be derived from Theorem 28 instead of Theorem 25.

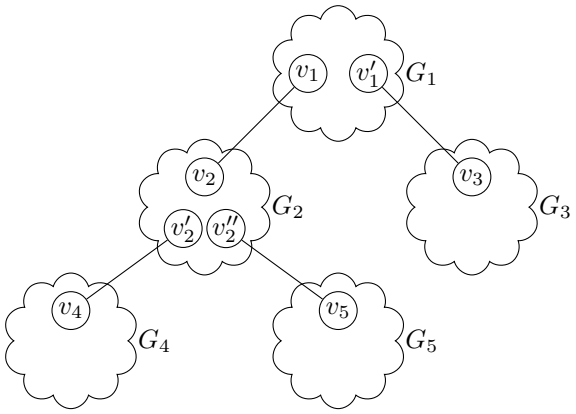


Figure 6 A forest of secure subgraphs.

5 Soundness (w.r.t. Noninterference)

As mentioned in section 1, the theory presented in the prequel seems well suited for describing secure information flow in distributed systems. Thus in this section we make the following notion precise: If information in some distributed program flows according to a secure Lagois graph G (i.e. the program respects G) then the distributed program observes noninterference. So far, our theory has been detached from any particular model of computation. We consider this a strength and seek to keep it that way. We take a knowledge-based approach, as shown by Askarov and Chong [1], in our description of when a program respects a graph and of noninterference. Unlike [1], the theory presented here is not parametric in the attacker, i.e., we assume that the attacker has perfect recollection.

Fix some Lagois graph G, a set S of program configurations, and a set of values N, then let the joint state of a distributed system over G be denoted by

$$\Sigma : \mathcal{S} = (v \in G) \to ((L(v) \to N) \times S). \tag{5}$$

Where $\Sigma(v) = (\rho, x)$ is the local state at v and ρ provides the security interface for which an attacker is able to observe the state. Essentially an attacker at v on level $\ell \in L(v)$ will be able to observe $\rho(\ell')$ for all $\ell' \leq \ell$. We let x denote the actual internal configuration of the program at that state. Similarly we will refer to $\Gamma : (v \in G) \to L(v) \to N$ as a distributed security interface.

We further assume there exists a *computation relation* $\longrightarrow \subseteq \mathcal{S} \times \mathcal{S}$ modelling the states of execution of a program. A trace is defined to be a sequence $\Sigma s = \Sigma_1, \Sigma_2, \ldots, \Sigma_n \in \mathcal{S}^*$ such that $\Sigma_1 \longrightarrow \Sigma_2 \longrightarrow \cdots \longrightarrow \Sigma_n$. Appendix A provides an example instance of $(\mathcal{S}, \longrightarrow)$. Take π_i to be the ith projection of a tuple; then for a given distributed state Σ the state observable by some attacker at v on level $\ell \in L(v)$ is denoted by $\Sigma|^v_\ell : L(v) \rightharpoonup N$ which is defined as

$$\Sigma|^v_\ell(\ell') = \begin{cases} \pi_1(\Sigma(v))(\ell') & \text{if } \ell' \leq \ell, \\ \text{nothing} & \text{otherwise.} \end{cases} \tag{6}$$

Given trace Σs, a partial trace observable by some attacker at v on level $\ell \in L(v)$ is denoted by $\Sigma s|^v_\ell : (L(v) \rightharpoonup N)^*$ which is defined as

$$()|^v_\ell = () \tag{7}$$

$$(\Sigma \star \Sigma s)|^v_\ell = \begin{cases} \Sigma s|^v_\ell & \text{if } \Sigma s|^v_\ell = \Sigma|^v_\ell \star \sigma s \\ \Sigma|^v_\ell \star \Sigma s|^v_\ell & \text{otherwise.} \end{cases} \tag{8}$$

Essentially $\Sigma s|_\ell^v$ applies $-|_\ell^v$ pointwise to Σs and filters out adjacent duplicates; we justify this choice by appealing to the intuition that an observer should not be able to detect a change if nothing happened from the observer's point of view.

We then say that a distributed state Σ can emit a partial trace $\sigma s \in (L(v) \rightharpoonup V)^*$ observable by an attacker at v on level $\ell \in L(v)$ denoted $\Sigma \Downarrow_\ell^v \sigma s$ iff there exists a trace Σs such that $\Sigma \star \Sigma s$ is a trace and $(\Sigma \star \Sigma s)|_\ell^v = \sigma s$.

