Hypercollecting Semantics and Its Application to Static Analysis of Information Flow

Mounir Assaf
Stevens Institute of Technology, Hoboken, US
first.last@stevens.edu

David A. Naumann
Stevens Institute of Technology, Hoboken, US
first.last@stevens.edu

Julien Signoles
Software Reliability and Security Lab, CEA LIST, Saclay, FR
first.last@cea.fr

Éric Totel
CIDRE, CentraleSupélec, Rennes, FR
first.last@centralesupelec.fr

Frédéric Tronel
CIDRE, CentraleSupélec, Rennes, FR
first.last@centralesupelec.fr

Abstract

We show how static analysis for secure information flow can be expressed and proved correct entirely within the framework of abstract interpretation. The key idea is to define a Galois connection that directly approximates the hyperproperty of interest. To enable use of such Galois connections, we introduce a fixpoint characterisation of hypercollecting semantics, i.e. a "set of sets" transformer. This makes it possible to systematically derive static analyses for hyperproperties entirely within the calculational framework of abstract interpretation. We evaluate this technique by deriving example static analyses. For qualitative information flow, we derive a dependence analysis similar to the logic of Amtoft and Banerjee (SAS'04) and the type system of Hunt and Sands (POPL'06). For quantitative information flow, we derive a novel cardinality analysis that bounds the leakage conveyed by a program instead of simply deciding whether it exists. This encompasses problems that are hypersafety but not k-safety. We put the framework to use and introduce variations that achieve precision rivalling the most recent and precise static analyses for information flow.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification–Assertion checkers; D.3 [*Programming Languages*]; F.3.1 [*Logics and meanings of programs*]: Semantics of Programming Language

Keywords static analysis, abstract interpretation, information flow, hyperproperties

1. Introduction

Most static analyses tell something about all executions of a program. This is needed, for example, to validate compiler optimizations. Functional correctness is also formulated in terms of a predicate on observable behaviours, i.e. more or less abstract execution traces: A program is correct if all its traces satisfy the predicate. By contrast with such *trace properties*, extensional definitions of dependences involve more than one trace. To express that the final value of a variable x may depend only on the initial value of a variable y, the requirement—known as *noninterference* in the security literature (Sabelfeld and Myers 2003)—is that any two traces with the same initial value for y result in the same final value for x. Sophisticated information flow policies allow dependences subject to quantitative bounds—and their formalisations involve more than two traces, sometimes unboundedly many.

For secure information flow formulated as decision problems, the theory of *hyperproperties* classifies the simplest form of noninterference as *2-safety* and some quantitative flow properties as *hypersafety properties* (Clarkson and Schneider 2010). A number of approaches have been explored for analysis of dependences, including type systems, program logics, and dependence graphs. Several works have used abstract interpretation in some way. One approach to 2-safety is by forming a *product program* that encodes execution pairs (Barthe et al. 2004; Terauchi and Aiken 2005; Darvas et al. 2005), thereby reducing the problem to ordinary safety which can be checked by abstract interpretation (Kovács et al. 2013) or other means. Alternatively, a 2-safety property can be checked by dedicated analyses which may rely in part on ordinary abstract interpretations for trace properties (Amtoft et al. 2006).

The theory of abstract interpretation serves to specify and guide the design of static analyses. It is well known that effective application of the theory requires choosing an appropriate notion of observable behaviour for the property of interest (Cousot 2002; Bertrane et al. 2012, 2015). Once a notion of "trace" is chosen, one has a program semantics and "all executions" can be formalized in terms of *collecting semantics*, which can be used to define a trace property of interest, and thus to specify an abstract interpretation (Cousot and Cousot 1977, 1979; Cousot 1999).

The foundation of abstract interpretation is quite general, based on Galois connections between semantic domains on which collecting semantics is defined. Clarkson and Schneider (2010) formalize the notion of hyperproperty in a very general way, as a set of sets of traces. Remarkably, prior works using abstract interpretation for secure information flow do not directly address the set-of-sets dimension and instead involve various ad hoc formulations. This paper presents a new approach of deriving information flow static analyses within the calculational framework of abstract interpretation.

POPL'17, January 15–21, 2017, Paris, France
ACM. 978-1-4503-4660-3/17/01...$15.00
http://dx.doi.org/10.1145/3009837.3009889

First contribution. We lift collecting semantics to sets of trace sets, dubbed *hypercollecting semantics*, in a fixpoint formulation which is not simply the lifted direct image. This can be composed with Galois connections that specify hyperproperties beyond 2-safety, without recourse to ad hoc additional notions. On the basis of this foundational advance, it becomes possible to derive static analyses entirely within the calculational framework of abstract interpretation (Cousot and Cousot 1977, 1979; Cousot 1999).

Second contribution. We use hypercollecting semantics to derive an analysis for ordinary dependences. This can be seen as a rational reconstruction of both the type system of Hunt and Sands (2006, 2011) and the logic of Amtoft and Banerjee (2004). They determine, for each variable x, a conservative approximation of the variables y whose initial values influence the final value of x.

Third contribution. We derive a novel analysis for quantitative information flow. This shows the benefit of taking hyperproperties seriously by means of abstract interpretation. For noninterference, once the variables y on which x depends have fixed values, there can be only one final value for x. For quantitative information flow, one is interested in measuring the extent to which other variables influence x: for a given range of variation for the "high inputs", what is the range of variation for the final values of x? We directly address this question as a hyperproperty: given a set of traces that agree only on the low inputs, what is the cardinality of the possible final values for x? Using the hypercollecting semantics, we derive a novel *cardinality abstraction*. We show how it can be used for analysis of quantitative information problems including a bounding problem which is not k-safety for any k.

The calculational approach disentangles key design decisions and it enabled us to identify opportunities for improving precision. We assess the precision of our analyses and provide a formal characterisation of precision for a quantitative information flow analysis vis a vis qualitative. Versions of our analyses rival state of the art analyses for qualitative and quantitative information flow.

Our technical development uses the simplest programming language and semantic model in which the ideas can be exposed. One benefit of working entirely within the framework of abstract interpretation is that a wide range of semantics and analyses are already available for rich programming languages.

Outline. Following the background (Section 2), we introduce domains and Galois connections for hyperproperties (Section 3) and hypercollecting semantics (Section 4). Hyperproperties for information flow are defined in Section 5. We use the framework to derive the static analyses in Section 6 and Section 7. Section 8 uses examples to evaluate the precision of the analyses, and shows how existing analyses can be leveraged to improve precision. We discuss related work (Section 9) and conclude. *An accompanying technical report (Assaf et al. 2016a) contains detailed proofs for all results, as well as a table of symbols.*

2. Background: Collecting Semantics, Galois Connections

The formal development uses deterministic imperative programs over integer variables. Let n range over literal integers $\mathbb{Z}$, x over variables, and $\oplus$ (resp. cmp) over some arithmetic (resp. comparison) operators.

$$\begin{aligned}
&\mathbf{c} ::= \mathbf{skip} \mid x := e \mid c_1; c_2 \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{while}\ b\ \mathbf{do}\ c\\
&\mathbf{e} ::= n \mid x \mid e_1 \oplus e_2 \mid b\\
&\mathbf{b} ::= e_1\ \mathrm{cmp}\ e_2
\end{aligned}$$

Different program analyses may consider different semantic domains as needed to express a given class of program properties. For imperative programs, the usual domains are based on states $\sigma \in$ **States** that map each variable to a value (Winskel 1993). Some program properties require the use of traces that include intermediate states; others can use more abstract domains. For information flow properties involving intermediate outputs, or restricted to explicit data flow (Schoepe et al. 2016), details about intermediate steps are needed. By contrast, bounding the range of variables can be expressed in terms of final states. As another example, consider determining which variables are left unchanged: To express this, we need both initial and final states.

$$\mathcal{P}(\mathbf{States}^*) \longrightarrow \mathcal{P}(\mathbf{Trc}) \longrightarrow \mathcal{P}(\mathbf{States})$$

Figure 1. Fragment of the hierarchy of semantic domains ($\xrightarrow{\text{abstraction}}$)

In this paper we use the succinct term ***trace*** for elements of **Trc** defined by $\mathbf{Trc} \triangleq \mathbf{States} \times \mathbf{States}$, interpreting $t \in \mathbf{Trc}$ as an initial and final state. In the literature, these are known as ***relational traces***, by contrast with ***maximal trace*** semantics using the set $\mathbf{States}^*$ of finite sequences. A uniform framework describes the relationships and correspondences between these and many other semantic domains using Galois connections (Cousot 2002). Three of these domains are depicted in Figure 1.

Given partially ordered sets $\mathcal{C}, \mathcal{A}$, the monotone functions $\alpha \in \mathcal{C} \to \mathcal{A}$ and $\gamma \in \mathcal{A} \to \mathcal{C}$ comprise a ***Galois connection***, a proposition we write $(\mathcal{C}, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{A}, \sqsubseteq)$, provided they satisfy $\alpha(c) \sqsubseteq a$ iff $c \leq \gamma(a)$ for all $c \in \mathcal{C}, a \in \mathcal{A}$.

For example, to specify an analysis that determines which variables are never changed, let $\mathcal{A}$ be sets of variables. Define $\alpha \in \mathcal{P}(\mathbf{Trc}) \to \mathcal{P}(\mathrm{Vars})$ by $\alpha(T) = \{x \mid \forall(\sigma, \sigma') \in T,\ \sigma(x) = \sigma'(x)\}$ and $\gamma(X) = \{(\sigma, \sigma') \mid \forall x \in X,\ \sigma(x) = \sigma'(x)\}$. Then $(\mathcal{P}(\mathbf{Trc}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(Var), \supseteq)$.

For the hierarchy of usual domains, depicted in Figure 1, the connections are defined by an "element-wise abstraction". Define $\mathrm{elt} \in \mathbf{States}^* \to \mathbf{Trc}$ by $\mathrm{elt}(\sigma_0\sigma_1 \ldots \sigma_n) \triangleq (\sigma_0, \sigma_n)$. This lifts to an abstraction $\mathcal{P}(\mathbf{States}^*) \to \mathcal{P}(\mathbf{Trc})$.

Lemma 1 Element-wise abstraction. Let $\mathrm{elt} \in \mathcal{C} \to \mathcal{A}$ be a function between sets. Let $\alpha_{\mathrm{elt}}(C) \triangleq \{\mathrm{elt}(c) \mid c \in C\}$ and $\gamma_{\mathrm{elt}}(A) \triangleq \{c \mid \mathrm{elt}(c) \in A\}$. Then $(\mathcal{P}(\mathcal{C}), \subseteq) \underset{\alpha_{\mathrm{elt}}}{\overset{\gamma_{\mathrm{elt}}}{\leftrightarrows}} (\mathcal{P}(\mathcal{A}), \subseteq)$.

The domain $\mathcal{P}(\mathbf{States})$, which suffices to describe the final reachable states of a program, is an abstraction of the relational domain $\mathcal{P}(\mathbf{Trc})$, by $\mathrm{elt}(\sigma, \tau) \triangleq \tau$. In this paper we focus on the domain **Trc** because it is the simplest that can express dependences.

Program semantics. We define both the denotational semantics $[\![c]\!] \in \mathbf{Trc}_\perp \to \mathbf{Trc}_\perp$ of commands and the denotational semantics $[\![e]\!] \in \mathbf{Trc} \to \mathbf{Val}$ of expressions. Here $\mathbf{Val} \triangleq \mathbb{Z}$ and $\mathbf{Trc}_\perp$ adds bottom element $\perp$ using the flat ordering.

Standard semantics of commands $\qquad [\![c]\!] \in \mathbf{Trc}_\perp \to \mathbf{Trc}_\perp$

$$[\![c]\!]\perp \triangleq \perp \qquad [\![x := e]\!](\sigma, \tau) \triangleq (\sigma,\ \tau[x \mapsto [\![e]\!](\sigma, \tau)])$$

$$[\![c_1; c_2]\!]t \triangleq [\![c_2]\!] \circ [\![c_1]\!]t \qquad [\![\mathbf{skip}]\!]t \triangleq t$$

$$[\![\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2]\!]t \triangleq \begin{cases} [\![c_1]\!]t & \text{if } [\![b]\!]t = 1 \\ [\![c_2]\!]t & \text{if } [\![b]\!]t = 0 \end{cases}$$

$$[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]t \triangleq (\mathrm{lfp}^{\dot{\preccurlyeq}}_{(\lambda t.\perp)}\ \mathcal{F})(t)$$
$$\text{where } \mathcal{F}(w)(t) \triangleq \begin{cases} t & \text{if } [\![b]\!]t = 0 \\ w \circ [\![c]\!]t & \text{otherwise} \end{cases}$$

Let t be a trace (σ, τ). The denotation $[\![e]\!]t$ evaluates e in the "current state", τ. (In Sect. 5 we also use $[\![e]\!]_{\mathrm{pre}}t$ which evaluates e in the initial state, σ.) The denotation $[\![c]\!]t$ is (σ, τ') where execution of c in τ leads to τ'. The denotation is $\bot$ in case c diverges from τ. Boolean expressions evaluate to either 0 or 1. We assume programs do not go wrong. We denote by $\dot{\preccurlyeq}$ the point-wise lifting to $\mathbf{Trc}_\bot \to \mathbf{Trc}_\bot$ of the approximation order $\preccurlyeq$ on $\mathbf{Trc}_\bot$.

The terminating computations of c can be written as its image on the initial traces: $\{[\![c]\!]t \mid t \in \mathbf{IniTrc} \text{ and } [\![c]\!]t \neq \bot\}$ where

$$\mathbf{IniTrc} \triangleq \{(\sigma, \sigma) \mid \sigma \in \mathbf{States}\}$$

To specify properties that hold for all executions we use ***collecting semantics*** which lifts the denotational semantics to arbitrary sets $T \in \mathcal{P}(\mathbf{Trc})$ of traces. The idea is that $\{\!|c|\!\}T$ is the direct image of $[\![c]\!]$ on T. To be precise, in this paper we focus on termination-insensitive properties, and thus $\{\!|c|\!\}T$ is the set of non-$\bot$ traces t' such that $[\![c]\!]t = t'$ for some $t \in T$. Later we also use the collecting semantics of expressions: $\{\!|e|\!\}T \triangleq \{[\![e]\!]t \mid t \in T\}$.

Importantly, the collecting semantics $\{\!|c|\!\} \in \mathcal{P}(\mathbf{Trc}) \to \mathcal{P}(\mathbf{Trc})$ can be defined compositionally using fixpoints (Cousot 2002, Sec. 7). For conditional guard b, write $\{\!|\,\mathrm{grd}^b\,|\!\}$ for the filter defined by $\{\!|\,\mathrm{grd}^b\,|\!\}T \triangleq \{t \in T \mid [\![b]\!]t = 1\}$.

