Reconciling Shannon and Scott with a Lattice of Computable Information

Citation for the original published paper (version of record):
Hunt, S., Sands, D., Stucki, S. (2023). Reconciling Shannon and Scott with a Lattice of Computable Information. Proceedings of the ACM on Programming Languages, 7: 1987-2016.
http://dx.doi.org/10.1145/3571740

Reconciling Shannon and Scott with a Lattice of Computable Information

SEBASTIAN HUNT, City, University of London, United Kingdom
DAVID SANDS, Chalmers University of Technology, Sweden
SANDRO STUCKI*, Amazon Prime Video, Sweden

This paper proposes a reconciliation of two different theories of information. The first, originally proposed in a lesser-known work by Claude Shannon (some five years after the publication of his celebrated quantitative theory of communication), describes how the information content of channels can be described *qualitatively*, but still abstractly, in terms of *information elements*, where information elements can be viewed as equivalence relations over the data source domain. Shannon showed that these elements have a partial ordering, expressing when one information element is more informative than another, and that these partially ordered information elements form a complete lattice. In the context of security and information flow this structure has been independently rediscovered several times, and used as a foundation for understanding and reasoning about information flow.

The second theory of information is Dana Scott's domain theory, a mathematical framework for giving meaning to programs as continuous functions over a particular topology. Scott's partial ordering also represents when one element is more informative than another, but in the sense of computational progress – i.e. when one element is a more defined or evolved version of another.

To give a satisfactory account of information flow in computer programs it is necessary to consider both theories together, in order to understand not only what information is conveyed by a program (viewed as a channel, à la Shannon) but also how the precision with which that information can be observed is determined by the definedness of its encoding (à la Scott). To this end we show how these theories can be fruitfully combined, by defining *the Lattice of Computable Information* (LoCI), a lattice of preorders rather than equivalence relations. LoCI retains the rich lattice structure of Shannon's theory, filters out elements that do not make computational sense, and refines the remaining information elements to reflect how Scott's ordering captures possible varieties in the way that information is presented.

We show how the new theory facilitates the first general definition of termination-insensitive information flow properties, a weakened form of information flow property commonly targeted by static program analyses.

CCS Concepts: • **Theory of computation** → *Program analysis*; *Denotational semantics*; • **Security and privacy** → **Information flow control**.

Additional Key Words and Phrases: Information Flow, Semantics

ACM Reference Format:
Sebastian Hunt, David Sands, and Sandro Stucki. 2023. Reconciling Shannon and Scott with a Lattice of Computable Information. *Proc. ACM Program. Lang.* 7, POPL, Article 68 (January 2023), 30 pages. https://doi.org/10.1145/3571740

*This publication was written while the third author was at Chalmers, prior to joining Amazon.

Authors' addresses: Sebastian Hunt, s.hunt@city.ac.uk, City, University of London, London, United Kingdom; David Sands, dave@chalmers.se, Chalmers University of Technology, Gothenburg, Sweden; Sandro Stucki, sastucki@amazon.com, Amazon Prime Video, Gothenburg, Sweden.

2475-1421/2023/1-ART68
https://doi.org/10.1145/3571740

1 INTRODUCTION

Note to Reader: this paper is not about information theory [Shannon 1948], but about a theory of information [Shannon 1953].

1.1 What is the Information in Information Flow?

The study of information flow is central to understanding many properties of computer programs, and in particular for certain classes of confidentiality and integrity properties. In this paper we are concerned with providing a better semantic foundation for studying information flow.

The starting point for understanding information flow is to understand information itself. Shannon's celebrated theory of information [Shannon 1948] naturally comes to mind, but Shannon's theory is a theory about *quantities* of information, and purposefully abstracts from the information itself. In a relatively obscure paper[1], Shannon [1953] himself notes:

> ... $H(X)$ [the entropy of a channel X] can hardly be said to represent the actual information. Thus, two entirely different sources might produce information at the same rate (same H) but certainly they are not producing the same information.

Shannon goes on to introduce the term *information elements* to denote the information itself. The concept of an information element can be derived by considering some channel – a random variable in Shannon's world, but we can think of it as simply a function f from a "source" domain to some "observation" codomain – and asking what information does f produce about its input. Shannon's idea was to view the information itself as the set of functions which are equivalent, up to bijective postprocessing, with f, i.e. $\{b \circ f \mid b \text{ is bijective on the range of } f\}$ – in other words, all the alternative ways in which the information revealed by f might be faithfully represented.

Shannon observed that information elements have a natural partial ordering, reflecting when one information element is subsumed by (represents more information than) another, and that any set of information elements relating to a common information source domain can be completed into a *lattice*, with a least-upper-bound representing any information-preserving combination of information elements, and a greatest-lower-bound, representing the common information shared by two information elements, thus providing the title of Shannon's note: "A Lattice Theory of Information". Shannon observes that any such lattice of information over a given source domain can be embedded into a general and well-known lattice, namely the lattice of equivalence relations over that source domain [Ore 1942]. In fact, the most precise lattice of information for a given domain, i.e. the one containing all information elements over that domain, is isomorphic to the lattice of equivalence relations over that domain. In the remainder of this paper will think in terms of the most precise lattice of information for any given domain.

This lattice structure, independently dubbed *the lattice of information* by Landauer and Redmond [1993], can be used in a uniform way to phrase a large variety of interesting information flow questions, from simple confidentiality questions (is the information in the public output channel of a program no greater than in the public input data?), to arbitrarily fine-grained, potentially conditional information flow policies. The lattice of information, described in more detail in §2, is the starting point of our study.

1.2 Shortcomings of the Lattice of Information

The lattice of information provides a framework for reasoning about a variety of information flow properties in a uniform way. It is natural in this approach, to view programs as functions from an input domain to some output domain. But this is where we hit a shortcoming in the lattice of

[1]With around 150 citations, a factor of 1000 fewer than his seminal work on information theory [Shannon 1948] (source: Google Scholar); according to Rioul et al. [2022], all but ten of these actually intended to cite the 1948 paper.

information: program behaviours may be *partial*, ranging from simple nontermination, to various degrees of partiality when modelling structured outputs such as streams. While these features can be modelled in a functional way using domain theory (see e.g. [Abramsky and Jung 1995]) the lattice of information is oblivious to the distinction between degrees of partiality.

Towards an example, consider the following two Haskell functions:

```
parity1 x = if even x then 1 else 0

parity2 x = if even x then "Even" else "Odd"
```

Even though these two functions have different codomains, intuitively they release the same information about their argument, albeit encoded in different ways. In Shannon's view they represent the same information element. The information released by a function f can be represented simply by its *kernel* – the smallest equivalence relation that relates two inputs whenever they get mapped to the same output by f. It is easy to see that the two functions above have the same kernel.

What about programs with partial behaviours? A natural approach is to follow the denotational semantics school, and model nontermination as a special "undefined" value, $\bot$, and more generally to capture nontermination and partiality via certain families of partially ordered sets (*domains* [Abramsky and Jung 1995]) and to model programs as continuous functions between domains. Consider this example:

```
parity0 x = if even x then 1 else parity0 x
```

Here the program returns 1 if the input is even, and fails to terminate otherwise. The kernel of (the denotation of) this function is the same as the examples above, which means that it is considered to reveal the same amount of information. But intuitively this is clearly not the case: `parity0` provides information in a less useful form than `parity1`. When the input is odd, an observer of `parity0` will remain in limbo, waiting for an output that never comes, whereas an observer of `parity1` will see the value 0 and thus learn the parity of the input. The two are only equivalent from Shannon's perspective if we allow *uncomputable* postprocessors. (Of course, we are abstracting away entirely from timing considerations here. This is an intrinsic feature of the denotational model, and a common assumption in security reasoning.)

Intuitively, `parity0` provides information which is consistent with `parity1`, but the "quality" is lower, since some of the information is encoded by nontermination.

Now consider programs `A` and `B`, where the input is the value of variable `x` and the output domain is a channel on which multiple values may be sent. Program `A` simply outputs the absolute value of `x`. Program `B` outputs the same value but in unary, as a sequence of outputs, then silently diverges.

Just as in the previous example, `A` and `B` compute functions which have the same kernel, so in the lattice of information they are equivalent. But consider what we can actually deduce from `B` after observing n output events: we know that the absolute value of `x` is some value $\geq n$, but we cannot infer that it is exactly n, since we do not know whether there are more outputs yet to come, or if the program is stuck in the final loop. By contrast, as soon as we see the output of A, we know with certainty the absolute value of `x`. In summary, the lattice of information fails to take into account that information can be encoded at different degrees of definedness[2].

```
A:  output(abs(x))

B:  y := abs(x);
    for i := 1 to y {
      output ()
    };
    while True { };
```

[2]Here we have drawn, albeit very informally, on foundational ideas developed by Smyth [1983], Abramsky [1987, 1991] and Vickers [1989], which reveal deep connections between domain theory, topology and logics of observable properties.

A second shortcoming addressed in this paper, again related to the lattice of information's unawareness of nontermination, is its inability to express, in a general non ad hoc way, a standard and widely used weakening of information flow properties to the so-called *termination-insensitive* properties [Sabelfeld and Myers 2003; Sabelfeld and Sands 2001]. (When considering programs with stream outputs, they are also referred to as *progress-insensitive* properties [Askarov and Sabelfeld 2009].) These properties are weakenings of information flow policies which ignore any information which is purely conveyed by the definedness of the output (i.e. termination in the case of batch computation, and progress in the case of stream-based output).

1.3 Contributions

Contribution 1: A refined lattice of information. In this paper we present a new abstraction for information, the *Lattice of Computable Information* (LoCI), which reconciles Shannon's lattice of information with Scott's domain ordering (§3). It does so by moving from a lattice of equivalence relations to a lattice of preorder relations, where the equivalence classes of the preorder reflect the "information elements", and the ordering between them captures the distinction in quality of information that arises through partiality and nontermination (the Scott ordering). Just as with the lattice of information, LoCI induces an information ordering relation on functions; in this ordering, `parity0` is less than `parity1`, but `parity1` is still equivalent to `parity2`. Similarly programs A and B above are related but not equivalent. We show that LoCI is, like the lattice of information, well behaved with respect to various composition properties of functions.

Contribution 2: A generalised definition of termination-insensitive noninterference. The lattice of computable information gives us the ability to make finer distinctions about information flow with respect to progress and termination. By modelling this distinction we also have the ability to systematically ignore it; this provides the first uniform generalisation of the definition of termination-insensitive information flow properties (§4).

The remainder of the paper begins with a review of the lattice of information (§2), which is followed by our refinement (§3), the treatment of termination-insensitivity (§4), a discussion of related work (§5), and some directions for further work (§6).

2 THE LATTICE OF INFORMATION

The lattice of information is a way to abstract the information about a data source D which might be revealed by various functions over that data. Mathematically, it is simply the set of equivalence relations over D, ordered by reverse inclusion, a structure that forms a *complete lattice* [Ore 1942], i.e. every set of elements in the lattice has a least upper bound and a greatest lower bound. The lattice of information has been rediscovered in several contexts relating to information and information flow, e.g. using partial equivalence relations (PERs) [Hunt 1991; Hunt and Sands 1991]. Here we use the terminology from Landauer and Redmond [1993] who call it the *lattice of information*.

To introduce the lattice of information let us consider a simple set of values

$$D = \{\text{Red}, \text{Orange}, \text{Green}, \text{Blue}\}$$

and the following three functions over D:

$$\text{isPrimary}(c) = \begin{cases} \text{True} & \text{if } c \in \{\text{Red}, \text{Blue}\} \\ \text{False} & \text{otherwise} \end{cases} \qquad \text{isTrafficLight}(c) = \begin{cases} \text{False} & \text{if } c = \text{Blue} \\ \text{True} & \text{otherwise} \end{cases}$$

$$\text{primary}(c) = \begin{cases} \text{“The primary colour red”} & \text{if } c = \text{Red} \\ \text{“The primary colour blue”} & \text{if } c = \text{Blue} \\ \text{“Not a primary colour”} & \text{otherwise} \end{cases}$$

Now consider the information that each of these functions reveals about its input: *isPrimary* and *isTrafficLight* reveal incomparable information about their inputs – for example we cannot define either one of them by postprocessing the result of the other. The function *primary*, however, not only subsumes both of them, but represents *exactly* the information that the pair of them together reveal about the input, nothing more, nothing less. The lattice of information (over D) makes this precise by representing the information itself as an equivalence relation on the elements of D. Elements that are equivalent for a given relation are elements which we can think of as indistinguishable.

Definition 1 (Lattice of Information). *For a set D, the lattice of information over D,* $\mathrm{LoI}(D)$*, is defined to be the lattice*

$$\mathrm{LoI}(D) = (\mathrm{ER}(D), \sqsubseteq_{\mathrm{LoI}}, \sqcup_{\mathrm{LoI}}, \sqcap_{\mathrm{LoI}})$$

where $\mathrm{ER}(D)$ *is the set of all equivalence relations over D, $P \sqsubseteq_{\mathrm{LoI}} Q \stackrel{\text{def}}{=} Q \subseteq P$, the join operation $\sqcup_{\mathrm{LoI}}$ is set intersection of relations, and the meet, $\sqcap_{\mathrm{LoI}}$, is the transitive closure of the set-union of relations.*

Note that $\mathrm{LoI}(D)$ is a complete lattice (contains all joins and meets, not just the binary ones) [Ore 1942]. The top element of $\mathrm{LoI}(D)$ is the identity relation on D, which we write as Id_D, or just Id when D is clear from context; the bottom element is the relation which relates every element to every other element, which we write as All_D, or just All.

In the above definitions we consider equivalence relations to be sets of pairs of elements of D. Another useful way to view equivalence relations is as partitions of D into disjoint blocks (equivalence classes). Given an equivalence relation P on a set D and an element $a \in D$, let $[a]_P$ denote the (necessarily unique) equivalence class of P which contains a. Let $[P]$ denote the set of all equivalence classes of P. Note that $[P]$ is a partition of A.