We can now describe when an initial distributed state respects G in relation to how communication takes place: Define for some v and $\ell \in L(v)$ the equivalence relations on distributed security interfaces where all information that may flow into (v, ℓ) is equal:

$$\Gamma \approx_\ell^v \Gamma' \iff \Gamma(v', \ell') = \Gamma'(v', \ell') \text{ for all } (v', \ell') \rightrightarrows (v, \ell) \tag{9}$$

This gives rise to an equivalence class (for each Γ):

$$[\![\Gamma]\!]_\ell^v = \{\Gamma' \mid \Gamma \approx_\ell^v \Gamma'\}. \tag{10}$$

We define the knowledge of an attacker at v on level $\ell \in L(v)$ that is trying to guess the initial distributed security interface the system after observing σs as

$$\mathbb{K}_\ell^v(\sigma s) = \{\Gamma \mid \exists \Sigma. (\pi_1 \circ \Sigma) = \Gamma \wedge \Sigma \Downarrow_\ell^v \sigma s\}. \tag{11}$$

Now we are ready to define what it means for a distributed state to respect G in terms of information flow:

▶ **Definition 29.** *We say that* Σ respects G *whenever for all* $v \in G$, $\ell \in L(v)$ *and* σs *such that* $\Sigma \Downarrow_\ell^v \sigma s$ *it is the case that*

$$[\![\pi_1 \circ \Sigma]\!]_\ell^v \subseteq \mathbb{K}_\ell^v(\sigma s). \tag{12}$$

Essentially Definition 29 tells us if some Σ respects G then the attacker may only learn in accordance to the flow of G. Said otherwise, an attacker at (v, ℓ) may only learn any information about $\pi_1(\Sigma(v'))(\ell')$ whenever $(v', \ell') \rightrightarrows (v, \ell)$.

The above is a very strong notion of noninterference, but not the one in which we are directly interested. Specifically, we want to ensure that a system at v may never observe a violation of its own policy $L(v)$. We define an equivalence relation over security interfaces at v that will seem the same to an attacker on level $\ell \in L(v)$ as follows

$$\rho \sim_\ell^v \rho' \iff \rho(\ell') = \rho'(\ell') \text{ for all } \ell' \leq \ell \in L(v). \tag{13}$$

As above, this gives rise to an equivalence class

$$[\rho]_\ell^v = \{\rho' \mid \rho \sim_\ell^v \rho'\}. \tag{14}$$

We then define the knowledge of an attacker at v on level $\ell \in L(v)$ that is trying to guess the initial security interface at their location as

$$K_\ell^v(\sigma s) = \{\rho \mid \exists \Sigma. \pi_1(\Sigma(v)) = \rho \wedge \Sigma \Downarrow_\ell^v \sigma s\}. \tag{15}$$

The notion of noninterference we are interested in can then be defined in the following manner:

▶ **Definition 30.** *We say that* Σ *observes* noninterference *whenever for all* $v \in G$, $\ell \in L(v)$ *and* σs *such that* $\Sigma \Downarrow_\ell^v \sigma s$ *it is the case that*

$$[\pi_1(\Sigma(v))]_\ell^v \subseteq K_\ell^v(\sigma s) \tag{16}$$

Thus, Σ observing noninterference intuitively means that an attacker at v on level $\ell \in L(v)$ interested in learning about the initial security interface at that location, i.e., $\pi_1(\Sigma(v)) = \rho$, may only learn about $\rho(\ell')$ for all $\ell' \leq \ell \in L(v)$. Soundness with respect to noninterference can then be stated and proved in the following manner:

▶ **Theorem 31** (Soundness). *If Σ respects G and G is secure then Σ observes noninterference.*

Proof. Fix $v \in G$, $\ell \in L(v)$ and σs such that $\Sigma \Downarrow_\ell^v \sigma s$ then for some ρ we need to show

$$(\forall \ell' \leq \ell \in L(v).\pi_1(\Sigma(v))(\ell') = \rho(\ell')) \implies \exists \Sigma'.\pi_1(\Sigma'(v)) = \rho \wedge \Sigma' \Downarrow_\ell^v \sigma s \tag{17}$$

Take $\Gamma = (\pi_1 \circ \Sigma)[v \mapsto \rho]$. We have

$$\forall (v', l') \rightrightarrows (v, l).\pi_1(\Sigma(v'))(l') = \Gamma(v', \ell') \tag{18}$$

by case analysis on v':

$v' = v$ We have $\pi_1(\Sigma(v'))(l') = \Gamma(v', \ell') = \rho(\ell')$ by the assumption stated in Equation 17 and the fact that $(v, l') \rightrightarrows (v, l) \implies l' \leq l$ because G is secure.