Collecting semantics $\qquad \{\!|c|\!\} \in \mathcal{P}(\mathbf{Trc}) \to \mathcal{P}(\mathbf{Trc})$

$$\{\!|x := e|\!\}T \triangleq \{[\![x := e]\!]t \mid t \in T\}$$

$$\{\!|c_1; c_2|\!\}T \triangleq \{\!|c_2|\!\} \circ \{\!|c_1|\!\}T \qquad \{\!|\mathbf{skip}|\!\}T \triangleq T$$

$$\{\!|\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2|\!\}T \;\triangleq\; \{\!|c_1|\!\} \circ \{\!|\,\mathrm{grd}^b\,|\!\}T \cup \{\!|c_2|\!\} \circ \{\!|\,\mathrm{grd}^{\neg b}\,|\!\}T$$

$$\{\!|\mathbf{while}\ b\ \mathbf{do}\ c|\!\}T \triangleq \{\!|\,\mathrm{grd}^{\neg b}\,|\!\} \left(\mathrm{lfp}^{\subseteq}_T\ \{\!|\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ \mathbf{skip}|\!\}\right)$$

The clause for while loops uses the denotation of a constructed conditional command as a definitional shorthand—its denotation is compositional.

Given a Galois connection $(\mathcal{P}(\mathbf{Trc}), \subseteq) \xleftarrow[\alpha]{\gamma} (\boldsymbol{\mathcal{A}}, \sqsubseteq)$, such as the one for unmodified variables, the desired analysis is specified as $\alpha \circ \{\!|c|\!\} \circ \gamma$. Since it is not computable in general, we only require an approximation $f^\sharp \in \boldsymbol{\mathcal{A}} \to \boldsymbol{\mathcal{A}}$ that is ***sound*** in this sense:

$$\alpha \circ \{\!|c|\!\} \circ \gamma \mathrel{\dot{\sqsubseteq}} f^\sharp \tag{1}$$

where $\dot{\sqsubseteq}$ denotes the point-wise lifting of the partial order $\sqsubseteq$.

To explain the significance of this specification, suppose one wishes to prove program c satisfies a trace property $T \in \mathcal{P}(\mathbf{Trc})$, i.e. to prove that $\{\!|c|\!\}(\mathbf{IniTrc}) \subseteq T$. Given Eq. (1) it suffices to find an abstract value a that approximates $\mathbf{IniTrc}$, i.e. $\mathbf{IniTrc} \subseteq \gamma(a)$, and show that

$$\gamma(f^\sharp(a)) \subseteq T \tag{2}$$

Eq. (1) is equivalent to $\{\!|c|\!\} \circ \gamma \mathrel{\dot{\subseteq}} \gamma \circ f^\sharp$ by a property of Galois connections. So Eq. (2) implies $\{\!|c|\!\}(\gamma(a)) \subseteq T$ which (by monotonicity of $\{\!|c|\!\}$) implies $\{\!|c|\!\}(\mathbf{IniTrc}) \subseteq \{\!|c|\!\}(\gamma(a)) \subseteq T$.

The beauty of specification Eq. (1) is that $f^\sharp$ can be obtained as an abstract interpretation $\{\!|c|\!\}^\sharp$, derived systematically for all c by calculating from the left side of Eq. (1) as shown by Cousot (1999).

3. Domains and Galois Connections for Hyperproperties

To express hyperproperties, we need Galois connections for domains that involve sets of sets of observable behaviours. This section spells out how such powerset domains form a hierarchy as illustrated along the top of Figure 2. We describe how dependences and cardinalities for quantitative information flow can be formulated as Galois connections. We spell out a methodology whereby the standard notions and techniques of abstract interpretation can be applied to specify and derive—in the same form as Equation (1)—static analyses for hyperproperties.

$$\begin{tikzcd}
\mathcal{P}(\mathcal{P}(\mathbf{States}^*)) \arrow[rr] \arrow[dr] & & \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \arrow[rr] \arrow[dr] & & \mathcal{P}(\mathcal{P}(\mathbf{States})) \arrow[dr] & \\
& \mathcal{P}(\mathbf{States}^*) \arrow[rr] & & \mathcal{P}(\mathbf{Trc}) \arrow[rr] & & \mathcal{P}(\mathbf{States})
\end{tikzcd}$$

Figure 2. Extended hierarchy of semantic domains ($\xrightarrow{\text{abstraction}}$)

As a first example, consider the condition: the final value of x depends only on the initial value of y. Its expression needs, at least, two traces: If two traces, denoted by (σ, σ') and (τ, τ'), agree on the initial value of y then they agree on the final value of x. That is, $\sigma(y) = \tau(y)$ implies $\sigma'(x) = \tau'(x)$. This must hold for any two traces of the program. This is equivalent to the following: For all sets T of traces, if traces in T all agree on the initial value of y then they all agree on the final value of x. Later we extend this example to an analysis that infers which dependences hold.

Consider the problem of quantifying information flow with min-capacity (Smith 2009). For a program on two integer variables h, l, the problem is to infer how much information is conveyed via l about h: considering some traces that agree on the initial value of l, how many final values are possible for l. For example, the program $l := (h\ mod\ 2) + l$ has two final values for l, for each initial l, though there are many possible initial values for h. This cardinality problem generalizes prior work on quantitative flow analysis, where typically low inputs are not considered.

Whereas the simple dependence problem can be formulated in terms of 2 traces, the cardinality problem involves trace sets of unbounded size. In the terminology of hyperproperties, it is not a k-safety hyperproperty for any k (Yasuoka and Terauchi 2011, Sec. 3), although it is hypersafety (Clarkson and Schneider 2010). For a fixed k, the problem "variable l has at most $k - 1$ final values" is k-safety, which means it can be formulated in terms of sets with at most k traces.

It turns out that by using Galois connections on sets of sets, we can develop a general theory that encompasses many hyperproperties and which enables derivation of interesting abstract interpreters. For our applications, we use relational traces as the notion of observable behavior, and thus $\mathcal{P}(\mathcal{P}(\mathbf{Trc}))$. The approach works as well for other notions, so there is a hierarchy of domains as shown at the top of Figure 2, in parallel with the ordinary hierarchy shown along the bottom.

The abstractions of this hierarchy are obtained by lifting each abstraction between two standard collecting semantics (Cousot 2002) to their hypercollecting versions, by element-wise abstraction (Lemma 1). For instance, Lemma 1 justifies the abstraction between $\mathcal{P}(\mathcal{P}(\mathbf{Trc}))$ and $\mathcal{P}(\mathcal{P}(\mathbf{States}))$, by lifting the abstraction between $\mathcal{P}(\mathbf{Trc})$ and $\mathcal{P}(\mathbf{States})$ (Cousot 2002, Sec. 8). Additionally, the diagonal lines in Figure 2 represent abstractions between hypercollecting semantics defined over some form of observations and the corresponding collecting semantics defined over the same observations.

Lemma 2 . Let $\boldsymbol{\mathcal{C}}$ be a set. Define $\alpha_{\mathrm{hpp}}(\mathbb{C}) \triangleq \cup_{C \in \mathbb{C}}\, C$ and $\gamma_{\mathrm{hpp}}(C) \triangleq \mathcal{P}(C)$. These form a Galois connection:

$$(\mathcal{P}(\mathcal{P}(\boldsymbol{\mathcal{C}})), \subseteq) \xleftarrow[\alpha_{\mathrm{hpp}}]{\gamma_{\mathrm{hpp}}} (\mathcal{P}(\boldsymbol{\mathcal{C}}), \subseteq)$$

It is noted by Clarkson and Schneider (2010) that any trace property can be lifted to a unique hyperproperty; this lifting is exactly the concretisation γ_{hpp} of Lemma 2. Although the model of Clarkson and Schneider (2010) is quite general, it does focus on infinite traces. But hyperproperties can be formulated in terms of other notions of observation, as illustrated in Figure 2.

Cardinality abstraction. To lay the groundwork for our quantitative information flow analysis, we consider abstracting a set of values by its cardinality. Cardinality is one ingredient in many quantitative information flow analyses estimating the amount of sensitive information a program may leak (Smith 2009; Backes et al. 2009; Braun et al. 2009; Köpf and Rybalchenko 2013; Mardziel et al. 2013; Doychev et al. 2013). The lattice of abstract representations we consider is the set

$$[0,\infty] \triangleq \mathbb{N} \cup \{\infty\}$$

where ∞ denotes an infinite cardinal number. We use the natural order $\leq$, and $\max$ as a join. Consider the abstraction operator $\mathrm{crdval} \in \mathcal{P}(\mathbf{Val}) \to [0,\infty]$ computing cardinality and given by $\mathrm{crdval}(V) \triangleq |V|$. This operator crdval is not ***additive***, i.e. it does not preserve joins; e.g. $\mathrm{crdval}(\{1,2\} \cup \{2,3\}) \neq \max(\mathrm{crdval}(\{1,2\}), \mathrm{crdval}(\{2,3\}))$. Thus, there exists no associated concretisation f for which crdval is the lower adjoint in a Galois connection. Yet, we can lift the abstraction operator crdval to a Galois connection over $\mathcal{P}(\mathcal{P}(\mathbf{Val}))$ through what is called a supremus abstraction (Cousot 2002, p.52).

Lemma 3 Supremus abstraction. Let $\mathrm{elt} \in \boldsymbol{\mathcal{C}} \to \boldsymbol{\mathcal{A}}$ be a function from a set $\boldsymbol{\mathcal{C}}$, with codomain forming a complete lattice $(\boldsymbol{\mathcal{A}}, \sqsubseteq)$. Let $\alpha_{\mathrm{elt}}(C) \triangleq \sqcup_{c \in C}\, \mathrm{elt}(c)$ and $\gamma_{\mathrm{elt}}(a) \triangleq \{c \in \boldsymbol{\mathcal{C}} \mid \mathrm{elt}(c) \sqsubseteq a\}$. Then

$$(\mathcal{P}(\boldsymbol{\mathcal{C}}), \subseteq) \underset{\alpha_{\mathrm{elt}}}{\overset{\gamma_{\mathrm{elt}}}{\longleftrightarrow}} (\boldsymbol{\mathcal{A}}, \sqsubseteq)$$

For example, define $\alpha_{\mathrm{crdval}}(\mathbb{V}) \triangleq \max_{V \in \mathbb{V}} \mathrm{crdval}(V)$ and $\gamma_{\mathrm{crdval}}(n) \triangleq \{V \mid \mathrm{crdval}(V) \leq n\}$. Thus we obtain a Galois connection $(\mathcal{P}(\mathcal{P}(\mathbf{Val})), \subseteq) \underset{\alpha_{\mathrm{crdval}}}{\overset{\gamma_{\mathrm{crdval}}}{\longleftrightarrow}} ([0,\infty], \leq)$.

As another example let us consider, in simplified form, an ingredient in dependency or noninterference analysis. For program variable x, $\mathrm{agree}_x \in \mathcal{P}(\mathbf{States}) \to \{\mathrm{tt}, \mathrm{ff}\}$ determines whether a set of states contains only states that all agree on x's value:

$$\mathrm{agree}_x(\Sigma) \triangleq (\forall \sigma, \sigma' \in \Sigma,\ [\![x]\!]\sigma = [\![x]\!]\sigma')$$

Function agree_x is not additive, so it is not part of a Galois connection from $\mathcal{P}(\mathbf{States})$ to $\{\mathrm{tt}, \mathrm{ff}\}$. The same problem arises with agreements on multiple variables, and with more concrete domains like the finite maximal trace semantics $\mathcal{P}(\mathbf{States}^*)$.

We lift the operator agree_x to a Galois connection over $\mathcal{P}(\mathcal{P}(\mathbf{States}))$. A supremus abstraction yields

$$\alpha_{\mathrm{agree}_x}(\mathbb{S}) \triangleq (\forall \Sigma \in \mathbb{S},\ \mathrm{agree}_x(\Sigma))$$
$$\gamma_{\mathrm{agree}_x}(\mathrm{bv}) \triangleq \{\Sigma \mid \mathrm{agree}_x(\Sigma) \Longleftarrow \mathrm{bv}\}$$

so that $(\mathcal{P}(\mathcal{P}(\mathbf{States})), \subseteq) \underset{\alpha_{\mathrm{agree}_x}}{\overset{\gamma_{\mathrm{agree}_x}}{\longleftrightarrow}} (\{\mathrm{tt}, \mathrm{ff}\}, \Longleftarrow)$.

These examples are consistent with the many formulations of noninterference (e.g. (Goguen and Meseguer 1982; Volpano and Smith 1997; Giacobazzi and Mastroeni 2004; Amtoft and Banerjee 2004; Hunt and Sands 2006)) that motivated the characterisation of information-flow security requirements as hyperproperties (Clarkson and Schneider 2010). Concretising an abstract value a can be seen as defining the denotation of a type expression (as in, for instance, Benton (2004, Sec. 3.3.1) and Hunt and Sands (1991)), i.e. defining the set of objects that satisfy the description a. Thus, concretising tt, when tt is interpreted as "satisfies a property requirement", naturally yields a set of traces. Concretising tt, where tt is interpreted as "satisfies a security requirement", yields a set of sets of traces.

Intuitively, the most abstract denotation/concretisation of a property requirement is defined in terms of a set of traces. The most abstract concretisation/denotation of a security requirement yields a set of sets of traces, namely a hyperproperty. Hints of this intuition appear in the literature (McLean 1994; Volpano 1999; Rushby 2001; Zakinthinos and Lerner 1997); e.g. security policies "are predicates on sets of traces (i.e. they are higher order)" (Rushby 2001, p.2). However, only recently has a comprehensive framework proposed a sharp characterisation of security policies as hyperproperties (Clarkson and Schneider 2008, 2010).

Abstract interpretation of hyperproperties. The basic methodology for the verification of a hyperproperty HP, may be described as follows:

Step 1. Design approximate representations forming a complete lattice $\boldsymbol{\mathcal{A}}$, choose a collecting semantics $\boldsymbol{\mathcal{C}}$ among the extended hierarchy (set of sets domains, e.g. $\mathcal{P}(\mathcal{P}(\mathbf{Trc}))$), and define α, γ for a Galois connection $(\boldsymbol{\mathcal{C}}, \leq) \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} (\boldsymbol{\mathcal{A}}, \sqsubseteq)$.

Step 2. Compute an approximation $a \in \boldsymbol{\mathcal{A}}$ of the semantics $C \in \boldsymbol{\mathcal{C}}$ of the program P of interest.

Step 3. Prove that the inferred approximation a implies that P satisfies HP. The concretisation $\gamma(a)$ is a set of trace sets, of which the program's trace set is a *member*—by contrast to approximations of trace properties, which infer a single trace set of which the program trace set is a *subset*. Then, it suffices to prove $\gamma(a) \subseteq \mathrm{HP}$.

Step 1 is guided by the need to have $\gamma(a) \subseteq \mathrm{HP}$, i.e. a describes a hyperproperty that implies HP. The calculational design (Cousot 1999) of abstract domains greatly systematises Step 2, by relying on the Galois connection defined in Step 1. Collecting semantics can be adapted to the additional structure of sets, as we show in Section 4.

4. Hypercollecting Semantics

In the following, we introduce a hypercollecting semantics defined over sets $\mathbb{T} \in \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$ of sets of traces. This is used in subsequent sections to derive static analyses.