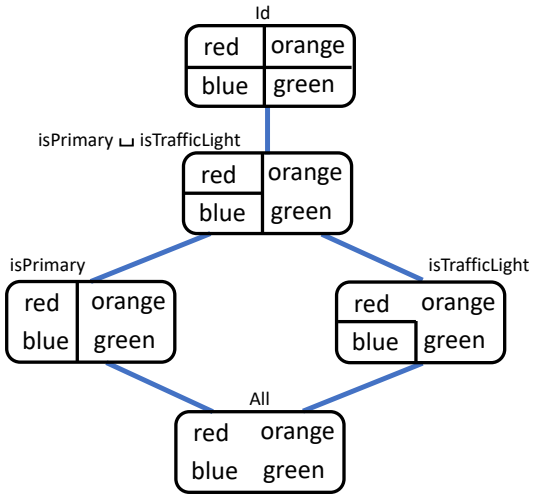


Fig. 1. An Example Sublattice of the lattice of Information over {Red, Orange, Green, Blue}

In Fig. 1 we present a Hasse diagram of a sublattice of the lattice of information containing the points representing the information provided by the functions above, and visualising the equivalence relations by representing them as partitions. Note that with the partition view, the ordering relation is partition refinement. The full lattice LoI{Red, Orange, Green, Blue} contains 15 elements (known in combinatorics as the 4^{th} *Bell number*).

2.1 The Information Ordering on Functions

To understand the formal connection between the functions and the corresponding information that they release, we use the well-known notion of the *kernel* of a function: We recall that the *kernel* of a function $f : D \to E$ is the equivalence relation $\ker(f) \in \mathrm{LoI}(D)$ which relates all elements mapped by f to the same result: $a \ \ker(f)\ b$ iff $f(a) = f(b)$.

Thus the points illustrated in the lattice do indeed correspond to the respective kernels of the functions, and it can readily be seen that $\ker(\text{primary}) = \ker(\text{isPrimary}) \sqcup_{\mathrm{LoI}} \ker(\text{isTrafficLight})$.

Note that taking kernels induces an information preorder on any functions f and g which have a common input domain (we write $\mathrm{dom}(f) = \mathrm{dom}(g)$), namely $f \precsim g$ iff $\ker(f) \sqsubseteq_{\mathrm{LoI}} \ker(g)$, i.e. g reveals at least as much information about its argument as f.

Note that this information ordering between functions can be characterised in a number of ways.

Proposition 1. *For any functions f and g such that $\mathrm{dom}(f) = \mathrm{dom}(g)$ the following are equivalent:*

(1) $f \precsim g$
(2) $\{p \circ f \mid \mathrm{codom}(f) = \mathrm{dom}(p)\} \subseteq \{p \circ g \mid \mathrm{codom}(g) = \mathrm{dom}(p)\}$ *(where $\mathrm{codom}(f)$ is the codomain of function f)*
(3) *There exists p such that $f = p \circ g$*

The proposition essentially highlights the fact that the information ordering on functions can alternatively be understood in terms of postprocessing (the function p). The set $\{p \circ f \mid \mathrm{codom}(f) = \mathrm{dom}(p)\}$ can be viewed as all the things which can be computed from the result of applying f.

2.2 An Epistemic View

In our refinement of the lattice of information we will lean on an *epistemic* characterisation of the function ordering which focuses on the facts which an observer of the output of a function might learn about its input.

Definition 2. *For $f : A \to B$ and $a \in A$, define the* f-knowledge set *for a as:*

$$K_f(a) = \{a' \in A \mid f(a) = f(a')\}$$

The knowledge set for an input a is thus what an observer who knows the function f can maximally deduce about the input if they only get to observe the result, $f(a)$. For example, $K_{\text{primary}}(\text{Green}) = \{c \mid \text{primary}(c) = \text{primary}(\text{Green})\} = \{\text{Green}, \text{Orange}\}$. Note that although we use the terminology "knowledge", following work on the semantics of dynamic security policies [Askarov and Chong 2012; Askarov and Sabelfeld 2007], it is perhaps more correct to think of this as *uncertainty* in the sense that a smaller set corresponds to a more precise deduction. The point here is that $f \precsim g$ can be characterised in terms of knowledge sets: g will produce knowledge sets which are at least as precise as those of f:

Proposition 2. *Let f and g be any two functions with domain A. Then $f \precsim g$ iff $K_g(a) \subseteq K_f(a)$ for all $a \in A$.*

2.3 Information Flow and Generalised Kernels

Although we can understand the information released by a function by considering its kernel as an element of the lattice of information, for various reasons it is useful to generalise this idea. The first reason is that we are often interested in understanding the information flow through a function when just a part of the function's output is observed. For example, if we want to know whether a function is secure, this may require verifying that the public parts of the output reveal information about at most the non-secret inputs. The second reason to generalise the way we think about information flow of functions is to build compositional reasoning principles. Suppose that we know

that a function f reveals information P about its input. Now suppose that we wish to reason about $f \circ g$. In order to make use of what we know about f we need to understand the information flow of g when the output is "observed" through P. This motivates the following generalised information flow definition (the specific notation here is taken from [Hunt 1991; Sabelfeld and Sands 2001], but we state it for arbitrary binary relations à la logical relations [Reynolds 1983]):

Definition 3. *Let P and Q be binary relations on sets A and B, respectively. Let $f : A \to B$. Define:*

$$f : P \Rightarrow Q \;\text{ iff }\; \forall a, a'.(a\ P\ a' \quad \textit{implies} \quad f(a)\ Q\ f(a'))$$

When P and Q are equivalence relations, these definitions describe information flow properties of f where P describes an upper bound on what can be learned about the input when observing the output "through" Q (i.e. we cannot distinguish Q-related outputs).

We can read $f : P \Rightarrow Q$ as an information flow typing at the semantic level. As such it can be seen to enjoy natural composition and subtyping properties. Again, we state these in a more general form as we will reuse them for different kinds of relation:

Fact 1. *The following inference rules are valid for all functions and binary relations of appropriate type:*

$$\frac{P' \subseteq P \quad f : P \Rightarrow Q \quad Q \subseteq Q'}{f : P' \Rightarrow Q'}\ \text{Sub} \qquad \frac{f : P \Rightarrow Q \quad g : Q \Rightarrow R}{g \circ f : P \Rightarrow R}\ \text{Comp}$$

When these relations are elements of the lattice of information, the conditions $P' \subseteq P$ and $Q \subseteq Q'$ in the Sub-rule amount to $P' \sqsupseteq_{\text{LoI}} P$ and $Q \sqsupseteq_{\text{LoI}} Q'$, respectively.

Information flow properties also satisfy weakest precondition and strongest postcondition-like properties. To present these, we start by generalising the notion of kernel of a function:

Definition 4 (Generalised Kernel). *Let $\text{REL}(A)$ denote the set of all binary relations on a set A. For any $f : A \to B$, define $f^* : \text{REL}(B) \to \text{REL}(A)$ as follows:*

$$x\ f^*(R)\ y \;\text{ iff }\; f(x)\ R\ f(y)$$

We call this the generalised kernel map, *since* $\ker(f) = f^*(\text{Id})$.

Now, it is evident that f^* preserves reflexivity, transitivity and symmetry, so restricting f^* to equivalence relations immediately yields a well defined map in LoI (Landauer and Redmond [1993] use the notation $f\#$ for this map). Moreover, we can define a partner $f_!$, which operates in the opposite direction and has dual properties (as formalised below):

Definition 5. *For $f : A \to B$:*

(1) $f^* : \text{LoI}(B) \to \text{LoI}(A)$ *is the restriction of the generalised kernel map to* $\text{LoI}(B)$.
(2) $f_! : \text{LoI}(A) \to \text{LoI}(B)$ *is given by* $f_!(P) = \bigsqcup_{LoI}\{Q \in \text{LoI} \mid f : P \Rightarrow Q\}$.

Note that we are overloading our notation here, using f^* for both the map on REL and its restriction to LoI. Later, in §3.6, we overload it again (along with $f_!$). Our justification for this overloading is that in each case these maps are doing essentially the same thing[3]: $f^*(Q)$ is the *weakest precondition* for Q (i.e. the smallest P such that $f : P \Rightarrow Q$) while $f_!(P)$ is the *strongest postcondition* of P (i.e. the largest Q such that $f : P \Rightarrow Q$), where "smallest" and "largest" are interpreted within the relevant lattice (LoI here, our refined lattice LoCI later). The following proposition formalises this for LoI (see Proposition 6 for its LoCI counterpart):

[3]This can be made precise, categorically. See §3.7.

Proposition 3. *For any $f : A \to B$, f^* and $f_!$ are monotone and, for any $P \in \mathrm{LoI}(A)$ and $Q \in \mathrm{LoI}(B)$, the following are all equivalent:*

$$(1)\ f : P \Rightarrow Q \qquad (2)\ f^*(Q) \sqsubseteq_{\mathrm{LoI}} P \qquad (3)\ Q \sqsubseteq_{\mathrm{LoI}} f_!(P)$$

We have summarised a range of key properties of the lattice of information that make it useful for both formulating a wide variety of information flow properties, as well as proving them in a compositional way. An important goal in refining the lattice of information will be to ensure that we still enjoy properties of the same kind.

3 LOCI: THE LATTICE OF COMPUTABLE INFORMATION

Our goal in this section is to introduce a refinement of noninterference which accounts for the difference in quality of knowledge that arises from nontermination, or more generally partiality, for example when programs produce output streams that may at some point fail to progress. We will assume that a program is modelled in a domain-theoretic denotational style, as a continuous function between partially ordered sets. In this setting, the order relation on a set of values models their relative degrees of "definedness". Simple nontermination is modelled as a bottom element, $\bot$, and in general the ordering relation reflects the evolution of computation. Following Scott, the pioneer of this approach, when one element d is dominated by another e, one can think of e as containing *more information* than d. In the domain-theoretic view, a partial element is not a concrete observation or outcome, but a degree of knowledge about a computation. In this sense $\bot$ represents no knowledge – you do not fully observe a nonterminating computation, it may still evolve into some more defined result. Note how this view emphasises how we are abstracting away from time. This also explains the basic requirement that all functions (which will be the denotation of programs) are monotone: if you know more about the input (in Scott's sense) you know more about the output. In domain theory (a standard reference is [Abramsky and Jung 1995]) one restricts attention to some subclass of well-behaved partially ordered sets (the *domains* of the theory), in order that recursive computations may be given denotations as least fixpoints. Being well-behaved in this context entails the existence of suprema of directed sets (and usually a requirement that the domain has a finitary presentation in terms of its compact elements). In this paper we keep our key definitions as general as possible by stating them for arbitrary partially ordered sets, but still requiring that the functions under study are continuous (preserve directed suprema when they exist). We expect that some avenues for future work may require additional structure to be imposed (see §6).

3.1 Order-Theoretic Preliminaries

A *partial order* on a set A is a reflexive, transitive and antisymmetric relation on A. A *poset* is a pair $(A, \sqsubseteq_A)$ where $\sqsubseteq$ is a partial order on A. We typically elide the subscript on $\sqsubseteq_A$ when A is clear from the context.

The *supremum* of a subset $X \subseteq A$, if it exists, is the least upper bound $\bigsqcup X$ with respect to $\sqsubseteq_A$.

A set $X \subseteq A$ is *directed* if X is non-empty and, for all $x_1 \in X, x_2 \in X$, there exists $x' \in X$ such that $x_1 \sqsubseteq x'$ and $x_2 \sqsubseteq x'$. For posets A and B, a function $f : A \to B$ is *monotone* iff $a \sqsubseteq a'$ implies $f(a) \sqsubseteq f(a')$. A function $f : A \to B$ is *Scott-continuous* if, for all X directed in A, whenever $\bigsqcup X$ exists in A then $\bigsqcup f(X)$ exists in B and is equal to $f(\bigsqcup X)$. From now on we will simply say continuous when we mean Scott-continuous. Note:

(1) Continuity implies monotonicity because $a \sqsubseteq a'$ implies both that $\{a, a'\}$ is directed and that $\bigsqcup\{a, a'\} = a'$, while $\bigsqcup\{f(a), f(a')\} = f(a')$ implies $f(a) \sqsubseteq f(a')$.
(2) Monotonicity in turn implies that, if X is directed in A, then $f(X)$ is directed in B.

Notation. *In what follows, we write $f \in [A \to B]$ as a shorthand to mean both that A and B are posets and that f is continuous.*

3.2 Ordered Knowledge Sets

Our starting point is the epistemic view presented in §2.2. Recall that we defined the f-knowledge set for an input a to be the set $\{a' \mid f(a') = f(a)\}$, which is what we learn by observing the output of f when the input is a. However, as discussed in the introduction to this section, in a domain-theoretic setting, observation of a *partial* output should be understood as provisional: there may be more to come. This requires us to modify the definition of knowledge set accordingly. What we learn about the input when we see a partial output is that the input could be anything which produces that observation, *or something greater*. Hence:

Definition 6. *For $f \in [A \to B]$ and $a \in A$, define the* ordered f-knowledge set *for a as:*

$$K_f^{\sqsubseteq}(a) = \{a' \in A \mid f(a) \sqsubseteq f(a')\}$$

Recall (Proposition 2) that the LoI preorder on functions has an alternative characterisation in terms of knowledge sets: the kernel of g is a refinement of (i.e. more discriminating than) the kernel of f just when each knowledge set of g is a subset of (i.e. more precise than) the corresponding knowledge set of f. Unsurprisingly however, if we compare continuous functions based on their *ordered* knowledge sets, the correspondence with LoI is lost. Consider the examples `parity0` and `parity1` from §1.2. We can model these as functions $f_0, f_1 \in [Z \to Z_\perp]$, where Z is discretely ordered (the partial order is just equality) and the lifting $Z_\perp$ adds a new element $\perp$ which is $\sqsubseteq$ everything. We have:

$$f_0(x) = \begin{cases} 1 & \text{if } x \text{ is even} \\ \perp & \text{if } x \text{ is odd} \end{cases} \qquad \text{and} \qquad f_1(x) = \begin{cases} 1 & \text{if } x \text{ is even} \\ 0 & \text{if } x \text{ is odd} \end{cases}$$

As discussed previously, these two functions have the same kernel and so are LoI-equivalent. Moreover, in accordance with Proposition 2, it is easy to see that they induce the same knowledge sets: $K_{f_0}(x) = K_{f_1}(x)$ for all $x \in Z$. However, they do not induce the same *ordered* knowledge sets. In particular, when x is odd we have $K_{f_1}^{\sqsubseteq}(x) = \{y \in Z \mid y \text{ is odd}\}$ but $K_{f_0}^{\sqsubseteq}(x) = Z$. In fact, not only do the two functions induce different ordered knowledge sets, but f_1 is (strictly) more informative than f_0, since $K_{f_1}^{\sqsubseteq}(x) \subseteq K_{f_0}^{\sqsubseteq}(x)$ for all x (and $K_{f_1}^{\sqsubseteq}(x) \neq K_{f_0}^{\sqsubseteq}(x)$ for some x).