$v' \neq v$ We have $\Gamma(v', \ell') = (\pi_1 \circ \Sigma)[v \mapsto \rho](v', \ell') = \pi_1(\Sigma(v'))(\ell')$.

By the assumption that Σ respects G and the fact of Equation 18 there exists Σ' such that $\Sigma' \Downarrow_\ell^v \sigma s$ and $(\pi_1 \circ \Sigma') = \Gamma$ which means $\pi_1(\Sigma'(v)) = \Gamma(v) = \rho$. ◀

This result is straightforward, but it does verify that our notion of Σ respecting G is safe with respect to noninterference. We leave exploration of further alternative formalisations for future work.

6 Formalisation

The formalisation covers the developments both in this paper and in the 2019 paper by Bhardwaj and Prasad [2]. In particular all results presented in [2] have been verified independently by us; further, all results not marked with an "*" within this paper have also been formalised and verified. The formalisation provides a solid foundation for automating proofs of soundness of various enforcement mechanisms. These types of proofs are usually unwieldy and tedious yet lend themselves to automation.

The formalisation is written in the *Rocq* proof assistant, and the source is available online[5]. Our work is built on top of the *Mathematical Components* library (and *Hierarchy Builder*), mainly for its order theory module. The formalised theory does not depend on any additional axioms besides those necessitated by Rocq's type system. The formalisation is split into six Rocq modules, each of which is given an overview below: `lagois`, `abssyn`, `sem`, `typrls`, `lagoisgraph`, and `soundness`.

The `lagois`module contains a formal definition of Lagois connections denoted `Lagois`, defined in terms of a Hierarchy Builder structure. Further, it contains formal proofs for results due to Melton et al. [6]. In particular: Proposition 3.3, Proposition 3.7, Proposition 3.8, Theorem 3.9, Proposition 3.11.(1), Proposition 3.11.(2). Generally we follow Bhardwaj and Prasad [2, 3] and limit the formalisation of these proofs to finite lattices with top and bottom elements, as opposed to partial orders. The formal proofs follow the structure of the original proofs by Melton et al. very closely. For some theorems, Melton et al. provide an explicit transpose result. In the formalisation we have left out such results and instead register

[5] `https://github.com/CasperStaahl/LC4S-Rocq/releases/tag/vpaper`

the transpose of a Lagois connection as an instance of the `Lagois` structure, from which transpose results can be easily derived. The module also contains some miscellaneous lemmas and corollaries: `PC` is a proof of the fact that an instance of `Lagois` satisfies Equation **PC1** and Equation **PC2**. `PC`, `melton_3_7` and `melton_3_7_full` are essentially variations of each other. `PC` follows the original proof by Melton et al., and `melton_3_7`, `melton_3_7_full` are derived from it in an ad hoc fashion. `CC` is a proof that an instance of `Lagois` satisfies Equation **CC1** and Equation **CC2**.

The modules `abssyn`, `sem`, and `typrls` contain the formalisation of the abstract syntax, operational model, and type system (and soundness thereof) from [2, 3], respectively. The formalisation of the abstract syntax and operational model contain some minor and inconsequential differences to its informal counterpart. In particular we limit import-, export- and local variables to be members of the natural numbers, instead of them being members of the mutually disjoint sets Z, Z', X, X', Y and Y'. Further, the syntax and operational model have been rewritten to factor out duplicate grammar- and induction rules.

In `sem` the inductive definitions `commsem` and `phrasesem` correspond to the inductive rules defining judgements $\langle \mu, \mu' \rangle \vdash p \Rightarrow \langle \nu, \nu' \rangle$ in the operational model given in [2, 3]. More specifically rule `Tsem`, `WRsem` and `RDsem` in `commsem` along with rule `Lcommsem` in `phrasesem` correspond to rules T (and T'), Wr (and Wr') and Rd (and Rd'), respectively. Here, `Lcommsem` factors out duplicate rules. Rules `TRLsem` and `TLRsem` in `phrasesem` correspond to rules Trl and Tlr, respectively. The inductive definition `seqsem` corresponds to the induction rules for the judgement $\langle \mu, \mu' \rangle \vdash s \Rightarrow^* \langle \nu, \nu' \rangle$. Specifically `seqsem` and `seqconssem` correspond to Seq0 and Seqs respectively. The `SubSemantic` Hierarchy Builder structure defined in `sem` formalises the constraints placed on the atomic transactional operations ranged over by t and t' in [2, 3]. We overload the notation `_ ==[ _ ]==> _` such that $\mu \vdash t \Rightarrow \nu$ has a formal counterpart μ `==[ t ]==>` ν, $\langle \mu, \mu' \rangle \vdash p \Rightarrow \langle \nu, \nu' \rangle$ has a formal counterpart `(`μ`, `μ`') ==[ p ]==> (`ν`, `ν`')` as defined by `phrasesem`, and $\langle \mu, \mu' \rangle \vdash s \Rightarrow^* \langle \nu, \nu' \rangle$ has a formal counterpart `(`μ`, `μ`') ==[ s ]==> (`ν`, `ν`')` as defined by `seqsem`.