Here is Step 2 of the methodology, spelled out in detail. Given a Galois connection $(\mathcal{P}(\mathcal{P}(\mathbf{Trc})), \subseteq) \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} (\boldsymbol{\mathcal{A}}, \sqsubseteq^\sharp)$ built by the supremus abstraction, and an approximation a of the initial traces (i.e. **IniTrc** is in $\gamma(a)$), find an approximation $a' \in \boldsymbol{\mathcal{A}}$ of the analysed program c, i.e. $\{\!|c|\!\}\,\mathbf{IniTrc}$ is in $\gamma(a')$. Then prove that the program satisfies the hyperproperty HP of interest, i.e. $\gamma(a') \subseteq \mathrm{HP}$. In order to compute a', we define a hypercollecting semantics $(\!|c|\!) \in \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$. That will serve to derive—in the manner of Equation (1)—a static analysis that is correct by construction.

Hypercollecting semantics $\quad (\!|c|\!) \in \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$

$$(\!|x := e|\!)\mathbb{T} \triangleq \{\{\!|x := e|\!\}T \mid T \in \mathbb{T}\}$$

$$(\!|c_1; c_2|\!)\mathbb{T} \triangleq (\!|c_2|\!) \circ (\!|c_1|\!)\mathbb{T} \qquad (\!|\mathbf{skip}|\!)\mathbb{T} \triangleq \mathbb{T}$$

$$(\!|\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2|\!)\mathbb{T} \triangleq \{\{\!|c_1|\!\} \circ \{\!|\,\mathrm{grd}^b\,|\!\}T \cup \{\!|c_2|\!\} \circ \{\!|\,\mathrm{grd}^{\neg b}\,|\!\}T \ \mid\ T \in \mathbb{T}\}$$

$$(\!|\mathbf{while}\ b\ \mathbf{do}\ c|\!)\mathbb{T} \triangleq (\!|\mathrm{grd}^{\neg b}|\!)\left(\mathrm{lfp}^{\subseteq}_{\mathbb{T}}\,(\!|\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else\ skip}|\!)\right)$$

$$(\!|\mathrm{grd}^b|\!)\mathbb{T} \triangleq \{\{\!|\,\mathrm{grd}^b\,|\!\}T \mid T \in \mathbb{T}\}$$

Recall from Section 2 that standard collecting semantics is a fixpoint-based formulation that captures the direct image on sets of the underlying program semantics – this is proved, for example, by Cachera and Pichardie (2010); Assaf and Naumann (2016). The fixpoint formulation at the level of sets-of-sets we use is not simply the direct image of the standard collecting semantics. The direct image of the standard collecting semantics would yield a set of (inner) fixpoints over sets of traces, whereas an outer fixpoint over sets of sets of traces enables straightforward application of the fixpoint transfer theorem.

Theorem 1 . For all c and all $T \in \mathcal{P}(\mathbf{Trc})$, $\{\!|c|\!\}T$ is in $(\!| c |\!)\{T\}$.

For a singleton $\{T\}$, the set $(\!|c|\!)\{T\} \in \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$ is not necessarily a singleton set containing only the element $\{\!|c|\!\}T$. If c is a loop, $(\!|c|\!)\{T\}$ yields a set of sets R of traces, where each set R of traces contains only traces that exit the loop after less than k iterations, for $k \in \mathbb{N}$. We prove this theorem as corollary of the following:

$$\forall \mathbb{T} \in \mathcal{P}(\mathcal{P}(\mathbf{Trc})), \{\{\!|c|\!\}T \mid T \in \mathbb{T}\} \subseteq (\!|c|\!)\mathbb{T}$$

This is proved by structural induction on commands. For loops, there is a secondary induction on iterations of the loop body.

In summary, suppose one wishes to prove program c satisfies hyperproperty $\mathrm{HP} \in \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$, i.e. one wishes to prove that $\{\!|c|\!\}(\mathbf{IniTrc}) \in \mathrm{HP}$. Suppose we have an approximation $f^\sharp$ of the hypercollecting semantics, similarly to Eq. (1), i.e.

$$\alpha \circ (\!|c|\!) \circ \gamma \mathrel{\dot{\sqsubseteq}^\sharp} f^\sharp \tag{3}$$

Given Eq. (3) it suffices to find an abstract value a that approximates **IniTrc**, i.e. $\mathbf{IniTrc} \in \gamma(a)$, and show that:

$$\gamma(f^\sharp(a)) \subseteq \mathrm{HP} \tag{4}$$

Why? Equation (3) is equivalent to $(\!|c|\!) \circ \gamma \mathrel{\dot{\subseteq}} \gamma \circ f^\sharp$ by a property of Galois connections. So we have $\{\!|c|\!\}(\mathbf{IniTrc}) \in (\!|c|\!)(\gamma(a)) \subseteq \gamma(f^\sharp(a)) \subseteq \mathrm{HP}$ using $\mathbf{IniTrc} \in \gamma(a)$, the Theorem, and Eq. (4).

5. Information Flow

This section gives a number of technical definitions which build up to the definition of Galois connections with which we specify information flow policies explicitly as hyperproperties.

When a fixed main program is considered, we refer to it as P and its variables as Var_P. Our analyses are parametrised by the program P to analyse, and an initial ***typing context*** $\Gamma \in \mathrm{Var}_\mathrm{P} \to \mathcal{L}$ mapping each variable to a security level $l \in \mathcal{L}$ for its initial value. We assume $(\mathcal{L}, \sqsubseteq, \sqcup, \sqcap)$ is a finite lattice. In the most concrete case, $\mathcal{L}$ may be defined as the ***universal flow lattice***, i.e. the powerset of variables $\mathcal{P}(\mathrm{Var}_\mathrm{P})$, from which all other information flow types can be inferred through a suitable abstraction (Hunt and Sands 2006, Sec. 6.2); the initial typing context is then defined as $\lambda x.\{x\}$.

Initial l-equivalence and variety. A key notion in information flow is ***l-equivalence***. Two states are l-equivalent iff they agree on the values of variables having security level at most l. We introduce the same notion over a set of traces, requiring that the *initial* states are l-equivalent. Let us first denote by $[\![e]\!]_\mathrm{pre} \in \mathbf{Trc} \to \mathbf{Val}$ the evaluation of expression e in the initial state σ of a trace $(\sigma, \tau) \in \mathbf{Trc}$—unlike $[\![e]\!] \in \mathbf{Trc} \to \mathbf{Val}$ which evaluates expression e in the final state τ. Then, we denote by $T \models_\Gamma l$ the judgement that all traces in a set $T \subseteq \mathbf{Trc}$ are ***initially l-equivalent***, i.e. they all initially agree on the value of variables up to a security level $l \in \mathcal{L}$.

For example, in the case that $\mathcal{L}$ is the universal flow lattice, $T \models_\Gamma \{x, y\}$ means $\forall t_1, t_2 \in T, [\![x]\!]_\mathrm{pre}t_1 = [\![x]\!]_\mathrm{pre}t_2 \wedge [\![y]\!]_\mathrm{pre}t_1 = [\![y]\!]_\mathrm{pre}t_2$.

Initial l-equivalence $\qquad T \models_\Gamma l$

$$T \models_\Gamma l \text{ iff. } \forall t_1, t_2 \in T, \forall x \in \mathrm{Var}_\mathrm{P},\; \Gamma(x) \sqsubseteq l \implies [\![x]\!]_\mathrm{pre}t_1 = [\![x]\!]_\mathrm{pre}t_2$$

The notion of variety (Cohen 1977) underlies most definitions of qualitative and quantitative information flow security. Information is transmitted from a to b over execution of program P if by "varying the initial value of a (exploring the variety in a), the resulting value in b after P's execution will also vary (showing that variety is conveyed to b)" (Cohen 1977). We define the **l-*variety*** of expression e, as the set of sets of values e may take, when considering only initially l-equivalent traces. The variety is defined first as a function $\mathcal{O}^l\{\!|e|\!\} \in \mathcal{P}(\mathbf{Trc}) \to \mathcal{P}(\mathcal{P}(\mathbf{Val}))$ on trace sets, from which we obtain a function $\mathcal{O}^l(\!|e|\!) \in \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathcal{P}(\mathcal{P}(\mathbf{Val}))$, on sets of trace sets. Intuitively, l-variety of expression e is the variety that is conveyed to e by varying only the input values of variables having a security level l' such that $\neg(l' \sqsubseteq l)$.

l-variety $\qquad \mathcal{O}^l\{\!|e|\!\} \qquad \mathcal{O}^l(\!|e|\!)$

$$\mathcal{O}^l\{\!|e|\!\} \in \mathcal{P}(\mathbf{Trc}) \to \mathcal{P}(\mathcal{P}(\mathbf{Val}))$$
$$\mathcal{O}^l\{\!|e|\!\}T \triangleq \{\{\!|e|\!\}R \mid R \subseteq T \text{ and } R \models_\Gamma l\}$$
$$\mathcal{O}^l(\!|e|\!) \in \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathcal{P}(\mathcal{P}(\mathbf{Val}))$$
$$\mathcal{O}^l(\!|e|\!)\mathbb{T} \triangleq \cup_{T \in \mathbb{T}}\, \mathcal{O}^l\{\!|e|\!\}T$$

Each set $V \in \mathcal{O}^l\{\!|e|\!\}T$ of values results from initially l-equivalent traces ($R \models_\Gamma l$ for $R \subseteq T$). Thus, expression e does not leak sensitive information to attackers having a security clearance $l \in \mathcal{L}$ if $\mathcal{O}^l\{\!|e|\!\}T$ is a set of singleton sets. Indeed, sensitive data for attackers with security clearance $l \in \mathcal{L}$ is all data having a security level l' for which attackers do not have access (i.e. $\neg(l' \sqsubseteq l)$ (Denning and Denning 1977)). Thus, if $\mathcal{O}^l\{\!|e|\!\}T$ is a set of singleton sets, this means that no matter how sensitive information varies, this variety is not conveyed to expression e.

Besides a pedagogical purpose, we define l-variety $\mathcal{O}^l\{\!|e|\!\}$ (resp. $\mathcal{O}^l(\!|e|\!)$) instead of simply lifting the denotational semantics $[\![e]\!]$ of expressions to sets of traces (resp. sets of sets of traces) since we want to build modular abstractions of traces by relying on underlying abstractions of values. Thus, l-variety enables us to pass information about initially l-equivalent traces to the underlying domain of values by keeping disjoint values that originate from traces that are not initially l-equivalent.

Specifying information flow. We now have the ingredients needed to describe information flow for command c, with respect to typing context $\Gamma \in \mathrm{Var}_\mathrm{P} \to \mathcal{L}$. A quantitative security metric, introduced by Smith (2009, 2011), relies on min-entropy and min-capacity (Rényi 1961) in order to estimate the leakage of a program. Let us assume a program P that is characterized by a set $T_\mathrm{P} \in \mathcal{P}(\mathbf{Trc})$ of traces, i.e. $T_\mathrm{P} \triangleq \{\!|\,\mathrm{P}\,|\!\}\, \mathbf{IniTrc}$. For simplicity, assume attackers only observe the value of a single variable $x \in \mathrm{Var}_\mathrm{P}$. (The generalization to multiple variables is straightforward.) The leakage of P, as measured by ***min-capacity***, to attackers having security clearance $l \in \mathcal{L}$ is defined by

$$\mathcal{ML}_l \triangleq \log_2 \circ\, \alpha_\mathrm{crdval} \circ \mathcal{O}^l\{\!|x|\!\}T_\mathrm{P}$$

(The definition of α_crdval follows Lemma 3.) For our purposes, it suffices to know that this quantity aims to measure, in bits, the remaining uncertainty about sensitive data for attackers with security clearance l. Refer to the original work (Smith 2009) for more details.

Leaving aside the logarithm in the definition of $\mathcal{ML}_l$, a quantitative security requirement may enforce a limit on the amount of information leaked to attackers with security clearance $l \in \mathcal{L}$, by requiring that the l-cardinality of variable x is less than or equal to some non-negative integer k. We denote by $\mathrm{SR}(l, k, x)$ the hyperproperty that characterises this security requirement, i.e. the set of program denotations satisfying it:

$$\mathrm{SR}(l, k, x) \triangleq \{T \in \mathcal{P}(\mathbf{Trc}) \mid \alpha_\mathrm{crdval} \circ \mathcal{O}^l\{\!|x|\!\}T \leq k\}$$

Note that SR implicitly depends on the choice of initial typing Γ, as does $\mathcal{O}^l\{\!|x|\!\}T$.

The termination-insensitive noninterference policy "the final value of x depends only on the initial values of variables labelled at most l" corresponds to the hyperproperty $\mathrm{SR}(l, 1, x)$. Therefore, the program P satisfies $\mathrm{SR}(l, 1, x)$ if $\alpha_{\mathrm{crdval}} \circ \mathcal{O}^l\{\!|x|\!\}T_{\mathrm{P}} \leq 1$. Let $\mathbb{T} = (\!|\mathrm{P}|\!)\{\mathbf{IniTrc}\}$. Since T_{P} is in $\mathbb{T}$ (Theorem 1), then P satisfies $\mathrm{SR}(l, 1, x)$ if $\alpha_{\mathrm{crdval}} \circ \mathcal{O}^l(\!|x|\!)\mathbb{T} \leq 1$, by monotony of α_{crdval} and by $\mathcal{O}^l\{\!|x|\!\}T_{\mathrm{P}} \subseteq \mathcal{O}^l(\!|x|\!)\mathbb{T}$ from the definition of $\mathcal{O}^l(\!|-|\!)$.

6. Dependences

We rely on abstract interpretation to derive a static analysis similar to existing ones inferring dependences (Amtoft and Banerjee 2004; Hunt and Sands 2006; Amtoft et al. 2006; Hunt and Sands 2011).

Recall that our analyses are parametrised on a security lattice $\mathcal{L}$ and program P. We denote by $l \rightsquigarrow x$ an atomic dependence constraint, with $l \in \mathcal{L}$ and $x \in \mathrm{Var}_{\mathrm{P}}$, read as "agreement up to security level l leads to agreement on x". It is an atomic pre-post contract expressing that the final value of x must only depend on initial values having at most security level l. Said otherwise, $l \rightsquigarrow x$ states the noninterference of variable x from data that is sensitive for attackers with security clearance l, i.e. all inputs having security level l' such that $\neg(l' \sqsubseteq l)$.

Dependences are similar to information flow types (Hunt and Sands 2006) and are the dual of independences assertions (Amtoft and Banerjee 2004). Both interpretations are equivalent (Hunt and Sands 2006, Sec. 5).

Lattice of dependence constraints $\quad \mathrm{Dep} \quad \mathscr{D} \in \mathrm{Dep}$

Given a lattice $\mathcal{L}$ and program P, define

$$\mathrm{Dep} \triangleq \mathcal{P}(\{l \rightsquigarrow x \mid l \in \mathcal{L}, x \in \mathrm{Var}_{\mathrm{P}}\})$$

$$\mathscr{D}_1 \sqsubseteq^\natural \mathscr{D}_2 \triangleq \mathscr{D}_1 \supseteq \mathscr{D}_2 \qquad \mathscr{D}_1 \sqcup^\natural \mathscr{D}_2 \triangleq \mathscr{D}_1 \cap \mathscr{D}_2$$

In the rest of this section, $\mathcal{L}$ and P are fixed, together with a typing context $\Gamma \in \mathrm{Var}_{\mathrm{P}} \to \mathcal{L}$.