Our key insight is that it is possible to define an alternative information lattice, one which corresponds exactly with ordered knowledge sets, by using (a certain class of) preorders, in place of the equivalence relations used in LoI.

3.3 Ordered Kernels

A *preorder* is simply a reflexive and transitive binary relation. Clearly, every equivalence relation is a preorder, but not every preorder is an equivalence relation. As with equivalence relations, it is possible to present a preorder in an alternative form, as a partition rather than a binary relation, but with one additional piece of information: a partial order on the blocks of the partition. In fact, there is a straightforward 1-1 correspondence between preorders and partially ordered partitions:

(1) Given a preorder Q on a set A, for each $a \in A$, define $[a]_Q = \{a' \mid a\ Q\ a' \wedge a'\ Q\ a\}$ and $[Q] = \{[a]_Q \mid a \in A\}$. (Note: although we appear to be overloading the notation introduced in §2, the definitions agree in the case that Q is an equivalence relation.)
Then define $[a_1]_Q \sqsubseteq_Q [a_2]_Q$ iff $a_1\ Q\ a_2$. This is a well-defined partial order on $[Q]$.

(2) Conversely, given a poset $(\Phi, \sqsubseteq)$, where Φ is a partition of set A, we recover the corresponding preorder on A by defining $a\ Q\ a'$ iff $[a]_\Phi \sqsubseteq [a']_\Phi$.

For a preorder Q, we refer to the equivalence relation with equivalence classes $[Q]$ as the *underlying* equivalence relation of Q. Clearly, the underlying equivalence relation of Q is just $Q \cap Q^{-1}$.

Taking the same path for kernels that we took from unordered to ordered knowledge sets, we arrive at the following definition:

Definition 7. *Let $(B, \sqsubseteq)$ be a poset. Given $f \in [A \to B]$, define its* ordered kernel $\ker_\sqsubseteq(f)$ *to be $f^*(\sqsubseteq)$, thus $x \ \ker_\sqsubseteq(f)\ y$ iff $f(x) \sqsubseteq f(y)$.*

Proposition 4. *$\ker_\sqsubseteq(f)$ is a preorder, and its underlying equivalence relation is $\ker(f)$.*

Only some preorders are the ordered kernels of continuous functions. For example, if $a \sqsubseteq a'$ and Q is the ordered kernel of some continuous f, then it must be the case that $a \ Q \ a'$, since $a \sqsubseteq a'$ implies $f(a) \sqsubseteq f(a')$.

Definition 8 (Complete Preorder). *Let A be a poset and let Q be a preorder on A. We say that Q is* complete *iff, whenever X is directed in A and $\bigsqcup X$ exists:*

(1) $\forall x \in X.\ x \ Q \ (\bigsqcup X)$
(2) $\forall a \in A.\ (\forall x \in X.\ x \ Q \ a)$ *implies* $(\bigsqcup X) \ Q \ a$

Note that part (1) entails that every complete Q contains the domain ordering $(\sqsubseteq)$.

It is perhaps more illuminating to see the definition of completeness for Q presented in terms of its corresponding partially ordered partition:

Lemma 1. *Let A be a poset and let Q be a preorder on A. Then Q is complete iff, whenever X is directed in A and $\bigsqcup X$ exists in A, $\bigsqcup \{[x]_Q \mid x \in X\}$ exists in $([Q], \sqsubseteq_Q)$ and is equal to $[\bigsqcup X]_Q$.*
In other words, Q is complete iff the quotient map $(\lambda a.[a]_Q) : A \to ([Q], \sqsubseteq_Q)$ is continuous.

To round off this section, we establish that the complete preorders on a poset are just the ordered kernels of all the continuous functions with that domain:

Theorem 1. *Let A be a poset. Then Q is a complete preorder on A iff there is some poset B and $f \in [A \to B]$ such that $Q = \ker_\sqsubseteq(f)$.*

Proof. The implication from left to right is established by Lemma 1.

For the implication right to left, assume f is continuous and let $Q = \ker_\sqsubseteq(f)$. Let X be directed in A such that $\bigsqcup X$ exists. Then:

(1) Let $x \in X$. Since f is monotone, $f(x) \sqsubseteq f(\bigsqcup X)$, thus $x \ Q \ (\bigsqcup X)$.
(2) Let $a \in A$ be such that $x \ Q \ a$ for all $x \in X$. Then $f(x) \sqsubseteq f(a)$ for all $x \in X$, hence $(\bigsqcup f(X)) \sqsubseteq f(a)$, hence $f(\bigsqcup X) \sqsubseteq f(a)$. Thus $(\bigsqcup X) \ Q \ a$.

□

3.4 LoCI

We now define the lattice of computable information as a lattice of complete preorders, directly analogous to the definition of LoI as a lattice of equivalence relations. In particular, we can rely on the fact that the complete preorders are closed under intersection:

Lemma 2. *Let $\{Q_i\}$ be an arbitrary family of complete preorders. Then $\bigcap Q_i$ is a complete preorder.*

Definition 9 (Lattice of Computable Information). *For a poset A, the lattice of information over A,* $\mathrm{LoCI}(A)$*, is defined to be the lattice*

$$\mathrm{LoCI}(A) = (\mathrm{PRE}(A), \sqsubseteq_{LoCI}, \sqcup_{LoCI})$$

where $\mathrm{PRE}(A)$ *is the set of all complete preorders on A, $P \sqsubseteq_{LoCI} Q \stackrel{\text{def}}{=} Q \subseteq P$, and $\bigsqcup_{LoCI} \stackrel{\text{def}}{=} \bigcap$*

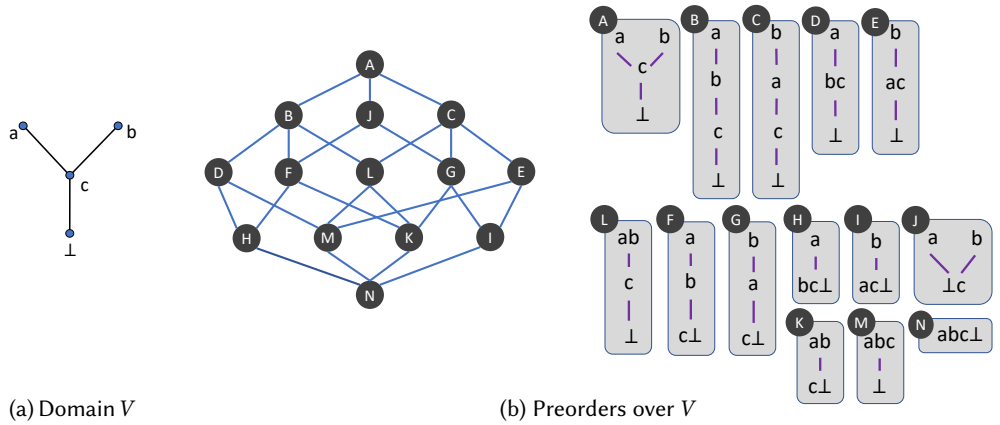
Since LoCI(A) has all joins (not just the binary ones), with the bottom element given by $\mathrm{All}_A = A \times A$, and top element $\sqsubseteq_A$, it also has all meets, and hence is a *complete* lattice. Meets are not used in what follows so we do not dwell on them further here.

As for LoI, we can define a preorder on (continuous) functions based on their ordered kernels: $f \precsim_{\mathrm{LoCI}} g$ iff $\ker_{\sqsubseteq}(f) \sqsubseteq_{\mathrm{LoCI}} \ker_{\sqsubseteq}(g)$. As claimed above, this corresponds exactly to an ordering of continuous functions based on their ordered knowledge sets:

Proposition 5. *Let A be a poset and let f and g be any two continuous functions with domain A. Then $f \precsim_{LoCI} g$ iff $K_g^{\sqsubseteq}(a) \subseteq K_f^{\sqsubseteq}(a)$ for all $a \in A$.*

3.5 An Example LoCI

In this section we describe LoCI(V) for the simple four-point domain V shown in Fig. 2a. A Hasse diagram of the lattice structure is shown in Fig. 2b. On the right we enumerate all the complete preorders on V, presented as partially ordered partitions. Note that we write a to mean the singleton block $\{a\}$, and $ac\bot$ to mean $\{a, c, \bot\}$, etc.



(a) Domain V (b) Preorders over V

Fig. 2. LoCI(V)

Let us now consider two continuous functions whose ordered kernels are presented here, $f_1, f_2 \in [V \to V]$ where:

$$f_1 = \lambda x.a \qquad f_2 = \lambda x. \begin{cases} a & \text{if } x = a \\ c & \text{if } x \in \{b, c\} \\ \bot & \text{if } x = \bot \end{cases}$$

Since f_1 is a constant function it conveys no information about its input, so its ordered kernel is the least element, N (= All). The ordered kernel of f_2 is D: when the input is a, the observer learns this exactly; when the input is b or c, the observer learns only that the input belongs to $\{a, b, c\}$; when the input is $\bot$, the observer (inevitably) learns nothing at all. Thus, in the LoCI ordering, f_2 is strictly more informative than f_1. It is interesting to note by contrast, that in the Scott-ordering on functions, f_1 is *maximal*, and strictly more defined than f_2 (recall that $f \sqsubseteq g$ in the Scott-order iff $f(x) \sqsubseteq g(x)$ for all x). In general, the Scott-ordering between functions tells us little or nothing about their relative capacity to convey information about their inputs. This can be viewed as an instance of the *refinement problem* known from secure information flow [McLean 1994], where a point in a domain can be viewed as its upper set (all its possible "futures") and a higher point is then a refinement (a smaller set of futures).

3.6 Information Flow Properties in LoCI

We can directly use the notation $f : P \Rightarrow Q$ introduced earlier to express information flow properties for P and Q in LoCI. Since the ordering on relations in LoCI is still reversed set containment, both the "subtyping" and composition properties stated previously (Fact 1) hold equally well for LoCI as for LoI. And, as promised, we also have weakest precondition and strongest postcondition properties, provided by appropriate versions of f^* and $f_!$ for continuous f and complete preorders:

Definition 10. *For $f \in [A \to B]$:*

(1) $f^* : \mathrm{LoCI}(B) \to \mathrm{LoCI}(A)$ *is the restriction of the generalised kernel map to* $\mathrm{LoCI}(B)$.
(2) $f_! : \mathrm{LoCI}(A) \to \mathrm{LoCI}(B)$ *is given by* $f_!(P) = \bigsqcup_{LoCI}\{Q \in \mathrm{LoCI} \mid f : P \Rightarrow Q\}$.

(Well-definedness of $f^* : \mathrm{LoCI}(B) \to \mathrm{LoCI}(A)$ is slightly less immediate than for the LoI variant, but the key requirement is to show that $f^*(Q)$ is complete and this follows easily using continuity of f.) The LoCI analogue of Proposition 3 is then:

Proposition 6. *For any $f \in [A \to B]$, f^* and $f_!$ are monotone and, for any $P \in \mathrm{LoCI}(A)$ and $Q \in \mathrm{LoCI}(B)$, the following are all equivalent:*

$$(1)\ f : P \Rightarrow Q \qquad (2)\ f^*(Q) \sqsubseteq_{LoCI} P \qquad (3)\ Q \sqsubseteq_{LoCI} f_!(P)$$

3.7 A Category of Computable Information

Some of the definitions and properties introduced earlier can be recast in category-theoretic terms through the framework of *Grothendieck fibrations*. In this subsection, we briefly sketch the relevant connections. The subsection is intended as an outline for interested readers rather than a definitive category-theoretic treatment of LoCI – which is beyond the scope of this paper. The remainder of the paper does not depend on any of the ideas discussed in this subsection, but some notational choices and technical developments are inspired by it.

So far we have treated posets A, B and continuous functions $f : A \to B$ as a semantic framework, in which we have studied, separately, the information associated with individual domains A via $\mathrm{LoCI}(A)$, and the flow of information over a channel f via $f : - \Rightarrow -$. An alternative approach is to combine the information represented by a preorder $P \in \mathrm{LoCI}(A)$ and its underlying poset A into a single mathematical structure, and to study the overall properties of such *information domains*.

Definition 11. *An* information domain *is a pair (A, P) consisting of a poset A and a complete preorder $P \in \mathrm{LoCI}(A)$. An* information-sensitive function *between information domains (A, P) and (B, Q) is a continuous function $f : A \to B$, such that $f : P \Rightarrow Q$.*

Information domains and information-sensitive functions form the *category of computable information* **CoCI**. Identities and composition are defined via the underlying continuous maps; composition preserves information-sensitivity by Fact 1 (Comp).

The category **CoCI** and the family of lattices $\mathrm{LoCI}(A)$ are related by a *fibration* or, to use the terminology coined by Melliès and Zeilberger [2015], by a *type refinement system*. Intuitively, we may think of a poset A as a *type*, and of an information domain (A, P) as a *refinement* of A. For each type A, there is a subcategory of **CoCI**, called the *fibre over* A, whose objects are the refinements of A, and which is equivalent to $\mathrm{LoCI}(A)$.