The inductive definitions `CommSType`, `PhraseSType`, `SeqSType` in the module `typrls` define the formal counterpart to the type system presented in [2, 3] with duplicate rules factored out. Specifically, rules `Tt`, `Trd` and `Twr` in `CommSType` correspond to rule Tt (and Tt'), Trd (and Trd') and Twr (and Twr'), respectively, with rules `LCT` and `RCT` in `PhraseSType` factoring out duplicate rules. Further, formal rules `TTrl` and `TTlr` in `PhraseSType` correspond to rules TT_{RL} and TT_{LR}', respectively. Finally, formal rules `Com0` and `ComP` in `SeqSType` correspond to rules Com0 and ComP respectively. Our formal proof of soundness `SeqSType_sound` for the type system follows the same structure as that of [2, 3], although in some cases a case analysis is replaced by a direct argument incidentally fixing a few missing cases in the analysis. Essentially, the formal proof proceeds by induction on the derivations of `seqsem`, from which a case analysis is conducted on the resulting derivations of `phrasesem` and `commsem`. The lemma `CommsSType_sound` corresponds to the case analysis on members in `commsem`, and the lemma `T_sound` corresponds to the remaining cases where `TTrl` (or `TTlr`) is the bottom-most rule used in the derivation of the transition.

The module `lagoisgraph` pertains to the formalisation of the content presented in sections 3 and 4. Besides the results presented in sections 3 and 4 the module contains several helper lemmas not covered here for the sake of brevity. The `LagoisGraph` module gives the formal definition of a Lagois graph as a Hierarchy Builder structure. Notably, the formalisation does not contain an underlying notion of a graph:

```
HB.mixin Record IsLagoisGraph V of Equality V := {
  lattice : V -> {d & porderType d} ;
  edge v v' : bool ;
  label v v' : edge v v' ->
      Lagois.type (projT2 (lattice v)) (projT2 (lattice v')) ;
  edge_irefl v : edge v v = false ;
  edge_sym v v' : edge v v' -> edge v' v ;
  label_sym v v' p p' : (label v v' p).1 =1 (label v' v p').2;
}.
HB.structure Definition LagoisGraph :=
    {T of IsLagoisGraph T & Equality T}.
```

Essentially, a Lagois graph is a type `V` equipped with a function `lattice` that maps points in `V` to their respective partial orders. In this regard the formalisation is more general, allowing the codomain of `lattice` to be the type of partial orders instead of the type of lattices. `edge` is the edge relation that is interpreted in the expected manner. `label` is a function labelling edges, i.e. proofs of `edge v v' = true`, with a Lagois connection between the lattices associated to `v` and `v'`. `edge_irefl` enforces that there are no immediate self-loops. `edge_sym` enforces that the edge relation is symmetric, and `label_sym` enforces that symmetric edges agree in their associated Lagois connections. Further, we do not require that the underlying type that represents the vertices in the graph is finite, only that it has decidable equality. We use `L(v)` to denote the underlying poset of a vertex `v` for some Lagois graph, and denote the presence of an edge between vertices `v` and `v'` as `v @> v'`. Further, we denote the type of paths, as defined by `path`, between `v` and `v'` in `G` as `v ~> v' :> G` (or `v ~> v'`) and path concatenation as `\*`. The formal inductive definition `flow` corresponds to the informal definition of the flow relation (Definition 14). The formal definitions `flow_secure` and `flow_secure_graph` correspond to the informal definition of flow secure (Definition 15). Similarly, the formal definitions `loop_secure`, `loop_secure_vertex` and `loop_secure_graph` correspond to the informal definition of loop secure (Definition 16). Lemma 17 and Proposition 18 have formal counterparts `flow2path` and `flow_secure_graph_is_loop_secure`. `simply_secure` is the formalisation of Definition 19, with related Lemma 20 and Proposition 21 formalised as `simply_secure_iff_loop_secure` and `path_nondecreasing`, respectively. The proof `simply_secure_iff_loop_secure` is based on an induction principle over loops in a graph instead of strong induction, the induction principle is defined in two parts as:

```
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 : v @> 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 :=
  (...)
Definition loop_ind (P : forall v : G, v ~> v :> G -> Prop)
    (bc_1 : forall v : G, P v (ε))
    (bc_2 : forall (v v' : G) (g : v @> 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) : P v f :=
  loop_ind_aux bc_1 bc_2 ic (subloopwf (existT _ v f)).
```