The semantic characterisation of dependences is tightly linked to variety. An atomic constraint $l \rightsquigarrow x$ holds if no variety is conveyed to x when the inputs up to security level l are fixed. We use this intuition to define the Galois connections linking the hypercollecting semantics and the lattice Dep, by instantiating the supremus abstraction in Lemma 3.

The agreement abstraction approximates a set $\mathbb{V} \in \mathcal{P}(\mathcal{P}(\mathbf{Val}))$ by determining whether it contains variety.

Agreements abstraction $\quad \mathrm{agree} \quad \alpha_{\mathrm{agree}} \quad \gamma_{\mathrm{agree}}$

$$\begin{array}{lcl}
\mathrm{agree} & \in & \mathcal{P}(\mathbf{Val}) \to \{\mathrm{tt}, \mathrm{ff}\} \\
\mathrm{agree}(V) & \triangleq & (\forall v_1, v_2 \in V, v_1 = v_2) \\
\alpha_{\mathrm{agree}} & \in & \mathcal{P}(\mathcal{P}(\mathbf{Val})) \to \{\mathrm{tt}, \mathrm{ff}\} \\
\alpha_{\mathrm{agree}}(\mathbb{V}) & \triangleq & \wedge_{V \in \mathbb{V}}\, \mathrm{agree}(V) \\
\gamma_{\mathrm{agree}} & \in & \{\mathrm{tt}, \mathrm{ff}\} \to \mathcal{P}(\mathcal{P}(\mathbf{Val})) \\
\gamma_{\mathrm{agree}}(\mathrm{bv}) & \triangleq & \{V \in \mathcal{P}(\mathbf{Val}) \mid \mathrm{agree}(V) \Longleftarrow \mathrm{bv}\}
\end{array}$$

$$(\mathcal{P}(\mathcal{P}(\mathbf{Val})), \subseteq) \underset{\alpha_{\mathrm{agree}}}{\overset{\gamma_{\mathrm{agree}}}{\longleftrightarrow}} (\{\mathrm{tt}, \mathrm{ff}\}, \Longleftarrow)$$

Note that $\gamma_{\mathrm{agree}}(\mathrm{tt})$ is $\{V \in \mathcal{P}(\mathbf{Val}) \mid \mathrm{agree}(V)\}$ and $\gamma_{\mathrm{agree}}(\mathrm{ff})$ is $\mathcal{P}(\mathbf{Val})$. Also, $\mathrm{agree}(V)$ iff $|V| \leq 1$.

The dependence abstraction approximates a set $\mathbb{T} \in \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$ by a dependence constraint $\mathscr{D} \in \mathrm{Dep}$. Recall that $\mathcal{O}^l\{\!|x|\!\}T$ is the set of final values for variable x in traces $t \in T$ that agree on inputs of level at most l. So $\alpha_{\mathrm{agree}}(\mathcal{O}^l\{\!|x|\!\}T)$ holds just if there is at most one final value.

Dependence abstraction $\quad \mathrm{deptr} \quad \alpha_{\mathrm{deptr}} \quad \gamma_{\mathrm{deptr}}$

$$\begin{array}{lcl}
\mathrm{deptr} & \in & \mathcal{P}(\mathbf{Trc}) \to \mathrm{Dep} \\
\mathrm{deptr}(T) & \triangleq & \{l \rightsquigarrow x \mid l \in \mathcal{L},\, x \in \mathrm{Var}_{\mathrm{P}},\, \alpha_{\mathrm{agree}}(\mathcal{O}^l\{\!|x|\!\}T)\} \\
\alpha_{\mathrm{deptr}} & \in & \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathrm{Dep} \\
\alpha_{\mathrm{deptr}}(\mathbb{T}) & \triangleq & \sqcup^\natural_{T \in \mathbb{T}}\, \mathrm{deptr}(T) \\
\gamma_{\mathrm{deptr}} & \in & \mathrm{Dep} \to \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \\
\gamma_{\mathrm{deptr}}(\mathscr{D}) & \triangleq & \{T \mid \mathrm{deptr}(T) \sqsubseteq^\natural \mathscr{D}\}
\end{array}$$

$$(\mathcal{P}(\mathcal{P}(\mathbf{Trc})), \subseteq) \underset{\alpha_{\mathrm{deptr}}}{\overset{\gamma_{\mathrm{deptr}}}{\longleftrightarrow}} (\mathrm{Dep}, \sqsubseteq^\natural)$$

Note that $\mathrm{deptr}(T)$ is the set of dependences $l \rightsquigarrow x$ for which $\alpha_{\mathrm{agree}}(\mathcal{O}^l\{\!|x|\!\}T)$ holds. For instance, the initial typing context $\Gamma \in \mathrm{Var}_{\mathrm{P}} \to \mathcal{L}$ determines the initial dependences of a program:

$$\begin{aligned}
&\alpha_{\mathrm{deptr}}(\{\mathbf{IniTrc}\}) \\
&\quad = \{l \rightsquigarrow x \mid l \in \mathcal{L},\, x \in \mathrm{Var}_{\mathrm{P}} \text{ and } \alpha_{\mathrm{agree}}(\mathcal{O}^l\{\!|x|\!\}\,\mathbf{IniTrc})\} \\
&\quad = \{l \rightsquigarrow x \mid l \in \mathcal{L}, x \in \mathrm{Var}_{\mathrm{P}} \text{ and } \Gamma(x) \sqsubseteq l\}
\end{aligned}$$

We derive an approximation $\mathcal{O}^l_D(\!|e|\!)^\natural$ of l-variety $\mathcal{O}^l(\!|e|\!)$. This approximation $\mathcal{O}^l_D(\!|e|\!)^\natural \in \mathrm{Dep} \to \{\mathrm{tt}, \mathrm{ff}\}$, called l-agreement of expression e, determines whether a set $\mathscr{D}$ of dependence constraints guarantees that no variety is conveyed to expression e when the inputs up to security level l are fixed. Notice that we use symbol $\natural$ and subscript D here, for contrast with similar notation using $\sharp$ and subscript C in later sections.

l-agreement of expressions $\quad \mathcal{O}^l_D(\!|e|\!)^\natural \in \mathrm{Dep} \to \{\mathrm{tt}, \mathrm{ff}\}$

$$\mathcal{O}^l_D(\!|n|\!)^\natural\mathscr{D} \triangleq \mathrm{tt} \qquad \mathcal{O}^l_D(\!|x|\!)^\natural\mathscr{D} \triangleq (l \rightsquigarrow x \in \mathscr{D})$$

$$\mathcal{O}^l_D(\!|e_1 \oplus e_2|\!)^\natural\mathscr{D} \triangleq \mathcal{O}^l_D(\!|e_1|\!)^\natural\mathscr{D} \wedge \mathcal{O}^l_D(\!|e_2|\!)^\natural\mathscr{D}$$

$$\mathcal{O}^l_D(\!|e_1 \mathrel{\mathrm{cmp}} e_2|\!)^\natural\mathscr{D} \triangleq \mathcal{O}^l_D(\!|e_1|\!)^\natural\mathscr{D} \wedge \mathcal{O}^l_D(\!|e_2|\!)^\natural\mathscr{D}$$

Deriving the clauses defining $\mathcal{O}^l_D(\!|-|\!)^\natural$ amounts to a constructive proof of the following.

Lemma 4 . $\mathcal{O}^l_D(\!|e|\!)^\natural$ is sound:

$$\forall e, \forall l, \forall \mathscr{D}, \quad \alpha_{\mathrm{agree}} \circ \mathcal{O}^l(\!|e|\!) \circ \gamma_{\mathrm{deptr}}(\mathscr{D}) \Longleftarrow \mathcal{O}^l_D(\!|e|\!)^\natural\mathscr{D} \ .$$

Dependence abstract semantics. We derive a dependence abstract semantics $(\!|c|\!)^\natural$ by approximating the hypercollecting semantics $(\!|c|\!)$. This abstract semantics $(\!|c|\!)^\natural \in \mathrm{Dep} \to \mathrm{Dep}$ over-approximates the dependence constraints that hold after execution of a command c, on inputs satisfying initial dependence constraints.

We assume a static analysis approximating the variables that a command modifies.

Modifiable variables $\quad \mathrm{Mod} \in \mathit{Com} \to \mathcal{P}(\mathit{Var})$

For all c, x, if there exists $t, t' \in \mathbf{Trc}$ such that $[\![c]\!]t = t'$ and $[\![x]\!]_{\mathrm{pre}}t' \neq [\![x]\!]t'$, then $x \in \mathrm{Mod}(c)$.

The abstract semantics of assignments $x := e$ discards all atomic constraints related to variable x in the input set $\mathscr{D}$ of constraints, and adds atomic constraints $l \rightsquigarrow x$ if $\mathscr{D}$ guarantees l-agreement for expression e. For conditionals, for each security level l, if the input set $\mathscr{D}$ guarantees l-agreement of the conditional guard, the abstract semantics computes the join over the dependences of both conditional branches, after projecting to only those atomic constraints related to l (notation $\pi^l(-)$). If $\mathscr{D}$ does not guarantee l-agreement of the conditional guard, atomic constraints related to both l and variables possibly modified are discarded. Intuitively, if $\mathscr{D}$ guarantees l-agreement of the conditional guard, then l-agreement over some variable x in both branches guarantees l-agreement over x after the conditional command. Otherwise, the only l-agreements that are guaranteed after the conditional are those that hold before the conditional for variables that are not modified.

Dependence abstract semantics $\quad (\!|c|\!)^\natural \in \mathrm{Dep} \to \mathrm{Dep}$

$$(\!|\textbf{skip}|\!)^\natural \mathscr{D} \triangleq \mathscr{D} \qquad (\!|c_1; c_2|\!)^\natural \mathscr{D} \triangleq (\!|c_2|\!)^\natural \circ (\!|c_1|\!)^\natural \mathscr{D}$$

$$(\!|x := e|\!)^\natural \mathscr{D} \triangleq \{l \rightsquigarrow y \in \mathscr{D} \mid y \neq x\} \cup \{l \rightsquigarrow x \mid l \in \mathcal{L},\ \mathcal{O}^l_D (\!|e|\!)^\natural \mathscr{D}\}$$

$$\begin{aligned}
&(\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\natural \mathscr{D} \triangleq\\
&\quad \text{let } \mathscr{D}_1 = (\!|c_1|\!)^\natural \mathscr{D} \text{ in}\\
&\quad \text{let } \mathscr{D}_2 = (\!|c_2|\!)^\natural \mathscr{D} \text{ in}\\
&\quad \text{let } W = \mathrm{Mod}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) \text{ in}\\
&\quad \bigcup_{l \in \mathcal{L}} \begin{cases} \pi^l(\mathscr{D}_1) \sqcup^\natural \pi^l(\mathscr{D}_2) & \text{if } \mathcal{O}^l_D (\!|b|\!)^\natural \mathscr{D} \\ \{l \rightsquigarrow x \in \pi^l(\mathscr{D}) \mid x \notin W\} & \text{otherwise} \end{cases}
\end{aligned}$$

$$(\!|\textbf{while } b \textbf{ do } c|\!)^\natural \mathscr{D} \triangleq \mathrm{lfp}^{\sqsubseteq^\sharp}_{\mathscr{D}} (\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\natural$$

$$\pi^l(\mathscr{D}) \triangleq \{l \rightsquigarrow x \in \mathscr{D} \mid x \in \mathrm{Var}_\mathrm{P}\}$$

Theorem 2 . The dependence semantics is sound:

$$\alpha_{\mathrm{deptr}} \circ (\!|c|\!) \circ \gamma_{\mathrm{deptr}} \mathrel{\dot{\sqsubseteq}^\natural} (\!|c|\!)^\natural \ .$$

We denote by $\dot{\sqsubseteq}^\natural$ the point-wise lifting of the partial order $\sqsubseteq^\natural$. We can derive this abstract semantics by directly approximating the relational hypercollecting semantics $(\!|c|\!)$ through the dependence Galois connection $(\alpha_{\mathrm{deptr}}, \gamma_{\mathrm{deptr}})$. The derivation is by structural induction on commands. It leverages mathematical properties of Galois connections. We start with the specification of the best abstract transformer $\alpha_{\mathrm{deptr}} \circ (\!|c|\!) \circ \gamma_{\mathrm{deptr}} \in \mathrm{Dep} \to \mathrm{Dep}$, and successively approximate it to finally obtain the definition of the dependence abstract semantics for each form of command. The derivation is the proof, and the obtained definition of the abstract semantics is correct by construction.

Let us showcase the simplest derivation for a sequence of commands in order to illustrate this process:

$$\begin{aligned}
&\alpha_{\mathrm{deptr}} \circ (\!|c_1; c_2|\!) \circ \gamma_{\mathrm{deptr}}\\
&\quad = \wr\text{By definition of the hypercollecting semantics}\wr\\
&\quad\ \ \alpha_{\mathrm{deptr}} \circ (\!|c_2|\!) \circ (\!|c_1|\!) \circ \gamma_{\mathrm{deptr}}\\
&\quad \dot{\sqsubseteq}^\natural \wr\text{By } \gamma_{\mathrm{deptr}} \circ \alpha_{\mathrm{deptr}} \text{ is extensive }\wr\\
&\quad\ \ \alpha_{\mathrm{deptr}} \circ (\!|c_2|\!) \circ \gamma_{\mathrm{deptr}} \circ \alpha_{\mathrm{deptr}} \circ (\!|c_1|\!) \circ \gamma_{\mathrm{deptr}}\\
&\quad \dot{\sqsubseteq}^\natural \wr\text{By induction hypothesis } \alpha_{\mathrm{deptr}} \circ (\!|c|\!) \circ \gamma_{\mathrm{deptr}} \mathrel{\dot{\sqsubseteq}^\natural} (\!|c|\!)^\natural\wr\\
&\quad\ \ (\!|c_2|\!)^\natural \circ (\!|c_1|\!)^\natural\\
&\quad \triangleq \wr\text{Take this last approximation as the definition.}\wr\\
&\quad\ \ (\!|c_1; c_2|\!)^\natural
\end{aligned}$$

Alternatively, we can leverage Galois connections to give the analysis as an approximation of the cardinality analysis. We work this out by Lemmas 6 and 7, introduced in Section 7.