Formally, there is a forgetful functor U from **CoCI** to the category **PC** of posets and continuous functions that maps refinements to their types $U(A, P) = A$ and information-sensitive functions to the underlying continuous maps $U(f) = f$. The fibre $\mathbf{CoCI}_A$ over A is the "inverse image" of A under U, i.e. the subcategory of **CoCI** with objects of the form (A, P) and arrows of the form $\mathrm{id}_A : (A, P) \to (A, Q)$, where $P, Q \in \mathrm{LoCI}(A)$. Note that the objects of $\mathbf{CoCI}_A$ are uniquely determined by their second component, and that there is an arrow between the pair of objects (A, P)

and (A, Q) iff $P \sqsupseteq_{\mathrm{LoCI}} Q$. In other words, the category $\mathbf{CoCI}_A$ is equivalent to the dual lattice of $\mathrm{LoCI}(A)$, thought of as a complete (and co-complete) posetal category. In line with the terminology of Melliès and Zeilberger, we may call $\mathbf{CoCI}_A$ the *subtyping* lattice over A.

Furthermore, the functor U is a *bifibration*. Intuitively, this ensures that we can reindex refinements along continuous maps. The formal definition of a bifibration is somewhat involved [see e.g. Melliès and Zeilberger 2015], but it can be shown, in our setting, to correspond to the existence of weakest preconditions and strongest postconditions as characterised in Proposition 6, plus the following identities

$$\mathrm{id}_A^* = \mathrm{id}_{\mathrm{LoCI}(A)} \qquad (g \circ f)^* = f^* \circ g^* \qquad (\mathrm{id}_A)_! = \mathrm{id}_{\mathrm{LoCI}(A)} \qquad (g \circ f)_! = g_! \circ f_!$$

which are easy to prove. For the last one, rather than showing $(g \circ f)_! = g_! \circ f_!$ directly – which is awkward – it is simpler to show $(g \circ f)^* = f^* \circ g^*$ first, and then use the fact that each $(h^*, h_!)$ is an adjoint pair. The *cartesian* and *opcartesian liftings* of $f : A \to B$ to (B, Q) and (A, P) are then given by $f : (A, f^*(Q)) \to (B, Q)$ and $f : (A, P) \to (B, f_!(P))$, respectively.

Using the reindexing maps f^* and $f_!$, we can extend the poset-indexed set $\{\mathbf{CoCI}_A\}_{A \in \mathbf{PC}}$ of fibres over A into a *poset-indexed category*, that is, a contravariant functor $F : \mathbf{PC}^{\mathrm{op}} \to \mathbf{Cat}$, that maps posets A to fibres $F(A) = \mathbf{CoCI}_A$ and whose action on continuous maps $f : A \to B$ is given by

$$F(f) : \mathbf{CoCI}_B \to \mathbf{CoCI}_A$$
$$F(f)(Q) = f^*(Q)$$

Replacing the reindexing map f^* with $f_!$, we obtain a similar, covariant functor $G : \mathbf{PC} \to \mathbf{Cat}$.[4]

The family of lattices $\mathrm{LoCI}(A)$ and the category $\mathbf{CoCI}$ fully determine each other: we may obtain $\mathrm{LoCI}(A)$ as the fibres of $\mathbf{CoCI}$ via U, and conversely, we may reconstruct the category $\mathbf{CoCI}$ from the indexed category F via the Grothendieck construction $\mathbf{CoCI} = \int F$.

Finally, note that the above can also be adapted to the simpler setting of LoI. In that case, types are simply sets, and refinements are *setoids*, i.e. pairs (S, R) consisting of a set S and an equivalence relation $R \in \mathrm{LoI}(S)$. The relevant fibration is the obvious forgetful functor $U : \mathbf{Setoid} \to \mathbf{Set}$ from the category of setoids and equivalence-preserving maps to the underlying sets and total functions.

3.8 A Partial Embedding of LoI into LoCI

As discussed earlier, a key advantage of LoCI in comparison to LoI is that it distinguishes between functions which have the same (unordered) kernel but which differ fundamentally in what information they actually make available to an output observer, due to different degrees of partiality.

But there is another advantage of LoCI: it excludes "uncomputable" kernels, those equivalence relations in LoI which are not the kernel of *any* continuous function. Consider the example of $\mathrm{LoCI}(V)$ in Fig. 2b. Since V has four elements, there are 15 distinct equivalence relations in $\mathrm{LoI}(V)$. Note, however, that $\mathrm{LoCI}(V)$ has only 14 elements. Clearly then there must be at least one equivalence relation which is being excluded by $\mathrm{LoCI}(V)$ (in fact, five elements of $\mathrm{LoI}(V)$ are excluded). Let us settle on some terminology for this:

Definition 12. *Let A be a poset. Let R be an equivalence relation on A and let Q be a complete preorder on A. Say that Q* realises *R if R is the underlying equivalence relation of Q. When such Q exists for a given R, we say that R is* realisable.

Note that, by Proposition 4, the underlying equivalence relation of $\ker_{\sqsubseteq}(f)$ is $\ker(f)$, so by Theorem 1 it is equivalent to say that R is realisable iff R is the kernel of some continuous function.

In $\mathrm{LoCI}(V)$, note that A, B and C all realise the identity relation. Similarly, F, G and J all realise the same equivalence relation as each other. Thus, while $\mathrm{LoCI}(V)$ has 14 elements, together they realise

[4] The existence of F and G is in fact sufficient to establish that U is a bifibration.

only 10 of the 15 possible equivalence relations over V. As an example of a missing equivalence relation, consider the one with equivalence classes $\{a, b, \bot\}, \{c\}$. Recall that a subset X of a poset is *convex* iff, whenever $x \sqsubseteq y \sqsubseteq z$ and $x, z \in X$, then $y \in X$. Note that $\{a, b, \bot\}$ is not convex, but it is easy to see that all equivalence classes in the kernel of a monotone function *must* be convex. (The convexity test also fails for the four other missing equivalence relations. But convexity alone is not sufficient for realisability, even in the finite case. See §3.8.1 below.)

When an equivalence relation S *is* realisable, we can show that there must be a *greatest* element of LoCI which realises it. Moreover, we can use this realiser to re-express an LoI property $f : R \Rightarrow S$ as an equivalent LoCI property. To this end, we define a pair of monotone maps which allow us to move back and forth between LoI and LoCI:

Definition 13. *For poset A define* $\mathrm{Cp}_A : \mathrm{LoI}(A) \to \mathrm{LoCI}(A)$ *and* $\mathrm{Er}_A : \mathrm{LoCI}(A) \to \mathrm{LoI}(A)$ *by:*

(1) $\mathrm{Cp}_A(R) = \bigsqcup_{LoCI}\{P \in \mathrm{LoCI}(A) \mid P \supseteq R\} = \bigcap\{P \in \mathrm{LoCI}(A) \mid P \supseteq R\}$

(2) $\mathrm{Er}_A(P)$ *is the underlying equivalence relation of* P*:* $\mathrm{Er}_A(P) = P \cap P^{-1}$

It is easy to see that both maps are monotone. We will routinely omit the subscripts on Cp *and* Er *in contexts where the intended domain is clear.*

Note that, by definition, $R \in \mathrm{LoI}(A)$ is realisable iff there exists some $P \in \mathrm{LoCI}(A)$ such that $\mathrm{Er}(P) = R$. Now, $\mathrm{Cp}(R)$ is defined above to be the greatest $P \in \mathrm{LoCI}(A)$ such that $P \supseteq R$. But observe that $\mathrm{Er}(P) \sqsubseteq_{\mathrm{LoCI}} R$ iff $\mathrm{Er}(P) \supseteq R$, and $\mathrm{Er}(P) = P \cap P^{-1} \supseteq R$ iff $P \supseteq R$, since R is symmetric. So we have actually defined $\mathrm{Cp}(R)$ to be the greatest $P \in \mathrm{LoCI}(A)$ such that $\mathrm{Er}(P) \sqsubseteq_{\mathrm{LoCI}} R$. The following propositions are immediate consequences:

Proposition 7. *R is realisable iff* $\mathrm{Er}(\mathrm{Cp}(R)) = R$ *(in which case* $\mathrm{Cp}(R)$ *is its greatest realiser).*

Proposition 8. *The pair* $(\mathrm{Er}_A, \mathrm{Cp}_A)$ *forms a* Galois connection *between* LoCI *(A) and* LoI *(A). That is to say for every* $P \in \mathrm{LoCI}(A)$ *and every* $R \in \mathrm{LoI}(A)$*:*

$$\mathrm{Er}(P) \sqsubseteq_{\mathrm{LoI}} R \;\text{ iff }\; P \sqsubseteq_{LoCI} \mathrm{Cp}(R) \tag{GC}$$

(See [Erné et al. 1993] for an introduction to Galois connections.)

This extends to an encoding of LoI properties as LoCI properties:

Theorem 2. *For all* $f \in [A \to B]$*, for all* $R \in LoI(A)$*, for all* $Q \in \mathrm{LoCI}(B)$*:*

$$f : R \Rightarrow \mathrm{Er}(Q) \;\text{ iff }\; f : \mathrm{Cp}(R) \Rightarrow Q$$

Proof. By Propositions 3 and 6, it suffices to show $f^*(\mathrm{Er}(Q)) \sqsubseteq_{\mathrm{LoI}} R$ iff $f^*(Q) \sqsubseteq_{\mathrm{LoCI}} \mathrm{Cp}(R)$. First we note that the following holds by an easy unwinding of the definitions:

$$f^*(\mathrm{Er}_B(Q)) = \mathrm{Er}_A(f^*(Q)) \tag{$*$}$$

Then we have:

$$f^*(\mathrm{Er}(Q)) \sqsubseteq_{\mathrm{LoI}} R \;\text{ iff }\; \mathrm{Er}(f^*(Q)) \sqsubseteq_{\mathrm{LoI}} R \;\text{ iff }\; f^*(Q) \sqsubseteq_{\mathrm{LoCI}} \mathrm{Cp}(R)$$

where the first equivalence holds by ($*$) and the second by (GC). □

Corollary 1. *If S is realisable then* $f : R \Rightarrow S$ iff $f : \mathrm{Cp}(R) \Rightarrow \mathrm{Cp}(S)$.

Proof. By Proposition 7, S is realisable iff $S = \mathrm{Er}(\mathrm{Cp}(S))$, so let $Q = \mathrm{Cp}(S)$ in the theorem. □

It is interesting to note that Corollary 1 does not require R to be realisable. However, in general, the equivalence does not hold unless S is realisable. For a counterexample, consider the three-point lattice $A = \{0, 1, 2\}$ with $0 \sqsubset 1 \sqsubset 2$, and let S be the equivalence relation with equivalence classes $\{0, 2\}$ and $\{1\}$. The first of these classes is not convex, so S is not realisable. Now consider the

property $f : \mathrm{All} \Rightarrow S$. For continuous $f : A \to B$, it is easy to see that this property holds iff f is constant. However, $\mathrm{Cp}(S) = \mathrm{Cp}(\mathrm{All}) = \mathrm{All}$ and $f : \mathrm{All} \Rightarrow \mathrm{All}$ holds trivially. (Of course, this is no great loss if the property of interest is actually constancy. The appropriate choice of S in this case is $S = \mathrm{Id}$, which is realisable.)

3.8.1 Verifying Realisability. We describe a simple necessary condition for realisability, which is also sufficient in the finite case. It is motivated by the following example. Let R be the equivalence relation shown in Fig. 3. The three equivalence classes are clearly convex, but R is not realisable. To see why, suppose that R is the kernel of f. There must be distinct elements x and y such that $f(\{a, b'\}) = \{x\}$ and $f(\{b, a'\}) = \{y\}$. If f is monotone then, since $a \sqsubseteq a'$ and $b \sqsubseteq b'$, it must be the case that $x \sqsubseteq y \sqsubseteq x$, which contradicts the assumption that x and y are distinct.

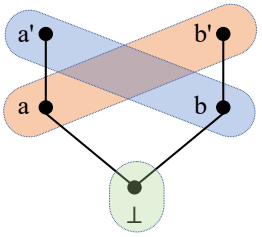


Fig. 3. An Unrealisable Equivalence Relation

The example of Fig. 3 generalises quite directly. Given any equivalence relation R on a poset A, define ϕ as the relation on $[R]$ which relates two equivalence classes whenever they contain ($\sqsubseteq$)-related elements: $[a]_R \ \phi \ [b]_R$ iff $\exists x \in [a]_R . \exists y \in [b]_R . x \sqsubseteq y$. In Fig. 3, unrealisability manifests as a non-trivial cycle in the graph of ϕ, that is, a sequence $[a_1]_R \ \phi \cdots \phi \ [a_n]_R \ \phi \ [a_1]_R$ with $n > 1$ and such that all $[a_i]_R$ are distinct. By the obvious inductive generalisation of the above argument, any monotone f necessarily maps all a_i to the same value, thus making $\ker(f) = R$ impossible. So if the graph of ϕ contains a non-trivial cycle, R is not realisable. (Note also that this generalises the convexity condition: if any $[a]_R$ is non-convex, there will be a non-trivial cycle with $n = 2$.)

Conversely, to say that ϕ is free of such cycles is just to say that the transitive closure ϕ^+ is antisymmetric. Clearly, ϕ^+ is also reflexive and transitive, thus $B = ([R], \phi^+)$ is a poset. Let $f : A \to B$ be the map $a \mapsto [a]_R$. Then f is monotone (because $x \sqsubseteq y$ implies $[x]_R \ \phi \ [y]_R$) and $\ker(f) = R$. In the case that $\sqsubseteq_A$ is of finite height, this establishes that R is realisable.

3.9 Post Processing

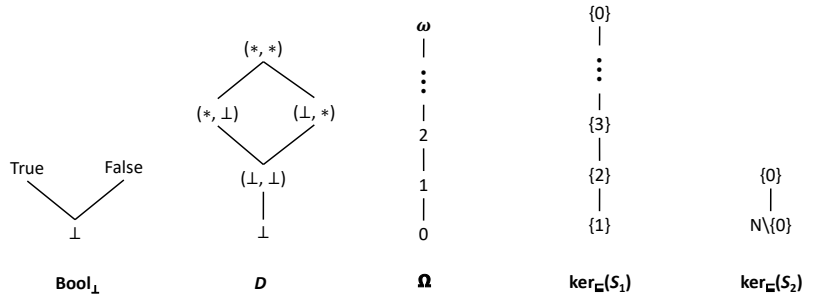
In §2 we introduced three equivalent ways of ordering functions, the first based on inclusion of their kernels ($\precsim$), the second in terms of their inter-definability via postprocessing (Proposition 1), and the third in terms of their knowledge sets (Proposition 2). Moving to a setting of posets and continuous functions, we have presented direct analogues of the first of these in terms of ordered kernels ($\precsim_{\mathrm{LoCI}}$), and of the third in terms of ordered knowledge sets (Proposition 5). However, it turns out that there is no direct analogue of the postprocessing correspondence. To see why, we consider two pairs of counterexamples which illustrate two essentially different ways in which the postprocessing correspondence fails for LoCI.