The induction principle is derived as described by [5] and the general pattern used in the proof of Proposition 21. That is, the induction principle is first written as a general recursive function, which is generally not allowed in Rocq. Then we add the assumption that `f` is accessible through the strict ordering of loops based on their length i.e., `ACC`. This allows us to do structural recursion on `ACC`, which is allowed in Rocq, 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. Indeed, 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`.

Moving on to the part of the module `lagoisgraph` in which section 4 has been formalised: For Lemma 22 we have the formal counterpart `reverse_path_loop_secure`, the reader should note that `f^<~` denotes the reverse path of `f`. The Hierarchy Builder structures `LagoisVForest` and `LagoisForest` correspond respectively to the definitions of virtual forests (Definition 24) and forests (Definition 23), with associated Theorem 25 and Corollary 26 formalised as `VForest_loop_secure` and `Forest_loop_secure`. The secure composition operation (Definition 27) is formally defined in terms of the carrier type:

```
Inductive Bridge (G G' : LagoisGraph.type)
    (v_abut : G) (v'_abut : G')
    (fg : Lagois.type L(v_abut) L(v'_abut)) : Type :=
  | lbank : G -> Bridge fg
  | rbank : G' -> Bridge fg.
```

Thus for $G \overset{f}{\underset{g}{\rightleftharpoons}}{}^{\bar{v}}_{\bar{v}'} G'$: `G`, `G'` correspond to G, G'; `v_abut`, `v'_abut` correspond to $\bar{v}, \bar{v}'$; and `fg` correspond to $(L(\bar{v}), f, g, L(\bar{v}'))$. The carrier type, `Bridge G G' v_abut v_abut' fg` (or `Bridge fg` when the preceding arguments can be inferred), is equivalent to `G + G'`. The carrier type is then made a canonical instance of the Lagois graph structure:

```
Context (G G' : LagoisGraph.type) (v_abut : G) (v'_abut : G')
    (fg : Lagois.type L(v_abut) L(v'_abut)).
(...)
HB.instance Definition _ :=
    IsLagoisGraph.Build (Bridge fg)
    Bridge_edge_irefl Bridge_edge_sym Bridge_label_sym.
```

such that `Bridge_edge_irefl`, `Bridge_edge_sym`, `Bridge_label_sym`, and the inferred arguments are defined in the expected manner. Finally, module `lagoisgraph` finishes with theorem `Bridge_secure` asserting formally that secure composition is indeed secure (Theorem 7).

Finally, the module `soundness` covers the formal counterpart to section 5. Variables G, S, N and $\mathcal{S}$ have formal counterparts with the same symbols, however the formal definitions of N and $\mathcal{S}$ are slightly more general. In particular, `N : forall (v : G) (ℓ: L(v)), Type` and $\mathcal{S}$ `:= forall (v : G), (forall ℓ: L(v), N ℓ) * S`, that is, `N` may depend on ℓ. $\longrightarrow \subseteq \mathcal{S} \times \mathcal{S}$ has the formal equivalent `R` and `Trace` is the type of traces. `Interface` gives the

type of distributed security interfaces informally ranged over by Γ. Similarly, `LocalInterface` gives the type of local security interfaces informally ranged over by ρ. $\Sigma|_\ell^v$ and $\Sigma s|_\ell^v = \sigma s$ have respective formal counterparts `obs v ℓ Σ : (forall ℓ' : L(v)), option (N ℓ')` and `Obs v ℓ Σs σs : Prop`, and `emits v ℓ Σ σs` formalises the emission predicate $\Sigma \Downarrow_\ell^v \sigma s$. `distr_interface_equiv`, `distr_interface_class`, `local_interface_equiv` and `local_interface_class` are the formal counterparts to $\approx$, $[\![\_]\!]$, $\sim$, and $[\_]$, respectively. The knowledge of an attacker $\mathbb{K}$ and K are denoted with the same symbols in the formalisation. `respects` and `noninterference` respectively formalise Definition 29 and Definition 30. Finally, Theorem 31 has the formal counterpart `respects2noninterference`.