Comparison with previous analyses. Our dependence analysis is similar to the logic of Amtoft and Banerjee (2004) as well as the flow-sensitive type system of Hunt and Sands (2006). The relationship between our sets $\mathscr{D} \in \mathrm{Dep}$ of dependence constraints and the type environments $\Delta \in \mathrm{Var}_\mathrm{P} \to \mathcal{L}$ of Hunt and Sands can be formalised by the abstraction:

$$\begin{aligned}
\alpha_{\mathrm{hs}} &\in \mathrm{Dep} \to \mathrm{Var}_\mathrm{P} \to \mathcal{L}\\
\alpha_{\mathrm{hs}}(\mathscr{D}) &\triangleq \lambda x. \sqcap \{l \mid l \rightsquigarrow x \in \mathscr{D}\}\\
\gamma_{\mathrm{hs}} &\in (\mathrm{Var}_\mathrm{P} \to \mathcal{L}) \to \mathrm{Dep}\\
\gamma_{\mathrm{hs}}(\Delta) &\triangleq \{l \rightsquigarrow x \mid x \in \mathrm{Var}_\mathrm{P},\ l \in \mathcal{L},\ \Delta(x) \sqsubseteq l\}
\end{aligned}$$

This is in fact an isomorphism because of the way we interpret dependences. Indeed, if $l \rightsquigarrow x$ holds, then also $l' \rightsquigarrow x$ for all $l' \in \mathcal{L}$ such that $l \sqsubseteq l'$ (cf. report (Assaf et al. 2016a)). This observation suggests reformulating the sets $\mathscr{D} \in \mathrm{Dep}$ of dependence constraints to contain only elements with minimal level, but we refrain from doing so for simplicity of presentation.

Our dependence analysis is at least as precise as the type system of Hunt and Sands. To state this result, we denote by $\bot_\mathcal{L}$ the bottom element of the lattice $\mathcal{L}$. We also assume that the modified variables is precise enough to simulate the same effect as the program counter used in the type system: $\mathrm{Mod}(c)$ is a subset of the variables that are targets of assignments in c.

Theorem 3 . For all c, $\mathscr{D}_0, \mathscr{D} \in \mathrm{Dep}$, $\Delta_0, \Delta \in \mathrm{Var}_\mathrm{P} \to \mathcal{L}$, where $\bot_\mathcal{L} \vdash \Delta_0\{c\}\Delta$, and $\mathscr{D} = (\!|c|\!)^\natural \mathscr{D}_0$, it holds that:

$$\alpha_{\mathrm{hs}}(\mathscr{D}_0) \mathrel{\dot{\sqsubseteq}} \Delta_0 \implies \alpha_{\mathrm{hs}}(\mathscr{D}) \mathrel{\dot{\sqsubseteq}} \Delta \ .$$

7. Cardinality Abstraction

Dependence analysis is only concerned with whether variety is conveyed. We refine this analysis by deriving a cardinality abstraction that enumerates variety.

We denote by $l \rightsquigarrow x \# n$ an atomic cardinality constraint where $l \in \mathcal{L}, x \in \mathrm{Var}_\mathrm{P}$ and $n \in [0, \infty]$, read as "agreement up to security level l leads to a variety of at most n values in variable x".

Lattice of cardinality constraints $\quad \mathrm{Card} \quad \mathscr{C} \in \mathrm{Card}$

For a program P and lattice $\mathcal{L}$, we say $\mathscr{C}$ is a ***valid set of constraints*** iff $\forall x \in \mathrm{Var}_\mathrm{P}, \forall l \in \mathcal{L}, \exists! n \in [0, \infty], l \rightsquigarrow x \# n \in \mathscr{C}$.

Let Card be the set of valid sets of constraints.

It is a complete lattice:

$$\begin{aligned}
\mathscr{C}_1 \sqsubseteq^\sharp \mathscr{C}_2 \text{ iff } &\ \forall l \rightsquigarrow x \# n_1 \in \mathscr{C}_1,\ \exists n_2,\\
&\quad l \rightsquigarrow x \# n_2 \in \mathscr{C}_2 \wedge n_1 \leq n_2\\
\mathscr{C}_1 \sqcup^\sharp \mathscr{C}_2 \triangleq &\ \{l \rightsquigarrow x \# \max(n_1, n_2) \mid\\
&\quad l \rightsquigarrow x \# n_1 \in \mathscr{C}_1,\ l \rightsquigarrow x \# n_2 \in \mathscr{C}_2\}
\end{aligned}$$

In the rest of this section, $\mathcal{L}$ and P are fixed, together with a typing context $\Gamma \in \mathrm{Var}_\mathrm{P} \to \mathcal{L}$.

A valid constraint set is essentially a function from l and x to n. So $\sqsubseteq^\sharp$ is essentially a pointwise order on functions, and we ensure that $\sqsubseteq^\sharp$ is antisymmetric.

The cardinality abstraction relies on the abstraction α_{crdval}, introduced in Section 3, in order to approximate l-variety of a variable into a cardinality $n \in [0, \infty]$.

Cardinality abstraction $\quad \mathrm{crdtr} \quad \alpha_{\mathrm{crdtr}} \quad \gamma_{\mathrm{crdtr}}$

$$\begin{array}{lll}
\mathrm{crdtr} & \in & \mathcal{P}(\mathbf{Trc}) \to \mathrm{Card} \\
\mathrm{crdtr}(T) & \triangleq & \{l \rightsquigarrow x\#n \mid \ l \in \mathcal{L},\, x \in \mathrm{Var}_{\mathrm{P}}, \\
 & & \qquad n = \alpha_{\mathrm{crdval}}(\mathcal{O}^l \{\!|x|\!\} T)\ \} \\
\alpha_{\mathrm{crdtr}} & \in & \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \to \mathrm{Card} \\
\alpha_{\mathrm{crdtr}}(\mathbb{T}) & \triangleq & \sqcup^{\sharp}{}_{T \in \mathbb{T}}\, \mathrm{crdtr}(T) \\
\gamma_{\mathrm{crdtr}} & \in & \mathrm{Card} \to \mathcal{P}(\mathcal{P}(\mathbf{Trc})) \\
\gamma_{\mathrm{crdtr}}(\mathscr{C}) & \triangleq & \{T \mid \mathrm{crdtr}(T) \sqsubseteq^{\sharp} \mathscr{C}\}
\end{array}$$

$$(\mathcal{P}(\mathcal{P}(\mathbf{Trc})), \subseteq) \underset{\alpha_{\mathrm{crdtr}}}{\overset{\gamma_{\mathrm{crdtr}}}{\leftrightarrows}} (\mathrm{Card}, \sqsubseteq^{\sharp})$$

The cardinality abstraction enables us to derive an approximation $\mathcal{O}^l_C(\!|e|\!)^{\sharp}$ of l-variety $\mathcal{O}^l(\!|e|\!)$. This approximation $\mathcal{O}^l_C(\!|e|\!)^{\sharp} \in \mathrm{Card} \to [0, \infty]$, called l-cardinality of expression e, enumerates the l-variety conveyed to expression e assuming a set $\mathscr{C} \in \mathrm{Card}$ of cardinality constraints holds. Note that the infinite cardinal ∞ is absorbing, i.e. $\forall n, \infty \times n \triangleq \infty$.

l-cardinality of expressions $\quad \mathcal{O}^l_C(\!|e|\!)^{\sharp} \in \mathrm{Card} \to [0, \infty]$

$$\mathcal{O}^l_C(\!|n|\!)^{\sharp}\mathscr{C} \triangleq 1 \qquad \mathcal{O}^l_C(\!|x|\!)^{\sharp}\mathscr{C} \triangleq n \text{ where } l \rightsquigarrow x\#n \in \mathscr{C}$$

$$\mathcal{O}^l_C(\!|e_1 \oplus e_2|\!)^{\sharp}\mathscr{C} \triangleq \mathcal{O}^l_C(\!|e_1|\!)^{\sharp}\mathscr{C} \times \mathcal{O}^l_C(\!|e_2|\!)^{\sharp}\mathscr{C}$$

$$\mathcal{O}^l_C(\!|e_1 \text{ cmp } e_2|\!)^{\sharp}\mathscr{C} \triangleq \min\left(2, \mathcal{O}^l_C(\!|e_1|\!)^{\sharp}\mathscr{C} \times \mathcal{O}^l_C(\!|e_2|\!)^{\sharp}\mathscr{C}\right)$$

Lemma 5 . $\mathcal{O}^l_C(\!|e|\!)^{\sharp}$ is sound:

$$\forall e, \forall l, \quad \alpha_{\mathrm{crdval}} \circ \mathcal{O}^l(\!|e|\!) \circ \gamma_{\mathrm{crdtr}} \mathrel{\dot{\leq}} \mathcal{O}^l_C(\!|e|\!)^{\sharp} \ .$$

We now derive a cardinality abstract semantics by approximating the relational hypercollecting semantics of Section 4. It uses definitions to follow.

Cardinality abstract semantics $\quad (\!|c|\!)^{\sharp} \in \mathrm{Card} \to \mathrm{Card}$

$$(\!|\mathbf{skip}|\!)^{\sharp}\mathscr{C} \triangleq \mathscr{C} \qquad (\!|c_1; c_2|\!)^{\sharp}\mathscr{C} \triangleq (\!|c_2|\!)^{\sharp} \circ (\!|c_1|\!)^{\sharp}\mathscr{C}$$

$$\begin{array}{l}
(\!|x := e|\!)^{\sharp}\mathscr{C} \triangleq \\
\quad \{l \rightsquigarrow y\#n \in \mathscr{C} \mid y \neq x\} \\
\quad \cup \{l \rightsquigarrow x\#n \mid l \in \mathcal{L},\, x \in \mathrm{Var}_{\mathrm{P}},\, n = \mathcal{O}^l_C(\!|e|\!)^{\sharp}\mathscr{C}\ \}
\end{array}$$

$$\begin{array}{l}
(\!|\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2|\!)^{\sharp}\mathscr{C} \triangleq \\
\quad \text{let } \mathscr{C}_1 = (\!|c_1|\!)^{\sharp}\mathscr{C} \text{ in} \\
\quad \text{let } \mathscr{C}_2 = (\!|c_2|\!)^{\sharp}\mathscr{C} \text{ in} \\
\quad \text{let } W = \mathrm{Mod}(\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2) \text{ in} \\
\quad \bigcup_{l \in \mathcal{L}} \begin{cases} \pi^l(\mathscr{C}_1) \sqcup^{\sharp} \pi^l(\mathscr{C}_2) & \text{if } \mathcal{O}^l_C(\!|b|\!)^{\sharp}\mathscr{C} = 1 \\ \pi^l(\mathscr{C}_1) \sqcup^{\sharp}{}_{\mathrm{add}(W, \pi^l(\mathscr{C}))}\, \pi^l(\mathscr{C}_2) & \text{otherwise} \end{cases}
\end{array}$$

$$(\!|\mathbf{while}\ b\ \mathbf{do}\ c|\!)^{\sharp}\mathscr{C} \triangleq \mathrm{lfp}^{\sqsubseteq^{\sharp}}_{\mathscr{C}}\ (\!|\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2|\!)^{\sharp}$$

$$\begin{array}{l}
\pi^l(\mathscr{C}) \triangleq \{l \rightsquigarrow x\#n \in \mathscr{C} \mid x \in \mathrm{Var}_{\mathrm{P}}, n \in [0, \infty]\} \\
C_1 \sqcup^{\sharp}{}_{\mathrm{add}(W, C_0)}\, C_2 \triangleq \ \bigcup_{x \in \mathrm{Var}_{\mathrm{P}} \setminus W} \{l \rightsquigarrow x\#n \in C_0\} \\
\qquad \cup \bigcup_{x \in W} \ \{l \rightsquigarrow x\#(n_1 {+} n_2) \mid \\
\qquad\qquad l \rightsquigarrow x\#n_j \in C_j,\ j = 1, 2\}
\end{array}$$

The abstract semantics of assignments $x := e$ is similar in spirit to the one for dependences: discard atomic constraints related to x, and add new ones by computing l-cardinality of expression e. The abstract semantics of conditionals is also similar to dependences: if the conditional guard does not convey l-variety, then all initially l-equivalent traces follow the same execution path and the join operator (defined as $\max$ over cardinality) over both conditional branches over-approximates the l-cardinality after the conditional. Otherwise, the l-cardinality over both conditional branches have to be summed—for the variables that may be modified in the conditional branches—to soundly approximate the l-cardinality after the conditional.

Theorem 4 . The cardinality abstract semantics is sound:

$$\alpha_{\mathrm{crdtr}} \circ (\!|c|\!) \circ \gamma_{\mathrm{crdtr}} \mathrel{\dot{\sqsubseteq}^{\sharp}} (\!|c|\!)^{\sharp} \ .$$

The lattice Card is complete, although not finite. We may define a widening operator $\nabla \in \mathrm{Card} \times \mathrm{Card} \to \mathrm{Card}$ to ensure convergence of the analysis (Cousot and Cousot 1992)(Nielson et al. 1999)(Cortesi and Zanioli 2011, Sec. 4).

$$\begin{array}{l}
\mathscr{C}_1 \nabla \mathscr{C}_2 \triangleq \{l \rightsquigarrow x\#n \mid \ l \rightsquigarrow x\#n_1 \in \mathscr{C}_1,\, l \rightsquigarrow x\#n_2 \in \mathscr{C}_2, \\
\qquad\qquad n = n_1 \nabla n_2\} \\
n_1 \nabla n_2 \triangleq \mathbf{if}\ (n_2 \leq n_1)\ \mathbf{then}\ n_1\ \mathbf{else}\ \infty
\end{array}$$

The occurrence of widening depends on the iteration strategy employed by the static analyser. Widening accelerates or forces the convergence of fixpoint computations. In the simplest setting, the analyser passes as arguments to the widening operator the old set $\mathscr{C}_1$ of cardinality as well as the new set $\mathscr{C}_2$ that is computed. For each atomic cardinality constraint, the widening operator then compares the old cardinality n_1 to the new cardinality n_2. If the cardinality is still strictly increasing ($n_2 > n_1$), the widening forces the convergence by setting it to ∞. If the cardinality is decreasing, the widening operator sets it to the maximum cardinality n_1 in order to force convergence and ensure the sequence of computed cardinalities is stationary.

Min-capacity leakage. So far, we showed how one can derive static analyses of hyperproperties—the abstract representations themselves are interpreted as hyperproperties—by approximating hypercollecting semantics. Let us now recall the security requirement $\mathrm{SR}(l, k, x)$ introduced in Section 4 in order to illustrate how these analyses may prove that a program satisfies a hyperproperty, i.e. Step 3 of the methodology in Section 3 (see also Equation (4)).