Counterexample 1: Non-Existence of a Monotone Postprocessor. Consider a test `isEven1` on natural numbers which simply returns True or False. This can be modelled in the obvious way by a function $\mathrm{isEven1} \in [N \to \mathrm{Bool}_\perp]$, where N is the unordered set of natural numbers and $\mathrm{Bool}_\perp$ is the lifted domain of Booleans in Fig. 4a.

Now consider the following Haskell-style function definition

```
isEven x  = if even x then ((), spin) else (spin, ())
  where spin = spin
```

Tuples in Haskell are both lazy and lifted, so this can be modelled by a function $\mathrm{isEven2} \in [N \to D]$, where D is the lifted diamond domain in Fig. 4a. (Haskell does not have a primitive type for



(a) Codomains (b) Kernels

Fig. 4. Postprocessing Counterexamples

natural numbers, but only integers, but for the sake of the example let us assume that the program operates over naturals.)

Both these functions have the same kernel (ordered and unordered): it simply partitions N into the sets of even and odd numbers. So (isEven2 $\precsim_{\text{LoCI}}$ isEven1) and (isEven1 $\precsim_{\text{LoCI}}$ isEven2). We can certainly obtain isEven2 from isEven1 by postprocessing: map $\bot$ to $\bot$, map T to $(*, \bot)$, and map F to $(\bot, *)$. However, there is no continuous postprocessor $p \in [D \to \text{Bool}_\bot]$ such that isEven1 $= p \circ$ isEven2. The problem is that any such p must map $(*, \bot)$ to T and $(\bot, *)$ to F. But then, since $(*, *)$ is greater than both $(*, \bot)$ and $(\bot, *)$, p must map $(*, *)$ to a value greater than both T and F, and no such value exists. Note, however that $(*, *)$ is not actually in the *range* of isEven2. If p was not required to be monotone, the problem would therefore be easily resolved, since p could arbitrarily map $(*, *)$ to either T or F (or even to $\bot$). Unfortunately, such p would not actually be computable. Nonetheless, it is clear that it is indeed *computationally feasible* to learn exactly the same information from the output of the two functions. For example, we may poll the two elements in the output of isEven2 in alternation, until one becomes defined; as soon as this happens we will know the parity of the input. This behaviour is clearly implementable in principle, even though it does not define a monotone function in $D \to \text{Bool}_\bot$. (Of course, we cannot implement this behaviour in sequential Haskell, but this is just a limitation of the language.)

Conceivably, a slightly more liberal postprocessing condition could be designed to accommodate this and similar counterexamples (allow postprocessors to be partial, for example).

Counterexample 2: Non-Existence of a Continuous Postprocessor. Consider these two programs:

```
S1: if (x == 0) while True output();
    for i := 1 to x - 1 {
      output ()
    };
    while True { }
```

```
S2: if (x == 0) while True output();
    while True { }
```

Both programs take a natural number x and produce a partial or infinite stream of units. They can be modelled by functions $S_1, S_2 \in [N \to \Omega]$, where Ω is the poset illustrated in Fig. 4a. (In the picture for Ω we represent each partial stream of units by its length; the limit point ω represents the infinite stream.) When $x = 0$, both programs produce an infinite stream. When $x > 0$, S1 produces a stream of length $x - 1$, and then diverges; S2 simply diverges immediately.

As illustrated in Fig. 4b, the ordered kernel for S_1 is isomorphic to Ω, while the ordered kernel for S_2 is a two-point lattice. Clearly, $S_2 \precsim_{\text{LoCI}} S_1$. But there is no continuous $p \in [\Omega \to \Omega]$ such that $S_2 = p \circ S_1$. The problem in this case is that p would have to send all the finite elements of Ω to the bottom point 0, while sending the limit point ω to a different value.

The key thing to note here is that, although $[\ker_{\sqsubseteq}(S_1)]$ contains $\{0\}$ as a maximal element (it is the inverse image under S1 of the infinite output stream) an observer of S1 will never actually learn that $x = 0$ in finite time. With each observed output event, the observer rules out one more possible value for x, but there will always be infinitely many possible values remaining. After observing n output events, the observer knows only that $x = 0 \vee x > n$. By contrast, an observer of S2 learns that $x = 0$ as soon as the first output event is observed. (On the other hand, when $x > 0$, an S2 observer learns nothing at all.)

Perhaps the best we can claim is that the LoCI model is conservative, in the sense that it faithfully captures what an observer will learn "in the limit". But, as S1 illustrates, sometimes the limit never comes.

4 TERMINATION-INSENSITIVE PROPERTIES

In this section we turn to the question of how LoCI can help us to formulate the first general definition of a class of weakened information-flow properties known as *termination-insensitive* properties (or sometimes, *progress-insensitive* properties).

4.1 What is Termination-Insensitivity?

We quote Askarov et al. [2008]:

> Current tools for analysing information flow in programs build upon ideas going back to Denning's work from the 70's ⟨[Denning and Denning 1977]⟩. These systems enforce an imperfect notion of information flow which has become known as termination-insensitive noninterference. Under this version of noninterference, information leaks are permitted if they are transmitted purely by the program's termination behaviour (i.e. whether it terminates or not). This imperfection is the price to pay for having a security condition which is relatively liberal (e.g. allowing while-loops whose termination may depend on the value of a secret) and easy to check.

The term *noninterference* in the language-based security literature refers to a class of information flow properties built around a lattice of security labels (otherwise known as *security clearance levels*) [Denning 1976], in the simplest case two labels, H (the label for secrets) and L (the label for non-secrets), together with a "may flow" partial order $\prec$, where in the simple case $L \prec H$, expressing that public data may flow to (be combined with) secrets.

On the semantic side, for each label k there is a notion of *indistinguishability* between inputs and, respectively, outputs – equivalence relations which determines whether an observer at level k can see the difference between two different elements. These relations must agree with the flow relation in the sense that whenever $j \prec k$ then indistinguishability at level k implies indistinguishably at level j. Indistinguishability relations are either given directly, or can be constructed as the kernel of some projection function which extracts the data of classification at most k. Thus "ideal" noninterference for a program denotation f can be stated in terms of the lattice of information as a conjunction of properties of the form $f : P_k \Rightarrow Q_k$, expressing that an output observer at level k learns no more than the level-k input.

Without focusing on security policies in particular, we will show how to take any property of the form $f : P \Rightarrow Q$ and weaken it to a property which allows for termination leaks. The key to

this is to use the preorder refinement of Q to get a handle on exactly what leaks to allow. The case when P and Q are used to model security levels will just be a specific instantiation. But even for this instantiation we present a new generalisation of the notion of termination sensitivity beyond the two special cases that have been studied in the literature, namely (i) the "batch-job" case when programs either terminate or deliver a result, and (ii) the case when programs output a stream of values. In the recent literature the term *progress-insensitivity* has been used to describe the latter case, but in this section we will not distinguish these concepts – they are equally problematic for a Denning-style program analysis. Case (i) we will refer to henceforth as *simple termination-insensitive noninterference* and is relevant when the result domain of a computation is a flat domain.

As a simple example of case (i) consider the programs

$$A = \texttt{while (h>0) \{ \}} \qquad \text{and} \qquad B = \texttt{while (h>0) \{h := h-1\}}.$$

Assume that h is a secret. Standard information flow analyses notice that the loop condition in each case references variable h, but since typical analyses do not have the ability to analyse termination properties of loops, they must conservatively assume that information about h leaks in both cases (when in fact it only leaks for program A). This prevents us from verifying the security of any loops depending on secrets. However, a termination-insensitive analysis ignores leaks through termination behaviour and thus both A and B are permitted by termination-insensitive noninterference: such an analysis is more permissive because it allows loops depending on secrets (such as B), but less secure because it also allows leaky program A (which terminates only when $h \leq 0$).

Case (ii), progress-insensitivity, is the same issue but for programs producing streams. Consider here two programs which never terminate (thanks to $D = \texttt{while True \{ \}}$):

$$A' = \texttt{output(1)}; A; \texttt{output(1)}; D \qquad \text{versus} \qquad B' = \texttt{output(1)}; B; \texttt{output(1)}; D.$$

Here B' is noninterfering but A' is not, but both are permitted by the termination-insensitive condition (aka progress-insensitivity) for stream output defined in e.g. [Askarov et al. 2008]. The point of this example is to illustrate that the carrier of the information leak is not just the simple "does it terminate or not", but the *cause* of the leak is the same.

The definition in [Askarov et al. 2008] is ad hoc in that it is specific to the particular model of computation. If the computation model is changed (for example, if there are parallel output streams, or if there is a value delivered on termination) then the definition has to be rebuilt from scratch, and there is no general recipe to do this.

4.2 Detour: Termination-Insensitivity in the Lattice of Information

Before we get to our definition, it is worth considering how termination-insensitive properties might be encoded in the lattice of information directly. The question is how to take an arbitrary property of the form $P \Rightarrow Q$ and weaken it to a termination-insensitive variant $P' \Rightarrow Q'$.

We are not aware of a general approach to this in the literature. In this section we look at a promising approach which works for some specific and interesting choices of P and Q, but which we failed to generalise. We will later prove that it cannot be generalised in a way which matches the definition which we provide in §4.3.

So how might one weaken a property of the form $P \Rightarrow Q$ to allow termination leaks? It is tempting to try to encode this by weakening Q (taking a more liberal relation) – and indeed that is what has been done in typical relational proofs of simple termination-insensitive noninterference by breaking transitivity and allowing any value in the codomain to be indistinguishable from $\bot$. Our approach in §4.3 can be seen as a generalisation of this approach. But it is useful first to consider how far we can get while remaining within the realm of equivalence relations. Sterling

and Harper in a recent paper on the topic [Sterling and Harper 2022] say (in relation to a specific work [Abadi et al. 1999] using a relational, semantic proof of noninterference)

> "A more significant and harder to resolve problem is the fact that the indistinguishability relation ... cannot be construed as an equivalence relation"

While this seems to be true if we restrict ourselves to solving the problem by weakening Q, in fact it *is* possible to express termination-insensitivity of types (i) and (ii) just using equivalence relations. The trick is not to weaken Q, but instead to strengthen P.

The approach, which we briefly introduce here, is based on Bay and Askarov's study of progress-insensitive noninterference [Bay and Askarov 2020]. Their idea is to characterise a hypothetical observer who *only* learns through progress or termination behaviour. In the specific case of [Bay and Askarov 2020] it is a "progress observer" who sees the length of the output stream, but not the values within it. Let us illustrate this idea in the more basic context of simple termination-insensitive properties. Suppose we want to define a simple termination-insensitive variant of a property of the form $f : P \Rightarrow Q$ for some function $f \in [D \to V_\bot]$ where V is a flat set of values. We characterise the termination observer by the relation $T = \{(\bot, \bot)\} \cup \{(\mathrm{lift}(u), \mathrm{lift}(v)) \mid u \in V, v \in V\}$. The key idea is that we modify the property $f : P \Rightarrow Q$ not by weakening the observation Q, but by strengthening the prior knowledge P. We need to express that by observing Q you learn nothing more than P plus whatever you can learn from termination; here "plus" means least upper bound, and "what you learn from termination" is expressed as the generalised kernel of f with respect to T, namely $f^*(T)$. Thus the simple termination-insensitive weakening of $f : P \Rightarrow Q$ is

$$f : P \sqcup_{\mathrm{LoI}} f^*(T) \Rightarrow Q.$$

The general idea could then be, for each codomain, to define a suitable termination observer T. Bay and Askarov did this for the domain of streams to obtain "progress-insensitive" noninterference. We see two reasons to tackle this differently:

(1) Reasoning explicitly about $P \sqcup_{\mathrm{LoI}} f^*(T)$ is potentially cumbersome, especially since we don't care *what* is leaked in a termination-insensitive property.
(2) Finding a suitable T that works as intended but over an arbitrary domain is not only non-obvious, but, we suspect, not possible in general.

In §4.4 we return to point (2) to show that it is not possible to find a definition of T which matches the generalised termination-insensitivity which we now introduce.

4.3 Using LoCI to Define Generalised Termination-Insensitivity

Here we provide a general solution to systematically weakening an LoI property $f : R \Rightarrow S$ to a termination-insensitive counterpart (we assume S is realisable).

The first step is to encode $f : R \Rightarrow S$ as the LoCI property $f : P \Rightarrow Q$, where $P = \mathrm{Cp}(R)$ and $Q = \mathrm{Cp}(S)$, as allowed by Corollary 1. Preorder Q has the same equivalence classes as S, but the classes themselves are minimally ordered to respect the domain order; it is precisely this ordering which gives us a handle on the weakening we need to make.

As a starting point, consider how simple termination-insensitive noninterference is proven: one ignores distinctions that the observer might make between nontermination and termination. In a relational presentation (e.g. [Abadi et al. 1999]) this is achieved by simply relating bottom to everything (and vice-versa) and not requiring transitivity. What is the generalisation to richer domains (i.e. domains with more "height")? The first natural attempt comes from the observation that, in a Scott-style semantics, operational differences in termination behaviour manifest denotationally as differences in definedness, i.e. as inequations with respect to the domain ordering.