7 Related Work

Lagois connections are originally due to the seminal work of Melton et al. [6] in which results are presented that are analogous to similar 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 several results are presented and adapted from [6] that are relevant for secure information flow. The framework is illustrated for a restricted language and a type system is developed that is sound with respect to noninterference for secure information flow between two organizations.

In [3], Bhardwaj and Prasad extend their prior work. Firstly, to establish and update secure connections based on the Lagois connection framework, again relying on results from [6]. In particular, they cover the extension from two actors to an arbitrary number, as discussed in section 1 and 2. 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 the type system of Bhardwaj and Prasad. Most importantly, the type system can deal with an arbitrary number of actors/organizations. 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 15; it is thus unclear how strong the noninterference guarantees are.[6]

In [1], Askarov and Chong present an alternative perspective on 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 particular they identify a collection of relevant attackers, most notably 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. The framing in [1] is quite different from ours: It is mainly 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.

[6] [9], also by Nielseon et al., builds on the concepts of [8]. However, the ability for actors to move between clusters is however not covered.

8 Conclusion and Future Work

As mentioned in section 1 and 2, the main motivation for studying arbitrary graphs of Lagois connections instead of chains was to bypass intermediate lattices 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 underlying paths of the Lagois Graph and in the worst case computing and then checking the flow relation. Thus if we want to bypass a lossy intermediate, we cannot determine a Lagois graph to be secure by looking at the graph structure alone, although Theorem 28 tells us that we only have to check the parts of the graph that do not have the structure of a forest.

Our soundness result w.r.t. noninterference is independent of a specific model of computation and enforcement mechanism. A natural next step is thus to fix such a model and define an enforcement mechanism that ensures a given system state Σ respects the graph G over which it is defined. One such example could possibly be an adaptation of the model and enforcement mechanism presented in Nielson et al. [8, 9].

For the soundness result we take a state to be in $\mathcal{S} = (v \in G) \to ((L(v) \to N) \times S)$. However it may be convenient for N to depend on the value of $L(v)$, i.e., $\mathcal{S} = (v \in G) \to ((\ell \in L(v)) \to N(\ell)) \times S)$. For example in the example presented in Appendix A it would be nice to have

$$N(\ell) = \begin{cases} G \to \mathsf{Stmt} \cup \mathbf{1} & \text{if } \ell = \bot \\ \mathbb{N}^* & \text{otherwise} \end{cases}, \text{ instead of } N = \mathbb{N}^* \cup (G \to \mathsf{Stmt}) \cup \mathbf{1}$$

as it would allow an attacker to learn that high buffers are actually only buffers, which is intuitively safe as long as they do not learn about the actual contents. Indeed, the soundness result still holds for such $\mathcal{S}$. This is evident by the formalisation in which soundness is defined by- and proved for this more general $\mathcal{S}$ in a similar manner to the proof of soundness already presented herein.

Two concepts that our framework does not account for is dynamism of agents (à la Nielson et al. [8, 9]) and dynamism of policies (à la Askarov and Chong [1]). 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. Conversely [1] has dynamism of policies: A program can update its information flow policy via a `setPolicy`$(\sqsubseteq)$ primitive for some reflexive relation $\sqsubseteq$. We conjecture that our framework may be expanded to cover both types of dynamism using dependent functions:

$$G \to_{\mathcal{G}} G' = (v \in G) \to \Big((v' \in G) \times \big(L(v) \to L(v')\big)\Big). \tag{19}$$

For $F \in G \to_{\mathcal{G}} G'$ and $v \in G$ we would then interpret $F(v)$ as v moving to or becoming $\pi_1(F(v))$ and v's lattice transforming as specified by $\pi_2(F(v))$. More specifically $\pi_2(F(v))(p) = p'$ would denote label p becoming label p'. We leave this for future work.

Finally, we would like to thank the reviewers for their detailed, constructive, and very helpful feedback.

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*, pages 308–322. IEEE, 2012. doi:10.1109/CSF.2012.31.

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*, pages 75–92. Springer, 2019. doi:10.1007/978-3-030-21759-4_5.

3 Chandrika Bhardwaj and Sanjiva Prasad. Secure information flow connections. *Journal of Logical and Algebraic Methods in Programming*, 127:100761, 2022. doi:10.1016/J.JLAMP.2022.100761.

4 Dorothy E Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976. doi:10.1145/360051.360056.

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. *Theoretical Computer Science*, 136(1):79–107, 1994.

7 Andrew C Myers and Barbara Liskov. A decentralized model for information flow control. *ACM SIGOPS Operating Systems Review*, 31(5):129–142, 1997. doi:10.1145/268998.266669.