Consider a program P characterised by a set $T_{\mathrm{P}} \in \mathcal{P}(\mathbf{Trc})$ of traces, i.e. T_{P} is $\{\!|\ \mathrm{P}\ |\!\}\ \mathbf{IniTrc}$. How do we prove that P satisfies the hyperproperty $\mathrm{SR}(l, k, x)$? We can use the cardinality analysis to prove that variable x has a l-cardinality that is at most k. Indeed, if $\mathscr{C}$ approximates T_{P} (i.e. $\alpha_{\mathrm{crdtr}}(\{T_{\mathrm{P}}\}) \sqsubseteq^{\sharp} \mathscr{C}$) then $\alpha_{\mathrm{crdval}} \circ \mathcal{O}^l \{\!|x|\!\} T_{\mathrm{P}} \leq \mathcal{O}^l_C(\!|x|\!)^{\sharp}\mathscr{C}$. Thus, if the inferred l-cardinality of $\mathscr{C}$ is at most k then program P is guaranteed to satisfy the hyperproperty $\mathrm{SR}(l, k, x)$. We have $\{T_{\mathrm{P}}\} \subseteq \gamma_{\mathrm{crdtr}}(\mathscr{C})$ since $\mathscr{C}$ approximates T_{P} (i.e. $\alpha_{\mathrm{crdtr}}(\{T_{\mathrm{P}}\}) \sqsubseteq^{\sharp} \mathscr{C}$). And we have $\gamma_{\mathrm{crdtr}}(\mathscr{C}) \subseteq \mathrm{SR}(l, k, x)$ by assumption $\mathcal{O}^l_C(\!|x|\!)^{\sharp}\mathscr{C} \leq k$. Hence $T_{\mathrm{P}} \in \mathrm{SR}(l, k, x)$.

The hyperproperty $\mathrm{SR}(l, k, x)$ is a $(\mathbf{k+1})$***-safety hyperproperty*** (Clarkson and Schneider 2010), i.e. it requires exhibiting at most $k + 1$ traces in order to prove that a program does not satisfy $\mathrm{SR}(l, k, x)$. For example, termination-insensitive noninterference for security level l, which corresponds to the hyperproperty $\mathrm{SR}(l, 1, x)$, is 2-safety. A k-safety hyperproperty of a program can be reduced to a safety property of a k-fold product program (Barthe et al. 2004; Terauchi and Aiken 2005; Darvas et al. 2005; Clarkson and Schneider 2010).

Various quantitative information flow properties are not k-safety. For example, the bounding problem that the cardinality analysis targets, namely min-capacity leakage is not a k-safety hyperproperty for any k (Yasuoka and Terauchi 2011, Sec. 3). Instead, this bounding problem is hypersafety (Clarkson and Schneider 2010).

Cardinalities vs. dependences. Just as quantitative security metrics are the natural generalisations of qualitative metrics such as non-interference, the cardinality abstraction is a natural generalisation of dependence analysis. Instead of deciding if variety is conveyed, the cardinality analysis enumerates this variety. In other words, dependences are abstractions of cardinalities. We can factor the Galois connections, e.g. $(\alpha_{\text{agree}}, \gamma_{\text{agree}})$ is $(\alpha_{\text{lqone}} \circ \alpha_{\text{crdval}}, \gamma_{\text{crdval}} \circ \gamma_{\text{lqone}})$ for suitable $(\alpha_{\text{lqone}}, \gamma_{\text{lqone}})$.

Lemma 6 . $(\alpha_{\text{agree}}, \gamma_{\text{agree}})$ is the composition of two Galois connections $(\alpha_{\text{crdval}}, \gamma_{\text{crdval}})$ and $(\alpha_{\text{lqone}}, \gamma_{\text{lqone}})$:

$$(\mathcal{P}(\mathcal{P}(\mathbf{Val})), \subseteq) \underset{\alpha_{\text{crdval}}}{\overset{\gamma_{\text{crdval}}}{\longleftrightarrow}} ([0, \infty], \leq) \underset{\alpha_{\text{lqone}}}{\overset{\gamma_{\text{lqone}}}{\longleftrightarrow}} (\{\text{tt}, \text{ff}\}, \Longleftarrow)$$

with:

$$\alpha_{\text{lqone}}(n) \triangleq \begin{cases} \text{tt} & \text{if } n \leq 1 \\ \text{ff} & \text{otherwise.} \end{cases}, \text{ and}$$

$$\gamma_{\text{lqone}}(\text{bv}) \triangleq \begin{cases} 1 & \text{if bv} = \text{tt} \\ \infty & \text{otherwise.} \end{cases}$$

Lemma 7 . $(\alpha_{\text{deptr}}, \gamma_{\text{deptr}})$ is the composition of two Galois connections $(\alpha_{\text{crdtr}}, \gamma_{\text{crdtr}})$ and $(\alpha_{\text{lqonecc}}, \gamma_{\text{lqonecc}})$:

$$(\mathcal{P}(\mathcal{P}(\mathbf{Trc})), \subseteq) \underset{\alpha_{\text{crdtr}}}{\overset{\gamma_{\text{crdtr}}}{\longleftrightarrow}} (\text{Card}, \sqsubseteq^\sharp) \underset{\alpha_{\text{lqonecc}}}{\overset{\gamma_{\text{lqonecc}}}{\longleftrightarrow}} (\text{Dep}, \sqsubseteq^\natural)$$

with:

$$\alpha_{\text{lqonecc}}(\mathscr{C}) \triangleq \{l \rightsquigarrow x \mid l \rightsquigarrow x\#n \in \mathscr{C} \text{ and } \alpha_{\text{lqone}}(n)\}$$
$$\gamma_{\text{lqonecc}}(\mathscr{D}) \triangleq \bigcup_{l \in \mathcal{L}, x \in \text{Var}_{\text{P}}} \{l \rightsquigarrow x\#n \mid n = \gamma_{\text{lqone}}(l \rightsquigarrow x \in \mathscr{D})\}$$

We use Lemmas 6 and 7 to abstract further the cardinality abstract semantics and derive the correct by construction dependence analysis of Section 6. This derivation, which can be found in Assaf et al. (2016a), proves Lemma 4 and Theorem 2 stated earlier.

As a corollary and by Theorem 3, this also proves the precision of the cardinality analysis relative to Amtoft and Banerjee's logic (Amtoft and Banerjee 2004) as well as Hunt and Sands' type system (Hunt and Sands 2006, 2011).

Corollary 1 No leakage for well-typed programs. For all c, $\mathscr{C}_0, \mathscr{C} \in \text{Card}$, $\Delta_0, \Delta \in \text{Var}_{\text{P}} \to \mathcal{L}$, where $\bot_{\mathcal{L}} \vdash \Delta_0\{c\}\Delta$, and $\mathscr{C} = (\!|c|\!)^\sharp \mathscr{C}_0$, it holds that:

$$\alpha_{\text{hs}} \circ \alpha_{\text{lqonecc}}(\mathscr{C}_0) \dot{\sqsubseteq} \Delta_0 \implies$$
$$\Big(\forall x \in \text{Var}_{\text{P}}, l \in \mathcal{L}, \quad \Delta(x) \sqsubseteq l \implies \mathcal{O}_C^l(\!|x|\!)^\sharp \leq 1\Big)$$

The cardinality analysis determines that there is no leakage for programs that are "well-typed" by the flow-sensitive type system of Hunt and Sands. By "well-typed", we mean that the final typing environment that is computed by the type system allows attackers with security clearance $l \in \mathcal{L}$ to observe a variable $x \in \text{Var}_{\text{P}}$.

To the best of our knowledge, the cardinality abstraction is the first approximation-based analysis for quantitative information flow that provides a formal precision guarantee wrt. traditional analyses for qualitative information flow. This advantage makes the cardinality analysis appealing even when interested in proving a qualitative security policy such as non-interference, since the cardinality abstraction provides quantitative information that may assist in making better informed decisions if declassification is necessary. Nonetheless, we need further experimentation to compare to other quantitative analyses —see Section 9.

8. Towards More Precision

This section introduces examples to evaluate the precision of the analyses, and shows how existing analyses can be leveraged to improve precision. For simplicity, we consider a two point lattice $\{L, H\}$ and an initial typing context where variables y_i are the only low variables ($\Gamma(y_i) = L$). As is usual, low may flow to high ($L \sqsubseteq H$).

Consider the following program.

```
if (y₁ ≥ secret) then
   x := y₂
else
   x := y₃
```

Listing 1. Leaking 1 bit of `secret`

The cardinality abstraction determines that x has at most 2 values after the execution of the program in Listing 1, for initially L-equivalent traces. For fixed low inputs, x has one value in the then branch and one value in the else branch, and these cardinalities get summed after the conditional since the conditional guard may evaluate to 2 different values. Thus, the cardinality abstraction proves that this example program satisfies the hyperproperty $\text{SR}(L, 2, x)$.

Stronger trace properties. Another way of proving a hyperproperty is by proving a stronger trace property. If a program is proven to satisfy a trace property $T \in \mathcal{P}(\mathbf{Trc})$, then proving that T is stronger than hyperproperty $H \in \mathcal{P}(\mathcal{P}(\mathbf{Trc}))$—in the sense that $\gamma_{\text{hpp}}(T) \subseteq H$—guarantees the program satisfies the hyperproperty H. For instance, by proving for some program that an output variable x ranges over an interval of integer values whose size is k, we can prove that program satisfies $\text{SR}(L, k, x)$.

However, approximating a hyperproperty by a trace property may be too coarse for some programs, as we can illustrate with an interval analysis (Cousot and Cousot 1977) on the example program in Listing 1. Such an interval analysis loses too much precision in the initial state of this program, since it maps all low input variables y_1, y_2 and y_3 to $[-\infty, +\infty]$. After the conditional, it determines that x belongs to the interval $[-\infty, +\infty]$, which is a coarse over-approximation. Also, a polyhedron (Cousot and Halbwachs 1978) does not capture the disjunction that is needed for this example program ($x = y_2$ or $x = y_3$). Both abstract domains and many more existing ones are not suitable for the task of inferring cardinalities or dependences because they are convex. Using them as a basis to extract counting information delivers an over-approximation of the leakage, but a coarse one, especially in the presence of low inputs.

A disjunction of two polyhedra —through powerset domains, disjunctive postconditions, or partitioning (Bourdoncle 1992)— is as precise as the cardinality analysis for this example. However, disjunctions are not tractable in general. As soon as one fixes a maximum number of disjunctive elements (as in the quantitative information flow analysis of Mardziel et al. (2011, 2013)) or defines a widening operator to guarantee convergence, one loses the relative precision wrt. classical dependence analyses (Amtoft and Banerjee 2004; Hunt and Sands 2006) that the cardinality analysis guarantees (Cf. Corollary 1). Future work will investigate relying on cardinality analysis as a strategy guiding trace partitioning (Rival and Mauborgne 2007). Combining our analyses with existing domains will also deliver better precision.

Consider the following program.

```
if (y₁ ≥ secret) then x := y₂ else x := y₃;
o := x * y₄
```

Listing 2. Leaking x

The cardinal abstraction determines that variable o leaks the two possible values of x: for fixed low inputs, x has two possible values whereas y_4 has one possible value. Relational abstract domains such as polyhedra (Cousot and Halbwachs 1978) or octogons (Miné

2006a) do not support non-linear expressions, and therefore are unable to compute a precise bound of the leakage for variable o. Consider an analysis with a disjunction $\{x = y_2 \vee x = y_3\}$ of polyhedra and linearisation over intervals (Miné 2006b). Linearisation of expressions y_2*y_4 and y_3*y_4 will compute the following constraints for variable o: $\{(o = y_2 * [-\infty, +\infty]) \vee (o = y_3 * [-\infty, +\infty])\}$ if linearisation happens for the right side of expressions, or constraint $\{(o = [-\infty, +\infty] * y_4) \vee (o = [-\infty, +\infty] * y_4)\}$ if linearisation happens for the left side expressions. Two more combinations of constraints are possible, but none will deduce that variable o has at most 2 values, because the underlying domain of intervals lacks the required precision. Linearisation over both intervals and cardinalities delivers better precision.

Scaling to richer languages. We can rely on existing abstract domains to support richer language constructs, e.g. pointers and aliasing. Consider the following variation of Listing 1.

```
if (y₁ ≥ secret) then
   p := &y₂
else
   p := &y₃
o := *p
```

Listing 3. Leaking 1 bit of `secret`

The cardinality abstraction determines that initially L-equivalent memories lead to a variety of at most 2 in the pointer p after the conditional, whereas both y_2 and y_3 have a variety of 1. Assuming an aliasing analysis determines that p may point to y_2 or y_3, the cardinality analysis determines that variable o has a variety of at most 2, for initially L-equivalent memories.

Improving precision. To improve precision of the cardinality abstraction, we can augment it with existing abstract domains. One shortcoming of the cardinality analysis is the fact that it is not relational. Assuming attackers with security clearance L observe both variables x and o after execution of the program in Listing 2, the cardinality abstraction leads us to compute a leakage of two bits: four different possible values, instead of only 2 possible values for initially L-equivalent memories. Relying on a relational domain with linearisation (Miné 2006b) over cardinalities captures the required constraints $\{L \rightsquigarrow x\#2, L \rightsquigarrow o\#1 * x\}$ to compute a leakage of only one bit; these constraints are to be interpreted as "initially L-equivalent memories result in o being equal to one fixed integer times x, and x having at most 2 values".

We leave these extensions of cardinality analysis —and its abstraction as dependence analysis— for future work. In the following, we focus on one particular improvement to both previous analyses in order to gain more precision. We uncovered this case while deriving the analyses, by relying on the ***calculational framework of abstract interpretation***. Indeed, notice that the following holds:

$$\alpha_{\text{crdval}} \circ \mathcal{O}^l (\!|x_1|\!) \circ (\!|\text{grd}^{x_1==x_2}|\!) \circ \gamma_{\text{crdtr}}(\mathscr{C}) \le \mathcal{O}^l_C (\!|x_2|\!)^\sharp \mathscr{C}$$
$$\alpha_{\text{crdval}} \circ \mathcal{O}^l (\!|x_2|\!) \circ (\!|\text{grd}^{x_1==x_2}|\!) \circ \gamma_{\text{crdtr}}(\mathscr{C}) \le \mathcal{O}^l_C (\!|x_1|\!)^\sharp \mathscr{C}$$

Therefore, we can deduce that:

$$\begin{aligned}
&\alpha_{\text{crdtr}} \circ (\!|\text{grd}^{x_1==x_2}|\!) \circ \gamma_{\text{crdtr}}(\mathscr{C}) \\
&\quad \sqsubseteq^\sharp \{l \rightsquigarrow x\#n \in \mathscr{C} \mid x \neq x_1, x \neq x_2\} \\
&\qquad \cup \{l \rightsquigarrow x_1\#\min(n_1, n_2), l \rightsquigarrow x_2\#\min(n_1, n_2) \mid \\
&\qquad\qquad l \rightsquigarrow x_1\#n_1 \in \mathscr{C}, l \rightsquigarrow x_2\#n_2 \in \mathscr{C}\} \\
&\quad \triangleq (\!|\text{grd}^{x_1==x_2}|\!)^\sharp \mathscr{C}
\end{aligned}$$

For other comparison operators, we use as before $(\!|\text{grd}^b|\!)^\sharp \mathscr{C} \triangleq \mathscr{C}$.