Towards a generalisation, let us start by assuming that S is the identity, so preorder $Q = \mathrm{Cp}(\mathrm{Id})$ is the top element of LoCI, i.e. it is just the domain ordering. This corresponds to an observer who can "see" everything (but some observations are more definite than others). The obvious weakening of the property $f : P \Rightarrow (\sqsubseteq)$ is to symmetrise $(\sqsubseteq)$ thus:

$$\{(d, e) \mid d \sqsubseteq e \text{ or } e \sqsubseteq d\}$$

This is "the right thing" for some domains but not all. As an example of where it does *not* do the right thing, consider the domain $\mathbf{2} \times \mathbf{2}$ where $\mathbf{2} = \mathbf{1}_\perp$, and $\mathbf{1} = \{*\}$. This domain contains four elements in a diamond shape. Suppose that a value of this type is computed by two loops, one to produce the first element, and one to produce the second. A termination-insensitive analysis ignores the leaks from the termination of each loop, so our weakening of any desired relation on $\mathbf{2} \times \mathbf{2}$ must relate $(\perp, *)$ and $(*, \perp)$ (and hence termination-insensitivity must inevitably leak all information about this domain). But what do $(\perp, *)$ and $(*, \perp)$ have in common? The answer is that they represent computations that might turn out to be the same, should their computations progress, i.e. they have an upper bound with respect to the domain ordering.

What about when the starting point is an arbitrary $Q \in \mathrm{LoCI}(D)$? The story here is essentially the same, but here we must think of the equivalence classes of Q instead of individual elements, and the relation Q instead of the domain ordering.

Definition 14 (Compatible extension). *Given two elements $d, d' \in D$, and a preorder Q on D, we say that d and d' are* Q-compatible *if there exists an e such that $d\ Q\ e$ and $d'\ Q\ e$. Define $\widetilde{Q}$, the* compatible extension *of Q, to be $\{(d, d') \mid d$ is Q-compatible with $d'\}$.*

For any preorder Q, compatible extension has the following evident properties:

(1) $\widetilde{Q} \supseteq Q$ (if $d\ Q\ e$ then e is a witness to the compatibility of d and e, since Q is reflexive).
(2) $\widetilde{Q}$ is reflexive and symmetric (but not, in general, transitive).

A candidate general notion of termination-insensitive noninterference is then to use properties of the form

$$f : P \Rightarrow \widetilde{Q}$$

where P and Q are complete preorders. This captures the essential idea outlined above, and passes at least one sanity check: $f : P \Rightarrow \widetilde{Q}$ is indeed a weaker property than $f : P \Rightarrow Q$ (simply because $\widetilde{Q} \supseteq Q$). However, a drawback of this choice is that it lacks a strong composition property. In general, $f : P \Rightarrow \widetilde{Q} \wedge g : Q \Rightarrow \widetilde{R}$ does *not* imply that $g \circ f : P \Rightarrow \widetilde{R}$. For a counterexample, consider the following function $g \in [A \to A]$, where $A = \{0, 1, 2\}_\perp$:

$$g(a) = \begin{cases} \perp & \text{if } a = \perp \\ 0 & \text{if } a = 0 \\ 1 & \text{if } a = 1 \\ \perp & \text{if } a = 2 \end{cases} \qquad Q = \quad \{2\} \quad \{0\} \quad \{1\} \quad \{\perp\}$$

Let Q be the complete preorder whose underlying equivalence relation is the identity relation but which orders the elements of A in a diamond shape, as pictured above. It is easily checked that $g : Q \Rightarrow \widetilde{(\sqsubseteq)}$. Now, since Q has a top element, $\widetilde{Q}$ is just All, so for *every* P and f of appropriate type, it will hold that $f : P \Rightarrow \widetilde{Q}$. But it is *not* true that $g \circ f : P \Rightarrow \widetilde{(\sqsubseteq)}$ holds for every P and f (take $P = \mathrm{All}$ and $f = \mathrm{id}$, for example).

Clearly, the above counterexample is rather artificial. Indeed, it is hard to see how we might construct a program with denotation g such that a termination-insensitive analysis could be expected to verify $g : Q \Rightarrow \widetilde{(\sqsubseteq)}$. Notice that g not only fails to send Q-related inputs to $(\sqsubseteq)$-related outputs, it effectively ignores the ordering imposed by Q entirely, in that it fails even to preserve

Q-compatibility. This suggests a natural strengthening of our candidate notion. We define our generalisation of termination-insensitive noninterference over the lattice of computable information to be "preservation of compatibility":

Definition 15 (Generalised Termination-Insensitivity). *Let $f \in [D \to E]$ and let P and Q be elements of* LoCI(D) *and* LoCI(E)*, respectively. Define:*

$$f : P \Rightarrow^{\text{TI}} Q \quad \text{iff} \quad f : \widetilde{P} \Rightarrow \widetilde{Q}$$

Crucially, although this is stronger than our initial candidate, it is still a weakening of $\_ \Rightarrow \_$:

Lemma 3. *Let $f \in [A \to B]$. Let P and Q be complete preorders on A and B, respectively. Then*

$$f : P \Rightarrow Q \quad \textit{implies} \quad f : P \Rightarrow^{\text{TI}} Q.$$

Proof. Assume $f : P \Rightarrow Q$ and suppose $x \mathrel{\widetilde{P}} y$. Since $x \mathrel{\widetilde{P}} y$, there is some z such that $x \mathrel{P} z$ and $y \mathrel{P} z$. Since $f : P \Rightarrow Q$, we have $f(x) \mathrel{Q} f(z)$ and $f(y) \mathrel{Q} f(z)$, hence $f(x) \mathrel{\widetilde{Q}} f(y)$. □

Furthermore, Definition 15 gives us both compositionality and "subtyping":

Proposition 9. *The following inference rules are valid for all continuous functions and elements of* LoCI *of appropriate type:*

$$\frac{P' \sqsupseteq_{LoCI} P \quad f : P \Rightarrow^{\text{TI}} Q \quad Q \sqsupseteq_{LoCI} Q'}{f : P' \Rightarrow^{\text{TI}} Q'}\ \text{SubTI} \qquad \frac{f : P \Rightarrow^{\text{TI}} Q \quad g : Q \Rightarrow^{\text{TI}} R}{g \circ f : P \Rightarrow^{\text{TI}} R}\ \text{CompTI}$$

Proof. We rely on the general Sub and Comp rules (Fact 1).

For SubTI, the premise for f unpacks to $f : \widetilde{P} \Rightarrow \widetilde{Q}$ and the conclusion unpacks to $f : \widetilde{P'} \Rightarrow \widetilde{Q'}$. It suffices then to show that $P' \sqsupseteq_{\text{LoCI}} P$ implies $\widetilde{P'} \subseteq \widetilde{P}$ (and similarly for Q, Q'), since we can then apply the general Sub rule directly. So, suppose $P' \sqsupseteq_{\text{LoCI}} P$, hence $P' \subseteq P$, and suppose $x \mathrel{\widetilde{P'}} y$. Then, for some z, we have $x \mathrel{P'} z$ and $y \mathrel{P'} z$, thus $x \mathrel{P} z$ and $y \mathrel{P} z$, thus $x \mathrel{\widetilde{P}} y$, as required.

For CompTI we observe that it is simply a specialisation of the general Comp rule, since the premises unpack to $f : \widetilde{P} \Rightarrow \widetilde{Q}$ and $g : \widetilde{Q} \Rightarrow \widetilde{R}$, while the conclusion unpacks to $g \circ f : \widetilde{P} \Rightarrow \widetilde{R}$. □

4.4 Impossibility of a Knowledge-based Definition

In this section we return to the question of whether there exists a knowledge-based characterisation which matches our definition of termination-insensitivity, and show why this cannot be the case.

Suppose we start with an "ideal" property of the form $f : P \Rightarrow S$, where S is assumed to be realisable (by $\text{Cp}(S)$), and (for simplicity but without loss of generality) P is over a discrete domain (so $\text{Cp}(P) = P$).

The question, which we will answer in the negative, is whether we can construct a "termination observer" T from the structure of the codomain of f such that

$$f : P \sqcup_{\text{LoI}} f^*(T) \Rightarrow S \ \text{ iff } \ f : P \Rightarrow^{\text{TI}} \text{Cp}(S)$$

We build a counterexample based on the following Haskell code:

```
data Kite = Body () () | Tail

spin = spin

f h = if h then Body () spin else Body spin ()
g h = if h then Body () spin else Tail
```

We will use some security intuitions to present the example (since that is the primary context in which termination-insensitivity is discussed). Suppose that we view the input to `f` and `g` as either **`True`** or **`False`**, and that this is a secret. We are being sloppy here and ignoring the fact that the input domain is lifted, but that has no consequence on the following.

Now consider the output to be public, and the question is whether `f` and `g` satisfy termination-insensitive noninterference. Standard noninterference in this case would be the property $(\_ : \mathrm{All} \Rightarrow \mathrm{Id})$. Our definition of termination-insensitive noninterference is thus $(\_ : \mathrm{All} \Rightarrow^{\mathrm{TI}} (\sqsubseteq))$ where $\sqsubseteq$ here is the ordering on the domain corresponding to `Kite`, namely the domain in Fig. 5. By our definition, `f` satisfies termination-insensitive noninterference but `g` does not. This is perhaps not obvious for `f` because a typical termination-insensitive *analysis* would reject it anyway, so it is instructive to see a semantically equivalent definition `f'` (assuming well-defined Boolean input) which would pass a termination-insensitive analysis[5].

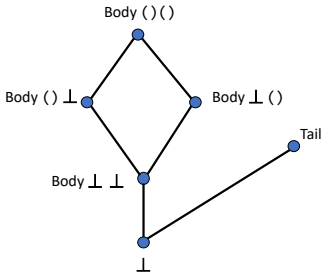


Fig. 5. Domain representing `Kite`

```
f' h = Body (assert h ()) (assert (not h) ())
         where  assert b y = seq (if b then () else spin) y
```

We claim that a semantic definition of termination-insensitive noninterference should accept `f'` (and hence `f`) but reject `g`. The reason for this is a fundamental feature of sequential computation, embodied in programming constructs such as call-by-value computation or sequential composition in imperative code. In Haskell, sequential computation is realised by a primitive function **`seq`**, which computes its first argument then, if it terminates, returns its second argument. Consider an expression of the form **`seq`** `a b` where `a` may depend on a secret, but `b` provably does not. The only way that such a computation reveals information about the secret is if the *termination* of `a` depends on the secret. This is the archetypal example of the kind of leak that a termination-insensitive analysis ignores. A particular case of this is the function `assert` in the code above, which leaks the value of its first parameter via (non)termination. For this reason, even when `h` is a secret, terms `assert h ()` and `assert (not h) ()` are considered termination-insensitive noninterfering (and thus so is `f'`). This example forms the basis of our impossibility claim, the technical content of which is the following:

Proposition 10. *There is no termination observer T (i.e. an equivalence relation) on the* `Kite` *domain for which* `f` : $\mathrm{All}\ \sqcup_{\mathrm{LoI}}\ \mathtt{f}^*(T) \Rightarrow \mathrm{Id}$ *but for which this does not hold for* `g`.

Proof. The problem is to define T in such a way that it distinguishes different `Body` instances but none of the `Body` instances from `Tail`, while still being an equivalence relation. T would either have to (1) relate `Body () ⊥` and `Body ⊥ ()` or (2) distinguish them and also distinguish one of them from `Tail` (if it related both to `Tail`, then, by transitivity and symmetry, it would also relate the two `Body` instances). Without loss of generality, assume $(\mathtt{Body\ ()\ \bot}, \mathtt{Tail}) \notin T$ (otherwise adjust `g` accordingly). In case (1), `f` does not have property $\mathrm{All}\ \sqcup_{\mathrm{LoI}}\ \mathtt{f}^*(T) \Rightarrow \mathrm{Id}$, because $\mathtt{f}^*(T)$ is All, but `f True` $\neq$ `f False`. In case (2), `g` does have this property because $\mathtt{g}^*(T)$ is the identity relation. □

[5]One should not be surprised that a program analysis can yield different results on semantically equivalent programs – as Rice's theorem [Rice 1953] shows, this is the price to pay for any non-trivial analysis which is decidable, and having a semantic soundness condition.

4.5 Case Study: Nondeterminism and Powerdomains

In this section we consider the application of generalised termination-insensitive properties to nondeterministic languages modelled using *powerdomains* [Plotkin 1976]. In the first part we instantiate our definition for a finite powerdomain representing a nondeterministic computation over lifted Booleans and illustrate that it "does the right thing". In the second part we prove that we have an analogous compositional reasoning principle to the function composition property CompTI (Proposition 9), but replacing regular composition with the Kleisli composition of the finite powerdomain monad.

Example: Termination-Insensitive Nondeterminism. We are not aware of any specific studies of termination-insensitive noninterference for nondeterministic languages, and the definitions in this paper were conceived independently of this example, so it provides an interesting case study.

Suppose we have a nondeterministic program C, modelled as a function in Bool $\to$ $\wp(\text{Bool}_\perp)$, where $\wp$ is the Plotkin powerdomain constructor and Bool = {**True**, **False**}. In the case of powerdomains over finite domains, the elements can be viewed as convex subsets of the underlying domain (see below for more technical details). In this section we only consider such finite powerdomains. $\wp(\text{Bool}_\perp)$, for example, is given in Figure 6.

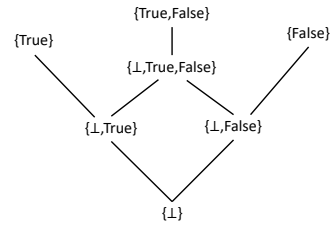


Fig. 6. Powerdomain $\wp(\text{Bool}_\perp)$

Each element of the powerdomain represents a set of possible outcomes of a nondeterministic computation. Let's consider the input of some program C to be a secret, and the output public. The property of interest here is what we can call TI-security, i.e., C : All $\Rightarrow^{\text{TI}}$ ($\sqsubseteq$).

To explore this property, let us assume an imperative programming language with the following features:

- a choice operator $C_1 \mid C_2$ which chooses nondeterministically to compute either C_1 or C_2,
- a Boolean input `x`, and
- an output statement to deliver a final result.

Note how the semantics of | can be given as set union of values in the powerdomain.