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*, pages 280–294. Springer, 2020. doi:10.1007/978-3-030-61470-6_17.

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*, pages 312–331. Springer, 2021. doi:10.1007/978-3-030-91631-2_17.

10 Casper Ståhl. Extending Lagois connections for secure information flow to n organizations. Master's thesis, Dept. of Computer Science, Aalborg University, 2025.

11 Casper Ståhl, Levs Gondelman, René Rydhof Hansen, and Danny Bøgsted Poulsen. LC4S-Rocq. Software, swhId: swh:1:dir:828cd84332eae467073735e4aee31198815adf58 (visited on 2026-07-23). URL: https://github.com/CasperStaahl/LC4S-Rocq, doi:10.4230/artifacts.27330.

A Example instance of computational model

In this appendix we provide an example instance of $(\mathcal{S}, \longrightarrow)$ in the form of a distributed while language with communication taking place between some buffers. Fix a set of variables Var and let x be a metavariable ranging over this set. We then define a simple while language with statements Stmt and expressions defined Exp inductively as:

$$e \in \mathsf{Exp} ::= n \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 = e_2 \mid \neg\, e \mid e_1 \,\texttt{mod}\, e_2 \mid e_1 / e_2$$
$$s \in \mathsf{Stmt} ::= \texttt{skip} \mid \texttt{get}_\ell(x) \mid \texttt{send}_\ell^v(x) \mid x := e \mid \texttt{if}\ e\ \texttt{then}\ s\ \texttt{else}\ s \mid \texttt{while}\ e\ \texttt{do}\ s \mid s_1; s_2$$

Fix some Lagois graph G, and set $N = \mathbb{N}^* \cup (G \to \mathsf{Stmt}) \cup \mathbf{1}$, $S = \mathsf{Stmt} \times (\mathsf{Var} \to \mathbb{N})$ such that

$$\mathcal{S} = (v \in G) \to ((L(v) \to \mathbb{N}^* \cup (G \to \mathsf{Stmt}) \cup \mathbf{1}) \times \mathsf{Stmt} \times (\mathsf{Var} \to \mathbb{N})).$$

Then for each $\Sigma \in \mathcal{S}$ we interpret $\Sigma(v) = (\rho, s, m)$ for some v as ρ is the current state of the buffers at v at a given level, except at level $\bot$ (the lowest level in $L(v)$) which contains the empty buffer or the initial distributed statement that was running in the system. Let s be the current statement executed at v, $\texttt{get}_\ell(x)$ denotes that the head of a buffer is retrieved and stored in variable x, and $\texttt{send}^v_\ell(x)$ indicates that the value of variable x is sent to the buffer associated to ℓ at v. The memory configuration for local variables at v is denoted m and we let $m(e)$ denote the evaluation of expression e with respect to memory $m \in \mathsf{Var} \to \mathbb{N}$. Let Sig be the set of signals ranged over by metavariable α that is inductively defined as:

$$\alpha ::= n?\ell \mid n!(v, \ell)$$

such that $n?\ell$ is interpreted as n was received in the buffer associated to ℓ, and $n!(v,\ell)$ as n was sent to buffer associated to ℓ at v. We then define $\longrightarrow$ in terms of another relation with judgements $v \vdash (\rho, s, m) \longrightarrow'_\alpha (\rho', s', m')$ such that $v \in G$ and $(\rho, s, m), (\rho', s', m') \in (L(v) \to \mathbb{N}^* \cup (G \to \mathsf{Stmt}) \cup \mathbf{1}) \times \mathsf{Stmt} \times (\mathsf{Var} \to \mathbb{N})$ that is defined inductively as seen below. $v \vdash (\rho, s, m) \longrightarrow'_\alpha (\rho', s', m')$ should intuitively be read as: in one step local state (ρ, s, m) at v can go to state (ρ', s', m') emitting signal α.

$$\frac{m' = m[x \mapsto m(e)]}{v \vdash (\rho, x := e, m) \longrightarrow'_\epsilon (\rho, \texttt{skip}, m')}, \quad \frac{v \neq v' \qquad \ell' \in L(v') \qquad m(x) = n}{v \vdash (\rho, \texttt{send}^{v'}_{\ell'}(x), m) \longrightarrow'_{n!(v',\ell')} (\rho, \texttt{skip}, m)},$$