We can now also improve the dependence abstraction:

$$\begin{aligned}
&\alpha_{\text{lqonecc}} \circ (\!|\text{grd}^{x_1==x_2}|\!)^\sharp \circ \gamma_{\text{lqonecc}}(\mathscr{D}) \\
&\quad \sqsubseteq^\natural \alpha_{\text{lqonecc}} (\{l \rightsquigarrow x\#n \in \gamma_{\text{lqonecc}}(\mathscr{D}) \mid x \neq x_1, x \neq x_2\}) \\
&\qquad \cup \alpha_{\text{lqonecc}}(\{l \rightsquigarrow x_1\#\min(n_1, n_2), l \rightsquigarrow x_2\#\min(n_1, n_2) \mid \\
&\qquad\qquad l \rightsquigarrow x_1\#n_1 \in \gamma_{\text{lqonecc}}(\mathscr{D}), l \rightsquigarrow x_2\#n_2 \in \gamma_{\text{lqonecc}}(\mathscr{D})\}) \\
&\quad \sqsubseteq^\natural \{l \rightsquigarrow x \in \mathscr{D} \mid x \neq x_1, x \neq x_2\} \\
&\qquad \cup \{l \rightsquigarrow x_1, l \rightsquigarrow x_2 \mid l \rightsquigarrow x_1 \in \mathscr{D} \text{ or } l \rightsquigarrow x_2 \in \mathscr{D}\} \\
&\quad \triangleq (\!|\text{grd}^{x_1==x_2}|\!)^\natural \mathscr{D}
\end{aligned}$$

For other comparison operators, we also use $(\!|\text{grd}^b|\!)^\natural \mathscr{D} \triangleq \mathscr{D}$.

With these new definitions, we can update the abstract semantics of conditionals and loops, for both dependences and cardinalities, to leverage the transfer functions $(\!|\text{grd}^-|\!)^\natural$ and $(\!|\text{grd}^-|\!)^\sharp$.

Improved dependences abstract semantics $\quad (\!|c|\!)^\natural \in \text{Dep} \to \text{Dep}$

$$\begin{aligned}
&(\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\natural \mathscr{D} \triangleq \\
&\qquad \text{let } \mathscr{D}_1 = (\!|\text{grd}^b|\!)^\natural \circ (\!|c_1|\!)^\natural \mathscr{D} \text{ in} \\
&\qquad \text{let } \mathscr{D}_2 = (\!|\text{grd}^{\neg b}|\!)^\natural \circ (\!|c_2|\!)^\natural \mathscr{D} \text{ in} \\
&\qquad \text{let } W = \text{Mod}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) \text{ in} \\
&\qquad \bigcup_{l \in \mathcal{L}} \begin{cases} \pi^l(\mathscr{D}_1) \sqcup^\natural \pi^l(\mathscr{D}_2) & \text{if } \mathcal{O}^l_D (\!|b|\!)^\natural \mathscr{D} \\ \{l \rightsquigarrow x \in \pi^l(\mathscr{D}) \mid x \notin W\} & \text{otherwise} \end{cases}
\end{aligned}$$

$$(\!|\textbf{while } b \textbf{ do } c|\!)^\natural \mathscr{D} \triangleq (\!|\text{grd}^{\neg b}|\!)^\natural \circ \text{lfp}^{\sqsubseteq^\sharp}_{\mathscr{D}} (\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\natural$$

Improved cardinality abs. semantics $\quad (\!|c|\!)^\sharp \in \text{Card} \to \text{Card}$

$$\begin{aligned}
&(\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\sharp \mathscr{C} \triangleq \\
&\qquad \text{let } \mathscr{C}_1 = (\!|\text{grd}^b|\!)^\sharp \circ (\!|c_1|\!)^\sharp \mathscr{C} \text{ in} \\
&\qquad \text{let } \mathscr{C}_2 = (\!|\text{grd}^{\neg b}|\!)^\sharp \circ (\!|c_2|\!)^\sharp \mathscr{C} \text{ in} \\
&\qquad \text{let } W = \text{Mod}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) \text{ in} \\
&\qquad \bigcup_{l \in \mathcal{L}} \begin{cases} \pi^l(\mathscr{C}_1) \sqcup^\sharp \pi^l(\mathscr{C}_2) & \text{if } \mathcal{O}^l_C (\!|b|\!)^\sharp \mathscr{C} = 1 \\ \pi^l(\mathscr{C}_1) \sqcup^\sharp_{\text{add}(W, \pi^l(\mathscr{C}))} \pi^l(\mathscr{C}_2) & \text{otherwise} \end{cases}
\end{aligned}$$

$$(\!|\textbf{while } b \textbf{ do } c|\!)^\sharp \mathscr{C} \triangleq (\!|\text{grd}^{\neg b}|\!)^\natural \circ \text{lfp}^{\sqsubseteq^\sharp}_{\mathscr{C}} (\!|\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2|\!)^\sharp$$

To illustrate the benefits of this improvement, consider the following example.

```
while (secret != y₃) do {
  x := x+1;
  secret := secret - 1;
}
o := secret;
```

Listing 4. Improved precision

The cardinality analysis determines that initially L-equivalent memories result in x having an infinity of values: the L-cardinality of x grows until it is widened to ∞. In contrast, cardinalities also determine that variables o and $secret$ have only 1 value, assuming L-equivalent memories. This is because of the reduction that concerns variable $secret$ after the while loop, specifically $(\!|\text{grd}^{secret==y_3}|\!)^\natural$. Similarly, the improved dependence analysis also determines that both variables $secret$ and o are low. These are sound precision gains for termination-insensitive noninterference; Askarov et al. (2008) discusses the guarantees provided by this security requirement.

Remarkably, this has been overlooked by many previous analyses. In fact, this simple improvement makes our dependence analysis strictly more precise than Amtoft and Banerjee (2004)'s and Hunt and Sands (2006, 2011)'s analyses and incomparable to the more recent dependence analysis of Müller et al. (2015).

Combination with intervals. Consider now the following example inspired from Müller et al. (2015).

```
if (secret == 0) then {
   x := 0;
   y := y + 1;
}
else {
   x := 0;
}
```

Listing 5. Example program from Müller et al. (2015)

The analysis of Müller et al. (2015) determines that x is low, whereas the cardinality abstraction determines that L-equivalent memories result in at most 2 values for variable x, because it does not track the actual values of variables. We can combine cardinality with an interval analysis to be more precise in such cases, through a reduced product (Cousot and Cousot 1979; Granger 1992; Cortesi et al. 2013).

Assume a set StInt of interval environments provided with the usual partial order that we denote by $\dot{\leq}^{\sharp,\mathrm{Int}}$. Assume also a Galois connection $(\alpha^{\mathrm{Int}}, \gamma^{\mathrm{Int}})$ enabling the derivation of an interval analysis as an approximation of a standard collecting semantics defined over $\mathcal{P}(\mathbf{Trc})$. We can lift this Galois connection to $\mathcal{P}(\mathcal{P}(\mathbf{Trc}))$ to obtain a Galois connection by compositing with $(\alpha_{\mathrm{hpp}}, \gamma_{\mathrm{hpp}})$, to obtain $(\alpha', \gamma') \triangleq (\alpha^{\mathrm{Int}} \circ \alpha_{\mathrm{hpp}}, \gamma^{\mathrm{Int}} \circ \gamma_{\mathrm{hpp}})$ with:

$$(\mathcal{P}(\mathcal{P}(\mathbf{Trc})), \subseteq) \underset{\alpha_{\mathrm{hpp}}}{\overset{\gamma_{\mathrm{hpp}}}{\leftrightarrows}} (\mathcal{P}(\mathbf{Trc}), \subseteq) \underset{\alpha^{\mathrm{Int}}}{\overset{\gamma^{\mathrm{Int}}}{\leftrightarrows}} (\mathrm{StInt}, \dot{\leq}^{\sharp,\mathrm{Int}})$$

A Granger's reduced product Granger (1992) for the cardinality abstraction and an interval analysis may be defined as a pair of functions $\mathrm{toint} \in \mathrm{Card} \times \mathrm{StInt} \to \mathrm{StInt}$ and $\mathrm{tocard} \in \mathrm{Card} \times \mathrm{StInt} \to \mathrm{Card}$ verifying the following conditions:

1. soundness:
$$\begin{aligned} \gamma'(\mathrm{toint}(\mathscr{C}, \imath)) \cap \gamma_{\mathrm{crdtr}}(\mathscr{C}) &= \gamma'(\imath) \cap \gamma_{\mathrm{crdtr}}(\mathscr{C}) \\ \gamma'(\imath) \cap \gamma_{\mathrm{crdtr}}(\mathrm{tocard}(\mathscr{C}, \imath)) &= \gamma'(\imath) \cap \gamma_{\mathrm{crdtr}}(\mathscr{C}) \end{aligned}$$
2. reduction:
$$\begin{aligned} \mathrm{toint}(\mathscr{C}, \imath) &\ \dot{\leq}^{\sharp,\mathrm{Int}}\ \imath \\ \mathrm{tocard}(\mathscr{C}, \imath) &\ \sqsubseteq^{\sharp}\ \mathscr{C} \end{aligned}$$

Let us denote by size the function that returns the size of an interval. One such Granger's reduced product can be defined as:

$$\begin{array}{lcl} \mathrm{tocard} & \in & \mathrm{Card} \times \mathrm{StInt} \to \mathrm{Card} \\ \mathrm{tocard}(\mathscr{C}, \imath) & \triangleq & \{l \rightsquigarrow x\#n' \mid \ l \rightsquigarrow x\#n \in \mathscr{C} \text{ and} \\ & & \qquad n' = \min(n, \mathrm{size}\ \imath(x))\} \\ \mathrm{toint} & \in & \mathrm{Card} \times \mathrm{StInt} \to \mathrm{Card} \\ \mathrm{toint}(\mathscr{C}, \imath) & \triangleq & \imath \end{array}$$

Once enhanced with this reduced product, the cardinality analysis determines for the program in Listing 5, that L-equivalent memories result in at most one possible value for variable x.

The dependence analysis can be improved similarly, with a reduction function defined as follows:

$$\mathrm{todep} \in \mathrm{Dep} \times \mathrm{StInt} \to \mathrm{Dep}$$
$$\mathrm{todep}(\mathscr{D}, \imath) \triangleq \mathscr{D} \cup \{l \rightsquigarrow x \mid l \in \mathcal{L} \text{ and } \mathrm{size}\ \imath(x) = 1\}$$

Once extended with a reduced product with intervals, the dependence analysis is also able to determine that variable x is low for the program in Listing 5.

```
//L ⇝ h#∞, L ⇝ y1#1, L ⇝ y2#1, L ⇝ y3#1
y1 := 1;//L ⇝ y1#1
if (h == y1) then {
   skip; //L ⇝ h#1, L ⇝ y1#1, L ⇝ y2#1
}
else {
   y2 := 5;   //L ⇝ y1#1, L ⇝ y2#1
   while (y2 != 1) do {
      y2 := y2-1;//L ⇝ y2#1
      y1 := y2;//L ⇝ y1#1
   }//L ⇝ y1#1, L ⇝ y2#1
}
//L ⇝ h#∞, L ⇝ y1#2, L ⇝ y2#2, L ⇝ y3#1
o := y1 * y3;//L ⇝ o#2
```

Listing 6. No leakage for variable o

More reduced products. As a final example, let us consider Listing 6, inspired by Besson et al. (2016, program 7), that we annotate with the result of the improved cardinality abstraction. To the best of our knowledge, no existing automated static analysis determines that variable o is low at the end of this program. Also, no prior monitor but the one recently presented by Besson et al. (2016) accepts all executions of this program, assuming attackers with clearance L can observe variable o.

For initially L-equivalent memories, the cardinality abstraction determines that variables y_1, y_2 and o have at most two values. This result is precise for y_2, but not precise for y_1 and o. As a challenge, let us see what is required to gain more precision to determine that both variables y_1 and o have at most 1 possible value – they are low.

To tackle this challenge, we need to consider cardinality combined with an interval analysis and a simple relational domain tracking equalities. With the equality $y_1 = y_2$ at the exit of the loop, both y_1 and y_2 will be reduced to the singleton interval $[1, 1]$. After the conditional, we still deduce that y_2 has at most 2 different values thanks to the cardinality abstraction. Using intervals, we deduce that variable y_1 has only one value (singleton interval $[1, 1]$). And finally, at the last assignment the cardinalities abstraction determines that variable o has only one possible value. Similarly, this same combination of analyses can be put to use to let the dependence analysis reach the desired precision.

9. Related Work

Although noninterference has important applications, for many security requirements it is too strong. That is one motivation for research in quantitative information flow analysis. In addition, a number of works investigate weakenings of noninterference and downgrading policies that are conditioned on events or data values (Askarov and Sabelfeld 2007; Banerjee et al. 2008; Sabelfeld and Sands 2009; Mastroeni and Banerjee 2011). Assaf (2015, Chapter 4) proposes to take the guarantees provided by termination-insensitive noninterference (Askarov et al. 2008) as an explicit definition for security; this *Relative Secrecy* requirement is inspired by Volpano and Smith (2000) who propose a type-system preventing batch-job programs from leaking secrets in polynomial time. Giacobazzi and Mastroeni (2004) introduce *abstract noninterference*, which generalizes noninterference by means of abstract interpretations that specify, for example, limits on the attacker's power and the extent of partial releases (declassification). The survey by Mastroeni (2013) further generalizes the notion and highlights, among other things, its applicability to a range of underlying semantics. The Galois connections in this work are at the level of trace sets, not sets of sets. Abstract noninterference retains the explicit 2-run formulation (Volpano et al.

1996; Sabelfeld and Myers 2003): from two related initial states, two executions lead to related final states. The relations are defined in terms of abstract interpretations of the individual states/executions. Mastroeni and Banerjee (2011) show how to infer indistinguishability relations—modelling attackers' observations—to find the best abstract noninterference policy that holds. The inference algorithm iteratively refines the relation by using counter-examples and abstract domain completion (Cousot and Cousot 1979).

Set-of-sets structures occur in work on abstraction for nondeterministic programs, but in those works one level of sets are powerdomains for nondeterminacy; the properties considered are trace properties (Schmidt 2009, 2012). Hunt and Sands (1991) develop a binding time analysis and a strictness analysis (Hunt 1990) based on partial equivalence relations: Their concretisations are sets of equivalence classes. Cousot and Cousot point out that this analysis could be achieved by a collecting semantics over sets-of-sets, defined simply as a direct image. To the best of our knowledge this has not been explored further in the literature, except in unpublished work on which this paper builds (Assaf 2015; Assaf et al. 2016b).

Clarkson et al. (2014); Finkbeiner et al. (2015) extend temporal logic with means to quantify over multiple traces in order to express hyperproperties, and provide model checking algorithms for finite space systems. Agrawal and Bonakdarpour (2016) introduce a technique for runtime verification of k-safety properties.

The dependences analysis we derive is similar to the information flow logic of Amtoft and Banerjee (2004) and the equivalent flow-sensitive type system of Hunt and Sands (2006). Amtoft and Banerjee use the domain $\mathcal{P}(\mathbf{Trc})$ and on the basis of a relational logic they validate a forward analysis. In effect their interpretation of "independences" is a Galois connection with sets of sets, but the analysis is not formulated or proved correct as an abstract interpretation. To deal with dynamically allocated state, Amtoft et al. (2006) augment the relational assertions of information flow logic with region assertions, which can be computed by abstract interpretation. This is used both to express agreement relations between the two executions and to approximate modifiable locations. This approach is generalized in Banerjee et al. (2016) to a relational Hoare logic for object-based programs that encompasses information flow properties with conditional downgrading (Banerjee et al. 2008).