Under our definition, the compatible extension of the domain ordering for $\wp(\text{Bool}_\perp)$ relates all the points in the lower diamond to each other. Note that in particular this means that {⊥, **True**} and {⊥, **False**} are related. This in turn means that the following program C is TI secure:

```
while True { } | output x
```

This looks suspicious, to say the least. A static analysis would never allow such a program. But our definition says that it is TI-secure, since the denotation of C maps **True** to {⊥, **True**} and **False** to {⊥, **False**}, and these are compatible by virtue of the common upper bound {⊥, **True**, **False**}.

To show that our definition is, nonetheless, "doing the right thing", we can write C in a semantically equivalent way as:

```
( while x { } ; output False ) | ( while (not x) { }; output True)
```

Not only is this equivalent, but the insecurity apparent in the first rendition of the program is now invisible to a termination-insensitive analysis.

Now we turn to properties relevant to compositional reasoning about generalised termination-insensitivity for nondeterministic programs modelled using finite powerdomains.

Compositional Reasoning for Finite Powerdomains. We review the basic theory of finite Plotkin powerdomains, as developed in [Plotkin 1976]. We then define a natural lifting of complete preorders to powerdomains and show that this yields pleasant analogues (Corollary 2) of the SubTI and CompTI inference rules (Proposition 9) with respect to the powerdomain monad. Note: throughout this section we restrict attention to finite posets, so a preorder is complete iff it contains the partial order of its domain.

The Plotkin powerdomain construction uses the so-called Egli-Milner ordering on subsets of a poset, derived from the order of the poset. For our purposes it is convenient to generalise the Egli-Milner definition to arbitrary binary relations:

Definition 16 (Egli-Milner extension). *Let R be a binary relation on A. Then* $\mathrm{EM}(R)$ *is the binary relation on subsets of A defined by*

$$X\ \mathrm{EM}(R)\ Y \ \text{ iff } \ (\forall x \in X.\exists y \in Y.x\ R\ y) \wedge (\forall y \in Y.\exists x \in X.x\ R\ y)$$

Fact 2. (1) $\mathrm{EM}(\_)$ *is monotone.* (2) $\mathrm{EM}(\_)$ *preserves reflexivity, transitivity, and symmetry.*

The Egli-Milner ordering on subsets of a poset A is then $\mathrm{EM}(\sqsubseteq)$. Note that (2) entails that $\mathrm{EM}(R)$ is a preorder whenever R is a preorder. However, since antisymmetry is *not* preserved, in general $\mathrm{EM}(\sqsubseteq)$ is only a preorder, so to obtain a partial order it is necessary to quotient by the induced equivalence relation. Conveniently, the *convex* subsets provide a natural canonical representative for each equivalence class:

Definition 17 (Convex Closure). *The* convex closure *of X is* $\mathrm{Cv}(X) \stackrel{\text{def}}{=} \{b \in A \mid a \in X, c \in X, a \sqsubseteq b \sqsubseteq c\}$.

Fact 3. (1) Cv *is a closure operator.* (2) $\mathrm{Cv}(X)$ *is the largest member of* $[X]_{\mathrm{EM}(\sqsubseteq)}$.

Definition 18 (Finite Plotkin Powerdomain). *Let $(A, \sqsubseteq)$ be a finite poset. Then the* Plotkin powerdomain $\wp(A)$ *is the poset of all non-empty convex subsets of A ordered by* $\mathrm{EM}(\sqsubseteq)$. *The union operation is defined by* $X \mathbin{\bar{\cup}} Y \stackrel{\text{def}}{=} \mathrm{Cv}(X \cup Y)$.

The powerdomain constructor is naturally extended to a monad, allowing us to compose functions with types of the form $A \to \wp(B)$.

Definition 19 (Kleisli-extension). *Let A, B be finite posets. Let $f \in [A \to \wp(B)]$. The Kleisli-extension of f is $f^\dagger \in [\wp(A) \to \wp(B)]$ defined by $f^\dagger(X) = \mathrm{Cv}(\bigcup_{x \in X} f(x))$.*

Definition 20 (Kleisli-composition). *Let A, B, C be finite posets and let $f \in [A \to \wp(B)]$ and $g \in [B \to \wp(C)]$. Then the Kleisli-composition $f ; g \in [A \to \wp(C)]$ is $g^\dagger \circ f$.*

We lift the powerdomain constructor to binary relations in the obvious way:

Definition 21. *Let R be a binary relation on finite poset A. Then $\wp(R)$ is the relation on $\wp(A)$ obtained by restricting* $\mathrm{EM}(R)$ *to non-empty convex sets.*

Lemma 4. *If P is a complete preorder on finite poset A then $\wp(P)$ is a complete preorder on $\wp(A)$.*

Now, in order to establish our desired analogues of SubTI and CompTI, we must be able to relate $\widetilde{\wp(P)}$ to $\widetilde{P}$. The key properties are the following:

Lemma 5. *Let R be a preorder and let P be a complete preorder. Then:*

(1) $\widetilde{\mathrm{EM}(R)} = \mathrm{EM}(\widetilde{R})$ (2) $\mathrm{Cv}(X)\ \widetilde{\wp(P)}\ \mathrm{Cv}(Y)$ *iff* $X\ \widetilde{\mathrm{EM}(P)}\ Y$ (3) $\widetilde{\wp(P)} = \wp(\widetilde{P})$

We then have:

Theorem 3. *Let A, B be finite posets and let $f \in [A \to \wp(B)]$. Let P, P' be a complete preorders on A and let Q be a complete preorder on B.*

(1) *If $P' \sqsupseteq_{LoCI} P$ then $\widetilde{\wp(P')} \subseteq \widetilde{\wp(P)}$.*
(2) *If $f : P \Rightarrow^{\text{TI}} \wp(Q)$ then $f^\dagger : \wp(P) \Rightarrow^{\text{TI}} \wp(Q)$.*

Proof.

(1) By definition $P' \sqsupseteq_{\text{LoCI}} P$ iff $P' \subseteq P$ hence, as argued in the proof of Proposition 9, $P' \sqsupseteq_{\text{LoCI}} P$ implies $\widetilde{P'} \subseteq \widetilde{P}$. The conclusion then follows by monotonicity of $\text{EM}(\_)$ and Lemma 5.
(2) Assume $f : P \Rightarrow^{\text{TI}} \wp(Q)$. By the definition of $f^\dagger$ and Lemma 5, it suffices to show that $X_1 \ \text{EM}(\widetilde{P}) \ X_2$ implies $Z_1 \ \text{EM}(\widetilde{Q}) \ Z_2$, where $Z_i = \bigcup_{x \in X_i} f(x)$. Let $z_1 \in Z_1$, thus $z_1 \in f(x_1)$ for some $x_1 \in X_1$. Since $X_1 \ \text{EM}(\widetilde{P}) \ X_2$, there is some $x_2 \in X_2$ with $x_1 \ \widetilde{P} \ x_2$. Since $f : P \Rightarrow^{\text{TI}} \wp(Q)$, it follows that $f(x_1) \ \widetilde{\wp(Q)} \ f(x_2)$, hence by Lemma 5 $f(x_1) \ \text{EM}(\widetilde{Q}) \ f(x_2)$, hence $z_1 \ \widetilde{Q} \ z_2$ for some $z_2 \in f(x_2) \subseteq Z_2$. Thus $\forall z_1 \in Z_1 . \exists z_2 \in Z_2 . z_1 \ \widetilde{Q} \ z_2$. It follows by a symmetrical argument that $\forall z_2 \in Z_2 . \exists z_1 \in Z_1 . z_1 \ \widetilde{Q} \ z_2$. □

Corollary 2. *Let A, B, C be finite posets and let $f \in [A \to \wp(B)]$ and $g \in [B \to \wp(C)]$. The following inference rules are valid for all elements of* LoCI *of appropriate type:*

$$\frac{P' \sqsupseteq_{LoCI} P \quad f : P \Rightarrow^{\text{TI}} \wp(Q) \quad Q \sqsupseteq_{LoCI} Q'}{f : P' \Rightarrow^{\text{TI}} \wp(Q')} \qquad \frac{f : P \Rightarrow^{\text{TI}} \wp(Q) \quad g : Q \Rightarrow^{\text{TI}} \wp(R)}{f;g : P \Rightarrow^{\text{TI}} \wp(R)}$$

5 RELATED WORK

Readers of this paper hoping to see a reconciliation of Shannon's quantitative information theory with domain theory may be disappointed to see that we have tackled a less ambitious problem based on Shannon's lesser-known qualitative theory of information. Abramsky [2008] discusses the issues involved in combining the quantitative theory of Shannon with the qualitative theory of Scott and gives a number of useful pointers to the literature.

As we mentioned in the introduction, Shannon's paper describing information lattices [Shannon 1953] is relatively unknown, but a more recent account by Li and Chong [2011] make Shannon's ideas more accessible (see also [Rioul et al. 2022]). Most later works using similar abstractions for representing information have been made independently of Shannon's ideas. In the security area, Cohen [1977] used partitions to describe varieties of information flow via so-called *selective dependencies*. In an independent line of work, various authors developed the use of the lattice of partial equivalence relations (PERs) to give semantic models to polymorphic types in programming languages e.g. [Abadi and Plotkin 1990; Coppo and Zacchi 1986]. PERs generalise equivalence relations by dropping the reflexivity requirement, so a PER is just an equivalence relation on a subset of the space in question. An important generalisation over equivalence relations, particularly when used for semantic models of types, is that "flow properties" of the form $f : P \Rightarrow Q$ can expressed by interpreting $P \Rightarrow Q$ itself as a PER over functions, and $f : P \Rightarrow Q$ is just shorthand for f being related to itself by this PER. The connection to information flow and security properties comes via *parametricity*, a property of polymorphic types which can be used to establish noninterference e.g. [Bowman and Ahmed 2015; Tse and Zdancewic 2004].

Independent of all of the above, Landauer and Redmond [1993] described the use of the lattice of equivalence relations to describe security properties, dubbing it *a lattice of information*. Sabelfeld and Sands [2001], inspired by the use of PERs for static analysis of dependency [Hunt 1991; Hunt and Sands 1991] (and independent of Landauer and Redmond's work) used PERs over domains to give semantic models of information flow properties, including for more complex domains for nondeterminism and probability, and showed that the semantic properties could be used to prove

semantic soundness of a simple type system. Our TI results in § 4.5 mirror the termination-sensitive composition principle for powerdomains given by Sabelfeld and Sands [2001]. Hunt and Sands [2021] introduce a refinement of LoI, orthogonal to the present paper, which adds disjunctive information flow properties to the lattice. Li and Zdancewic [2005] use a postprocessing definition of declassification policies in the manner of Proposition 1(2); Sabelfeld and Sands [2009] sketched how this could be reformulated within LoI.

Giacobazzi and Mastroeni [2004] introduced *abstract noninterference* (ANI) in which a security-centric noninterference property is parameterised by abstract interpretations to represent the observational power of an attacker, and the properties to be protected. Hunt and Mastroeni [2005] showed how so-called *narrow* ANI in [Giacobazzi and Mastroeni 2004] and some key results can be recast as properties over LoI. In its most general form, Giacobazzi and Mastroeni [2018, Def 4.2] define ANI as a property of a function f parameterised by three upper closure operators (operating on *sets* of values): an output observation ρ, an input property ϕ which may flow, and an input property η "to protect". A function f is defined to have abstract noninterference property $\{\phi, \eta\} f \{\rho\}$ if, for all x, y:

$$\phi(\{x\}) = \phi(\{y\}) \text{ implies } \rho(\hat{f}(\eta(\{x\}))) = \rho(\hat{f}(\eta(\{y\})))$$

where $\hat{f}$ is the lifting of f to sets. Note that this can be directly translated to an equivalent property over the lattice of information, as follows:

$$\hat{f} \circ \eta' : \ker(\phi') \Rightarrow \ker(\rho)$$

where $\phi'(x) = \phi(\{x\})$ and $\eta'(x) = \eta(\{x\})$. In the special case that η is the identity, this reduces simply to an information flow property of f, namely:

$$f : \ker(\phi') \Rightarrow \ker(\rho')$$

where $\rho'(y) = \rho(\{y\})$. In the general case, Giacobazzi and Mastroeni [2018] observe that the ANI framework models attackers whose ability to make logical deductions (about the inputs of f) is constrained within the abstract interpretation fixed by ρ and η. Inheriting from the underlying abstract interpretation framework, ANI can be developed within a variety of semantic frameworks (denotational, operational, trace-based, etc.).

The lattice of information, either directly or indirectly provides a robust baseline for various quantitative measures of information flow [Malacaria 2015; McIver et al. 2014]. In the context of quantitative information flow, Alvim et al. [2020, Chapter 16] discuss leakage refinement orders for potentially nonterminating probabilistic programs. Their ordering allows increase in "security" or termination. Increase in security here corresponds to decrease in information (a system which releases no information being the most secure). Thus Alvim et al.'s ordering is incomparable to ours: the LoCI ordering reflects increase in information (decrease in security) or increase in termination.

Regarding the question of termination-sensitive noninterference, the first static analysis providing this kind of guarantee was by Denning and Denning [1977]. This used Dorothy Denning's lattice model of information [Denning 1976]. It is worth noting that Denning's lattice model is a model for security properties expressed via labels, inspired by, but generalising, classical military security clearance levels. As such these are syntactic lattices used to identify different objects in a system and to provide a definition of the intended information flows. But Denning's work did not come with any actual formal definition of information flow, and so their analysis did not come with any proof of a semantic security property. Such a proof came later in the form of a termination-insensitive noninterference property for a type system [Volpano et al. 1996], intended to capture the essence of Denning's static analysis. The semantic guarantees for such analysis in the presence of stream outputs was studied by Askarov et al. [2008]. There they showed that stream outputs can leak

arbitrary amounts of information through the termination (progress) insensitivity afforded by a Denning-style analysis, but also that the information is bound to leak slowly.