$$\frac{\ell \in L(v) \qquad \rho(\ell) = n \star ns}{v \vdash (\rho, \texttt{get}_\ell(x), m) \longrightarrow'_\epsilon (\rho[\ell \mapsto ns], \texttt{skip}, m[x \mapsto n])}, \quad \frac{\ell \in L(v) \qquad \rho(\ell) \neq n \star ns}{v \vdash (\rho, \texttt{get}_\ell(x), m) \longrightarrow'_\epsilon (\rho, \texttt{skip}, m)},$$

$$\frac{\ell \in L(v) \qquad \rho(\ell) \in \mathbb{N}^*}{v \vdash (\rho, s, m) \longrightarrow'_{n?\ell} (\rho[\ell \mapsto ns \star n], s, m)}, \quad \frac{\ell \in L(v) \qquad \rho(\ell) \notin \mathbb{N}^*}{v \vdash (\rho, s, m) \longrightarrow'_{n?\ell} (\rho[\ell \mapsto n], s, m)},$$

$$\frac{m(e) = 0}{v \vdash (\rho, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2, m) \longrightarrow'_\varepsilon (\rho, s_2, m)}, \quad \frac{v \vdash (\rho, s_1, m) \longrightarrow_\alpha (\rho', s_1', m')}{v \vdash (\rho, s_1; s_2, m) \longrightarrow_\alpha (\rho', s_1'; s_2, m')},$$

$$\frac{m(e) > 0}{v \vdash (\rho, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2, m) \longrightarrow'_\varepsilon (\rho, s_1, m)}, \quad \frac{}{v \vdash (\rho, \texttt{skip}; s, m) \longrightarrow_\varepsilon (\rho, s, m)},$$

$$\frac{}{v \vdash (\rho, \texttt{while } e \texttt{ do } s, m) \longrightarrow_\varepsilon (\rho, \texttt{if } e \texttt{ then } (s; \texttt{while } e \texttt{ do } s) \texttt{ else skip}, m)}.$$

Let $\mathsf{INIT}(\Sigma)$, and $\mathsf{IM}(\Sigma)$ denote that the distributed security interface of Σ is initial and intermediate, respectively, which we formally define as follows:

$$\mathsf{INIT}(\Sigma) = \forall v \in G. \pi_1(\Sigma(v))(\bot) \in \mathbf{1}$$
$$\mathsf{IM}(\Sigma) = \forall v \in G. \pi_1(\Sigma(v))(\bot) \in (G \to \mathsf{Stmt})$$

Then $\longrightarrow$ can be defined, with judgements $\Sigma \longrightarrow \Sigma'$, as:

$$\frac{\mathsf{INIT}(\Sigma)}{\Sigma \longrightarrow \Sigma[v \mapsto (\pi_1(\Sigma(v))[\bot \mapsto \pi_2 \circ \Sigma], \pi_2(\Sigma(v)), \pi_3(\Sigma(v))) \mid v \in G]},$$

$$\frac{v' \vdash \Sigma(v') \longrightarrow'_\varepsilon \sigma \qquad \mathsf{IM}(\Sigma)}{\Sigma \longrightarrow \Sigma[v' \mapsto \sigma]}, \quad \frac{\begin{array}{c} v_1 \vdash \Sigma(v_1) \longrightarrow_{n!(v_2,\ell)} \sigma_1 \\ v_2 \vdash \Sigma(v_2) \longrightarrow_{n?\ell} \sigma_2 \qquad \mathsf{IM}(\Sigma) \end{array}}{\Sigma \longrightarrow \Sigma[v_1 \mapsto \sigma_1, v_2 \mapsto \sigma_2]}.$$

To further concretise the example fix a lattice $\mathcal{L} = \bot \leq l \leq h$ and a Lagois graph $G = (\{v_1, v_2\}, \{(v_1, v_2), (v_2, v_1)\}, L, F)$ such that $L(v_1) = L(v_2) = \mathcal{L}$ and $F((v_1, v_2)) = F((v_2, v_1)) = \mathrm{id}$. Then $\Sigma(v_1) = (\rho_1, s_1, m_2)$ and $\Sigma(v_2) = (\rho_2, s_1, m_2)$ where $\rho_1(\bot) = \rho_2(\bot) = *$, and s_1, s_2 are given by Figure 7, and Σ therefore observes noninterference whenever $42 < \pi_3(\Sigma(v_1))(\mathtt{x})$ because in that case Σ respects G which is secure.

```
while 1 do (
  send_h^{v2}(x);
  x := x + 1
)
```

(a) Statement s_1.

```
while 1 do (
  get_h(x);
  if x = 42 then
    send_l^{v1}(x)
  else
    send_h^{v1}(x)
)
```

(b) Statement s_2.

Figure 7 Statements s_1 and s_2.