Müller et al. (2015) give a backwards analysis that infers dependencies and is proved strictly more precise than (Hunt and Sands 2006; Amtoft and Banerjee 2004). This is achieved by product construction that facilitates inferring relations between variables in executions that follow different control paths. Correctness of the analysis is proved by way of a relational Hoare logic. The variations of our proposed analyses, in Section 8, rivals theirs in terms of precision—they are incomparable.

Our dependence analysis relies on an approximation of the modifiable variables, to soundly track implicit flows due to control flow, instead of labelling a program counter variable pc to account for implicit flows (Sabelfeld and Myers 2003). Zanioli and Cortesi (2011) also derive a similar analysis through a syntactic Galois connection—a syntactic assignment $z := x * y$ is abstracted into a propositional formula $x \to z \wedge y \to z$ denoting an information flow from variables x and y to variable z. The soundness of this analysis wrt. a semantic property such as noninterference requires more justification, though it is remarkable that the concretisation of propositional formula yields, roughly speaking, a set of program texts. Zanotti (2002) also provides an abstract interpretation account of a flow-insensitive type system (Volpano et al. 1996) enforcing noninterference by guaranteeing a stronger safety property, namely that sensitive locations should not influence public locations (Boudol 2008).

Kovács et al. (2013) explicitly formulate termination-insensitive noninterference as an abstract interpretation, namely the "merge over all twin computations" that makes explicit both the 2-safety aspect and the need for an analysis to relate some aligned intermediate states. Their analysis, like many others, is based on reducing the problem to a safety property of product programs. Sousa and Dillig (2016) implement an algorithm that automates reasoning in a Hoare logic for k-safety, implicitly constructing product programs; the performance compares favorably with explicit construction of product programs. Program dependency graphs are another approach to dependency, shown to be correct for noninterference by Wasserrab et al. (2009) using slicing and a simulation argument.

Denning (1982, Chap. 5) proposes the first quantitative measure of a program's leakage in terms of Shannon entropy (Shannon 1948). Other quantitative metrics emerge in the literature (Braun et al. 2009; Clarkson et al. 2009; Smith 2009; Dwork 2011; Smith 2011; Alvim et al. 2012). These quantitative security metrics model different scenarios suitable for different policies. Most existing static analyses for quantitative information flow leverage existing model checking tools and abstract domains for safety; they prove that a program satisfies a quantitative security requirement by proving a stronger safety property. In contrast, the cardinal abstraction proves a hyperproperty by inferring a stronger hyperproperty satisfied by the analysed program. This is key to target quantitative information flow in mutlilevel security lattices, beyond the 2-point lattice $\{L, H\}$.

Backes et al. (2009) synthesize equivalence classes induced by outputs over low equivalent memories by relying on software model checkers, in order to bound various quantitative metrics. Heusser and Malacaria (2009) also rely on a similar technique to quantify information flow for database queries. Köpf and Rybalchenko (2010) note that the exact computation of information-theoretic characteristics is prohibitively hard, and propose to rely on approximation-based analyses, among which are randomisation techniques and abstract interpretation ones. They also propose to rely on a self-composed product program to model a scenario where attackers may refine their knowledge by influencing the low inputs. Klebanov (2014) relies on similar techniques to handle programs with low inputs, and uses polyhedra to synthesize linear constraints (Cousot and Halbwachs 1978) over variables. Mardziel et al. (2013) decide whether answering a query on sensitive data augments attackers' knowledge beyond a certain threshold, by using probabilistic polyhedra.

10. Conclusion

Galois connection-based semantic characterisations of program analyses provide new perspectives and insights that lead to improved techniques. We have extended the framework to fully encompass hyperproperties, through a remarkable form of hypercollecting semantics that enables calculational derivation of analyses. This new foundation raises questions too numerous to list here.

One promising direction is to combine dependence and cardinality analysis with existing abstract domains, e.g. through advanced symbolic methods (Miné 2006b), and partitioning (Handjieva and Tzolovski 1998; Rival and Mauborgne 2007).

Static analysis of secure information flow has yet to catch up with recent advances in dynamic information flow monitoring (Besson et al. 2013; Bello et al. 2015; Hedin et al. 2015; Assaf and Naumann 2016; Besson et al. 2016). We discussed, in Section 8, how existing static analyses may be of use to statically secure information flow. It seems likely that hypercollecting semantics will also be of use for dynamic analyses.

Acknowledgments

Thanks to Anindya Banerjee and the anonymous reviewers for thoughtful comments and helpful feedback. This work was partially supported by NSF awards CNS-1228930 and CCF-1649884, ANR project AnaStaSec ANR-14-CE28-0014 and a CFR CEA Phd Fellowship.

References

S. Agrawal and B. Bonakdarpour. Runtime verification of k-safety hyperproperties in HyperLTL. In *IEEE Computer Security Foundations Symposium*, pages 239–252, 2016.

M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *IEEE Computer Security Foundations Symposium*, pages 265–279, 2012.

T. Amtoft and A. Banerjee. Information flow analysis in logical form. In *Static Analysis Symposium*, pages 100–115, 2004.

T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In *ACM Symposium on Principles of Programming Languages*, pages 91–102, 2006.

A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *IEEE Symposium on Security and Privacy*, 2007.

A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands. Termination-insensitive noninterference leaks more than just a bit. In *European Symposium on Research in Computer Security*, volume 5283 of *LNCS*, 2008.

M. Assaf. *From Qualitative to Quantitative Program Analysis : Permissive Enforcement of Secure Information Flow*. PhD thesis, Université de Rennes 1, May 2015. `https://hal.inria.fr/tel-01184857`.

M. Assaf and D. Naumann. Calculational design of information flow monitors. In *IEEE Computer Security Foundations Symposium*, pages 210–224, 2016.

M. Assaf, D. Naumann, J. Signoles, É. Totel, and F. Tronel. Hypercollecting semantics and its application to static analysis of information flow. Technical report, Apr. 2016a. URL `https://arxiv.org/abs/1608.01654`.

M. Assaf, J. Signoles, É. Totel, and F. Tronel. The cardinal abstraction for quantitative information flow. In *Workshop on Foundations of Computer Security (FCS)*, June 2016b. `https://hal.inria.fr/hal-01334604`.

M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, pages 141–153. IEEE, 2009.

A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *IEEE Symposium on Security and Privacy*, pages 339–353, 2008.

A. Banerjee, D. A. Naumann, and M. Nikouei. Relational logic with framing and hypotheses. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2016. To appear.

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

L. Bello, D. Hedin, and A. Sabelfeld. Value sensitivity and observable abstract values for information flow control. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 63–78, 2015.

N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *ACM Symposium on Principles of Programming Languages*, pages 14–25, 2004.

J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*, 2012.

J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. *Foundations and Trends in Programming Languages*, 2 (2-3):71–190, 2015.

F. Besson, N. Bielova, and T. Jensen. Hybrid information flow monitoring against web tracking. In *IEEE Computer Security Foundations Symposium*, pages 240–254. IEEE, 2013.

F. Besson, N. Bielova, and T. Jensen. Hybrid monitoring of attacker knowledge. In *IEEE Computer Security Foundations Symposium*, pages 225–238, 2016.

G. Boudol. Secure information flow as a safety property. In *Formal Aspects in Security and Trust*, pages 20–34, 2008.

F. Bourdoncle. *Journal of Functional Programming*, 2(04):407–435, 1992.

C. Braun, K. Chatzikokolakis, and C. Palamidessi. Quantitative notions of leakage for one-try attacks. In *Mathematical Foundations of Programming Semantics (MFPS)*, volume 249, pages 75–91, 2009.

D. Cachera and D. Pichardie. A certified denotational abstract interpreter. In *Interactive Theorem Proving (ITP)*, pages 9–24. 2010.

M. R. Clarkson and F. B. Schneider. Hyperproperties. In *IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17:655–701, 2009.

M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *Principles of Security and Trust*, volume 8414 of *LNCS*, pages 265–284, 2014.

E. Cohen. Information transmission in computational systems. In *Proceedings of the sixth ACM Symposium on Operating Systems Principles*, pages 133–139, 1977.

A. Cortesi and M. Zanioli. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures*, pages 24–42, 2011.

A. Cortesi, G. Costantini, and P. Ferrara. A survey on product operators in abstract interpretation. In *Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday*, volume 129 of *EPTCS*, pages 325–336, 2013.

P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*, volume 173, pages 421–506. NATO ASI Series F. IOS Press, Amsterdam, 1999.

P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277 (1-2):47–103, 2002.

P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages).

P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

P. Cousot and R. Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In *Programming Language Implementation and Logic Programming (PLILP)*, pages 269–295, 1992.

P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM Symposium on Principles of Programming Languages*, pages 84–96, 1978.

Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *Security in Pervasive Computing*, pages 193–209. 2005.

D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

D. E. R. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of ACM*, 20(7):504–513, 1977.

G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. Cacheaudit: A tool for the static analysis of cache side channels. In *USENIX Security Symposium*, pages 431–446, 2013.

C. Dwork. A firm foundation for private data analysis. *Communications of ACM*, pages 86–95, 2011.

B. Finkbeiner, M. N. Rabe, and C. Sánchez. Algorithms for model checking HyperLTL and HyperCTL ^*. In *Computer Aided Verification*, volume 9206 of *LNCS*, pages 30–48, 2015.

R. Giacobazzi and I. Mastroeni. Abstract non-interference: parameterizing non-interference by abstract interpretation. In *ACM Symposium on Principles of Programming Languages*, pages 186–197, 2004.

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

P. Granger. Improving the results of static analyses programs by local decreasing iteration. In *Foundations of Software Technology and Theoretical Computer Science*, volume 652, pages 68–79, 1992.

M. Handjieva and S. Tzolovski. Refining dtatic analyses by trace-based partitioning using control flow. In *International Static Analysis Symposium*, 1998.

D. Hedin, L. Bello, and A. Sabelfeld. Value-sensitive hybrid information flow control for a JavaScript-Like language. In *IEEE Computer Security Foundations Symposium*, pages 351–365, 2015.

J. Heusser and P. Malacaria. Applied quantitative information flow and statistical databases. In *Formal Aspects in Security and Trust*, pages 96–110, 2009.

S. Hunt. PERs generalize projections for strictness analysis (extended abstract). In *Proceedings of the Third Annual Glasgow Workshop on Functional Programming*, 1990.

S. Hunt and D. Sands. Binding time analysis: A new PERspective. In *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'91, Yale University, New Haven, Connecticut, USA, June 17-19, 1991*, pages 154–165, 1991.

S. Hunt and D. Sands. On flow-sensitive security types. In *ACM Symposium on Principles of Programming Languages*, pages 79–90, 2006.

S. Hunt and D. Sands. From exponential to polynomial-time security typing via principal types. In *ACM Workshop on Programming Languages and Analysis for Security*, pages 297–316, 2011.

V. Klebanov. Precise quantitative information flow analysis - a symbolic approach. *Theoretical Computer Science*, 538:124–139, 2014.

B. Köpf and A. Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *IEEE Computer Security Foundations Symposium*, pages 3–14, 2010.

B. Köpf and A. Rybalchenko. Automation of quantitative information-flow analysis. In *Formal Methods for Dynamical Systems - 13th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, volume 7938 of *LNCS*, pages 1–28, 2013.

M. Kovács, H. Seidl, and B. Finkbeiner. Relational abstract interpretation for the verification of 2-hypersafety properties. In *ACM SIGSAC conference on Computer and Communications Security*, pages 211–222, 2013.

P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies. In *IEEE Computer Security Foundations Symposium*, pages 114–128. IEEE, 2011.

P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation. *Journal of Computer Security*, 21(4):463–532, 2013.

I. Mastroeni. Abstract interpretation-based approaches to security - A survey on abstract non-interference and its challenging applications. In *Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday*, volume 129 of *EPTCS*, pages 41–65, 2013.

I. Mastroeni and A. Banerjee. Modelling declassification policies using abstract domain completeness. *Mathematical Structures in Computer Science*, 21(06):1253–1299, 2011.

J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symposium on Security and Privacy*, pages 79–93, 1994.

A. Miné. The octagon abstract domain. *Higher-order and symbolic computation*, 19(1):31–100, 2006a.

A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *Verification, Model Checking, and Abstract Interpretation*, pages 348–363. 2006b.

C. Müller, M. Kovács, and H. Seidl. An analysis of universal information flow based on self-composition. In *IEEE Computer Security Foundations Symposium*, pages 380–393, 2015.

F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.

A. Rényi. On measures of entropy and information. In *the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, 1961.

X. Rival and L. Mauborgne. The trace partitioning abstract domain. *ACM Transactions on Programming Languages and Systems*, 29(5):26, 2007.

J. Rushby. Security requirements specifications: How and what. In *Symposium on Requirements Engineering for Information Security (SREIS)*, 2001.

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

A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5), 2009.

D. A. Schmidt. Abstract interpretation from a topological perspective. In *Static Analysis, 16th International Symposium*, volume 5673 of *LNCS*, pages 293–308, 2009.

D. A. Schmidt. Inverse-limit and topological aspects of abstract interpretation. *Theoretical Computer Science*, 430:23–42, 2012.

D. Schoepe, M. Balliu, B. C. Pierce, and A. Sabelfeld. Explicit secrecy: A policy for taint tracking. In *IEEE European Symposium on Security and Privacy*, pages 15–30, 2016.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

G. Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302, 2009.

G. Smith. Quantifying information flow using min-entropy. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 159–167. IEEE, 2011.

M. Sousa and I. Dillig. Cartesian Hoare logic for verifying k-safety properties. In *ACM Conference on Programming Language Design and Implementation*, pages 57–69, 2016.

T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *Static Analysis Symposium*, pages 352–367. 2005.

D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *IEEE Computer Security Foundations Workshop*, pages 156–168, 1997.

D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *ACM Symposium on Principles of Programming Languages*, pages 268–276, 2000.

D. Volpano, C. Irvine, and G. Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2-3):167–187, 1996.

D. M. Volpano. Safety versus secrecy. In *Static Analysis Symposium*, pages 303–311, 1999.

D. Wasserrab, D. Lohner, and G. Snelting. On PDG-based noninterference and its modular proof. In *ACM Workshop on Programming Languages and Analysis for Security*, pages 31–44, 2009.

G. Winskel. *The Formal Semantics of Programming Languages: an Introduction*. Cambridge, 1993.

H. Yasuoka and T. Terauchi. On bounding problems of quantitative information flow. *Journal of Computer Security*, 19(6):1029–1082, 2011.

A. Zakinthinos and S. Lerner. A general theory of security properties. In *IEEE Symposium on Security and Privacy*, pages 94–102, 1997.

M. Zanioli and A. Cortesi. Information leakage analysis by abstract interpretation. In *SOFSEM 2011: Theory and Practice of Computer Science*, pages 545–557. 2011.

M. Zanotti. Security typings by abstract interpretation. In *Static Analysis Symposium*, volume 2477, pages 360–375, 2002.