In recent work, Sterling and Harper [2022] propose a new semantic model for termination-insensitive noninterference properties using more elaborate domain-theoretic machinery. The approach is fundamentally type-centric, adopting sheaf semantics ideas from their earlier work on the semantics of phase distinctions [Sterling and Harper 2021]. The fundamental difference in their work is that it is an *intrinsic* approach to the semantics of information flow types in the sense of Reynolds [2003], whereby information flow specifications are viewed as an integral part of types, and thus the meaning of the type for a noninterfering function is precisely the semantics of noninterference. This is in contrast with the *extrinsic* relational models studied here in which information flow properties are characterised by properties carved out of a space of arbitrary functions. Though the merger of domains and relations sketched in §3.7 may be considered an intrinsic presentation, the approach of Sterling and Harper goes further: it requires a language of information flow properties to be part of the type language (and the underlying semantics). In their work the class of properties discussed is quite specific, namely those specifiable by a Denning-style (semi)lattice of security labels. The approach is particularly suited to reasoning about systems in the style of DCC [Abadi et al. 1999] in which security labels are part of the programming language itself. Unlike in the present work, the only kind of termination-insensitive noninterference discussed in their paper is the simple case in which a program either terminates or it does not.

The most advanced semantic soundness proof of termination-insensitive noninterference (in terms of programming language features) is the recent work of Gregersen et al. [2021]. In terms of advanced typing features (combinations of higher-order state, polymorphism, existential and recursive types…) this work is a tour de force, although the notion of termination-sensitivity at the top level is just the simplest kind; any termination-insensitive notions that arise internally through elaborate types are not articulated explicitly.

6 CONCLUSION AND FUTURE WORK

In this paper we have reconciled two different theories of information:

- Shannon's lattice model, which gives an encoding-independent view of the information that is released from some data source by a function, and orders one information element above another when it provides more information about the source; and
- Scott's domain theory, where an "information element" is a provisional representation of the information produced so far by a computational process, and the ordering relation reflects an increase in definedness, or computational progress.

Our combination of these models, which we have dubbed the Lattice of Computable Information (as a nod to the fact that Scott's theory is designed to model computable functions via continuity, even if it does not always do so perfectly) retains the essential features of both theories – it possesses the lattice properties which describe how information can be combined and compared, at the same time as taking into account the Scott ordering in a natural way. We have also shown how the combination yields the first definition, general in its output domain, of what it means to be the termination-insensitive weakening of an arbitrary flow property.

We identify some lines of further work which we believe would be interesting to explore:

New Information Flow Policies using LoCI. LoCI allows the expression of new, more fine-grained information flow properties, but which ones are useful? One example worth exploring relates to noninterference for systems with input streams. In much of the literature on noninterference for such systems there is an explicit assumption that systems are "input total", so that the system never blocks when waiting for a secret input. Using LoCI we have the machinery

to explore this space without such assumptions – we can formulate what we might call *input termination-sensitive* and *input termination-insensitive* properties within LoCI (without weakening). Input termination-sensitive properties are very strong since they assume that the high user might try to sneak information to a low observer via the decision to supply or withhold information, whereas insensitive properties permit upfront knowledge of the number of high inputs consumed, thus ignoring these flows.

Another place where LoCI can prove useful is in *required release* policies [Chong 2012], where a minimum amount of information flow is required (e.g. a freedom of information property). In this case we would like to ensure that the information which is released is produced in a "decent" form – i.e. as a maximal element among those LoCI elements which have the same equivalence classes. This prevents the use of nontermination to obfuscate the information.

Semantic Proofs of Noninterference. We have developed some basic semantic-level tools for compositional reasoning about information flow properties in LoCI, and their termination-insensitive relatives (e.g. Proposition 9 and Corollary 2). It seems straightforward to establish a flow-sensitive variant of the progress-insensitive type system of Askarov et al. [2008] that can be given a semantic soundness proof based on the definition of termination-insensitivity given here (the proof in [Askarov et al. 2008] is not given in the paper, but it is a syntactic proof). It would be interesting to tackle a more involved language, for example with both input and output streams, and with input termination-sensitive/insensitive variants. It would be important, via such case studies, to further develop an arsenal of properties, established at the semantic level, which can be reused across different proofs for different systems. For languages which support higher-order functions, semantic proofs would call for the ability to build complete preorders on continuous function spaces $[A \to B]$ by the usual logical relations construction. That is, given complete preorders P and Q on A and B, we would like to construct a complete preorder $P \Rightarrow Q$, relating f, and g just when $a\ P\ a'$ implies $f(a)\ Q\ g(a')$. In fact, defined this way, the relation $P \Rightarrow Q$ will in general only be a *partial* preorder (some elements of $[A \to B]$ will not be in the relation at all). Promisingly, the results in [Abadi and Plotkin 1990] suggest that complete partial preorders are well-behaved, yielding a cartesian-closed category.

Domain Constructors. The powerdomain results in §4.5 are limited to finite posets. It would be interesting to extend these results beyond the finite case and, more generally, to see if other domain constructions, including via recursive domain equations, can be lifted to complete preorders. Clearly this will require restriction to an appropriate category of algebraic domains, rather than arbitrary posets. It remains to be seen whether it will also be necessary to impose additional constraints on the preorders.

ACKNOWLEDGMENTS

Thanks to the anonymous referees for numerous constructive suggestions, in particular connections to category theory that formed the basis of §3.7, and the suggestion to use an example based on powerdomains. Thanks to Andrei Sabelfeld and Aslan Askarov for helpful advice. This work was partially supported by the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council (VR).

REFERENCES

M. Abadi, A. Banerjee, N. Heintze, and J. Riecke. 1999. A Core calculus of Dependency. In *Proc. ACM Symp. on Principles of Programming Languages*. 147–160.

Martín Abadi and Gordon D. Plotkin. 1990. A PER model of polymorphism and recursive types. *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science* (1990), 355–365.

Samson Abramsky. 1987. *Domain Theory and the Logic of Observable Properties*. Ph. D. Dissertation. University of London.

Samson Abramsky. 1991. Domain theory in logical form. *Annals of Pure and Applied Logic* 51, 1 (1991), 1–77. https://doi.org/10.1016/0168-0072(91)90065-T

Samson Abramsky. 2008. Information, processes and games. *J. Benthem van & P. Adriaans (Eds.), Philosophy of Information* (2008), 483–549.

Samson Abramsky and Achim Jung. 1995. Domain Theory. In *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*. Oxford University Press, Inc., USA, 1–168.

Mário S. Alvim, Konstantinos Chatzikokolakis, Annabelle McIver, Carroll Morgan, Catuscia Palamidessi, and Geoffrey Smith. 2020. *The Science of Quantitative Information Flow*. Springer. https://doi.org/10.1007/978-3-319-96131-6

Aslan Askarov and Stephen Chong. 2012. Learning is Change in Knowledge: Knowledge-Based Security for Dynamic Policies. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, Stephen Chong (Ed.). IEEE Computer Society, 308–322. https://doi.org/10.1109/CSF.2012.31

A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands. 2008. Termination Insensitive noninterference leaks more than just a bit. In *Proc. European Symp. on Research in Computer Security*.

A. Askarov and A. Sabelfeld. 2007. Gradual Release: Unifying Declassification, Encryption and Key Release Policies. In *Proc. IEEE Symp. on Security and Privacy*. 207–221.

Aslan Askarov and Andrei Sabelfeld. 2009. Tight Enforcement of Information-Release Policies for Dynamic Languages. In *2009 22nd IEEE Computer Security Foundations Symposium*. 43–59. https://doi.org/10.1109/CSF.2009.22

Johan Bay and Aslan Askarov. 2020. Reconciling progress-insensitive noninterference and declassification. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. 95–106. https://doi.org/10.1109/CSF49147.2020.00015

William J. Bowman and Amal Ahmed. 2015. Noninterference for Free. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming* (Vancouver, BC, Canada) *(ICFP 2015)*. Association for Computing Machinery, New York, NY, USA, 101–113. https://doi.org/10.1145/2784731.2784733

Stephen Chong. 2012. Required information release. *J. Comput. Secur.* 20, 6 (2012), 637–676.

Ellis Cohen. 1977. Information Transmission in Computational Systems. *SIGOPS Oper. Syst. Rev.* 11, 5 (Nov. 1977), 133–139. https://doi.org/10.1145/1067625.806556

M. Coppo and M. Zacchi. 1986. Type inference and logical relations. In *LICS*.

D. E. Denning. 1976. A Lattice Model of Secure Information Flow. *Comm. of the ACM* 19, 5 (May 1976), 236–243.

Dorothy E. Denning and Peter J. Denning. 1977. Certification of Programs for Secure Information Flow. *Commun. ACM* 20, 7 (1977), 504–513.

M. Erné, J. Koslowski, A. Melton, and G. E. Strecker. 1993. A Primer on Galois Connections. *Annals of the New York Academy of Sciences* 704, 1 (1993), 103–125. https://doi.org/10.1111/j.1749-6632.1993.tb52513.x

Roberto Giacobazzi and Isabella Mastroeni. 2004. Abstract non-interference: parameterizing non-interference by abstract interpretation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*. ACM, 186–197. https://doi.org/10.1145/964001.964017

Roberto Giacobazzi and Isabella Mastroeni. 2018. Abstract Non-Interference: A Unifying Framework for Weakening Information-Flow. *ACM Trans. Priv. Secur.* 21, 2, Article 9 (feb 2018), 31 pages. https://doi.org/10.1145/3175660

Simon Oddershede Gregersen, Johan Bay, Amin Timany, and Lars Birkedal. 2021. Mechanized Logical Relations for Termination-Insensitive Noninterference. *Proc. ACM Program. Lang.* 5, POPL, Article 10 (jan 2021), 29 pages. https://doi.org/10.1145/3434291

Sebastian Hunt. 1991. *Abstract interpretation of functional languages: from theory to practice*. Ph. D. Dissertation. Imperial College London, UK.

Sebastian Hunt and Isabella Mastroeni. 2005. The PER Model of Abstract Non-interference. In *Static Analysis*, Chris Hankin and Igor Siveroni (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 171–185.

Sebastian Hunt and David Sands. 1991. Binding Time Analysis: A New PERspective. In *In Proceedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'91*. ACM Press, 154–164.

Sebastian Hunt and David Sands. 2021. A Quantale of Information. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–15. https://doi.org/10.1109/CSF51468.2021.00031

J. Landauer and T. Redmond. 1993. A Lattice of Information. In *6th IEEE Computer Security Foundations Workshop - CSFW'93, Proceedings*. IEEE Computer Society, 65–70.

Hua Li and Edwin K. P. Chong. 2011. On a Connection between Information and Group Lattices. *Entropy* 13, 3 (2011), 683–708. https://doi.org/10.3390/e13030683

Peng Li and Steve Zdancewic. 2005. Downgrading Policies and Relaxed Noninterference. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Long Beach, California, USA) *(POPL '05)*. Association for Computing Machinery, New York, NY, USA, 158–170. https://doi.org/10.1145/1040305.1040319

Pasquale Malacaria. 2015. Algebraic foundations for quantitative information flow. *Mathematical Structures in Computer Science* 25, 2 (2015), 404–428. https://doi.org/10.1017/S0960129513000649

Annabelle McIver, Carroll Morgan, Geoffrey Smith, Barbara Espinoza, and Larissa Meinicke. 2014. Abstract Channels and Their Robust Information-Leakage Ordering. In *Principles of Security and Trust*, Martín Abadi and Steve Kremer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 83–102.

J. McLean. 1994. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. 79–93. https://doi.org/10.1109/RISP.1994.296590

Paul-André Melliès and Noam Zeilberger. 2015. Functors Are Type Refinement Systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 3–16. https://doi.org/10.1145/2676726.2676970

Oystein Ore. 1942. Theory of equivalence relations. *Duke Math. J.* 9, 3 (09 1942), 573–627. https://doi.org/10.1215/S0012-7094-42-00942-6

Gordon D. Plotkin. 1976. A Powerdomain Construction. *SIAM J. Comput.* 5, 3 (1976), 452–487. https://doi.org/10.1137/0205035

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*.

John C. Reynolds. 2003. What do types mean? — From intrinsic to extrinsic semantics. In *Programming Methodology*. Springer New York, New York, NY, 309–327. https://doi.org/10.1007/978-0-387-21798-7_15

H. G. Rice. 1953. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Amer. Math. Soc.* 74, 2 (1953), 358–366.

Olivier Rioul, Julien Béguinot, Victor Rabiet, and Antoine Souloumiac. 2022. La véritable (et méconnue) théorie de l'information de Shannon. In *28e Colloque GRETSI'22*.

A. Sabelfeld and A.C. Myers. 2003. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 21, 1 (2003), 5–19. https://doi.org/10.1109/JSAC.2002.806121

A. Sabelfeld and D. Sands. 2001. A Per Model of Secure Information Flow in Sequential Programs. *Journal of Higher-Order and Symbolic Computation* 14, 1 (March 2001), 59–91.

Andrei Sabelfeld and David Sands. 2009. Declassification: Dimensions and principles. *J. Comput. Secur.* 17, 5 (2009), 517–548. https://doi.org/10.3233/JCS-2009-0352

C. Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

C. Shannon. 1953. The lattice theory of information. *Transactions of the IRE Professional Group on Information Theory* 1, 1 (1953), 105–107. https://doi.org/10.1109/TIT.1953.1188572

M. B. Smyth. 1983. Power domains and predicate transformers: A topological view. In *Automata, Languages and Programming*, Josep Diaz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 662–675.

Jonathan Sterling and Robert Harper. 2021. Logical Relations as Types: Proof-Relevant Parametricity for Program Modules. *J. ACM* 68, 6, Article 41 (oct 2021), 47 pages. https://doi.org/10.1145/3474834

Jonathan Sterling and Robert Harper. 2022. Sheaf semantics of termination-insensitive noninterference. In *FSCD, the 7th International Conference on Formal Structures for Computation and Deduction*.

Stephen Tse and Steve Zdancewic. 2004. Translating Dependency into Parametricity. *SIGPLAN Not.* 39, 9 (sep 2004), 115–125. https://doi.org/10.1145/1016848.1016868

Steven Vickers. 1989. *Topology via Logic*. Cambridge University Press, USA.

D. Volpano, G. Smith, and C. Irvine. 1996. A Sound Type System for Secure Flow Analysis. *J. Computer Security* 4, 3 (1996), 167–187.

Received 2022-07-07; accepted 2022-11-07