A Per Model of Secure Information Flow in Sequential Programs *

Andrei Sabelfeld and David Sands ({andrei,dave}@cs.chalmers.se)
Department of Computer Science, Chalmers University of Technology and the University of Göteborg, 412 96 Göteborg, Sweden.

Abstract. This paper proposes an extensional semantics-based formal specification of secure information-flow properties in sequential programs based on representing degrees of security by partial equivalence relations (pers). The specification clarifies and unifies a number of specific correctness arguments in the literature and connections to other forms of program analysis. The approach is inspired by (and in the deterministic case equivalent to) the use of partial equivalence relations in specifying binding-time analysis, and is thus able to specify security properties of higher-order functions and "partially confidential data". We also show how the per approach can handle nondeterminism for a first-order language, by using powerdomain semantics and show how *probabilistic security properties* can be formalised by using probabilistic powerdomain semantics. We illustrate the usefulness of the compositional nature of the security specifications by presenting a straightforward correctness proof for a simple type-based security analysis.

Keywords: semantics, security, confidentiality, partial equivalence relations, non-interference, powerdomains, probabilistic covert channels.

1. Introduction

1.1. Motivation

You have received a program from an untrusted source. Let us call it company M. M promises to help you to optimise your personal financial investments, information about which you have stored in a database on your home computer. The software is free (for a limited time), under the condition that you permit a log-file containing a summary of your usage of the program to be automatically e-mailed back to the developers of the program (who claim they wish to determine the most commonly used features of their tool). Is such a program safe to use? The program must be allowed access to your personal investment information, and is allowed to send information, via the log-file, back to M. But how can you be sure that M is not obtaining your sensitive private financial information by cunningly encoding it in the contents of the innocent-looking log-file? This is an example of the problem of

* This is an extended version of the paper from the *Proceedings of European Symposium on Programming'99* [41].

determining that the program has *secure information flow*. Information about your sensitive "high-security" data should not be able to propagate to the "low-security" output (the log-file). Traditional methods of access control are of limited use here since the program has legitimate access to the database.

This paper proposes an extensional semantics-based formal specification of secure information-flow properties in sequential programs based on representing degrees of security by partial equivalence relations (pers[1]). The specification clarifies and unifies a number of specific correctness arguments in the literature, and connections to other forms of program analysis. The approach is inspired by and, in the deterministic case, equivalent to the use of partial equivalence relations in specifying binding-time analysis [19], and is thus able to specify security properties of higher-order functions and "partially confidential data" (e.g. one's financial database could be deemed to be partially confidential if the number of entries is not deemed to be confidential even though the entries themselves are). We show how the approach can also be adapted to handle nondeterminism in a first-order language, and illustrate how the various choices of powerdomain semantics affect the kinds of security properties that can be expressed, ranging from termination-insensitive properties (corresponding to the use of the Hoare (partial correctness) powerdomain) to *probabilistic security properties*, obtained when one uses a probabilistic powerdomain.

1.2. Background

The study of information flow in the context of systems with multiple levels of confidentiality was pioneered by Denning [9, 10] in an extension of Bell and LaPadula's early work [5]. Denning's approach is to apply a static analysis suitable for inclusion into a compiler. The basic idea is that security levels are represented as a lattice (for example the two point lattice $PublicDomain \leq TopSecret$). The aim of the static analysis is to ensure that information from inputs, variables or processes of a given security level only flows to outputs, variables or processes which have been assigned a higher or equal security level.

1.3. Semantic Foundations of Information Flow Analysis

In order to verify a program analysis or a specific proof of a program's security one must have a formal specification of what constitutes secure information flow. The value of a semantics-based specification for secure

[1] A partial equivalence relation is symmetric and transitive but not necessarily reflexive.

information flow is that it contributes significantly to the reliability of and the confidence in such activities, and can be used in the systematic design of program analyses. Many approaches to Denning-style analyses (including the original articles) contain a fair degree of formalism but arguably are lacking a rigorous soundness proof. Volpano et al. [48] claim to give the first satisfactory treatment of soundness of Denning's analysis. Such a claim rests on the dissatisfaction with soundness arguments based on an instrumented operational semantics [35] or denotational semantics [32], or on "axiomatic" approaches which define security in terms of a program logic [3] without any models to relate the logic to the semantics of the programming language. The problem here is that an "instrumented semantics" or a "security logic" is just a definition, not subject to any further mathematical justification. McLean points out [30] in a related discussion about the (non language-specific) Bell and LaPadula model:

> One problem is that ... they *[the Bell LaPadula security properties]* constitute a possible implementation of security, ... , rather than an abstract specification of what all secure systems must satisfy. By concerning themselves with particular controls over files inside the computer, rather than limiting themselves to the relation between input and output, they make it harder to reason about the requirements, ...

This criticism points to more abstract, *extensional* notions of soundness, based on, for example, the idea of *noninterference* introduced by Goguen and Meseguer [12]. In the setting of information flow properties of programs—rather than other abstract models of computation—the earlier work of Cohen [6, 7] is the true predecessor to the present article, since it also aims to provide semantics-based characterisations of Denning-style information-flow properties. Cohen's formalisation of information flow properties of sequential deterministic programs is discussed in Section 2.4.

1.4. Semantics-based Models of Information Flow

The problem of secure information flow, or "noninterference" is now quite mature, and very many specifications exist in the literature—see McLean's tutorial [31] for an overview. Many approaches – including those of Goguen and Meseguer and McLean have been phrased in terms of abstract trace-based models of computation. These models are typically based on nonstandard notions of automata. One criticism of these abstractions is that they are not always well-motivated from the point of view of the kinds of systems that they are intended to abstract. For example, Goguen and Meseguer's machines described only deterministic systems; McCullough's nondeterministic version focused on defining security via communication [28, 13], but computation is modelled as a globally clocked synchronous system, which in retrospect is not well suited to modelling properties of distributed systems. Even when "standard" models are used e.g., Moskowitz and Costich's use of classical automata [33], they have still been typically based on traces, and lack information about deadlock and branching behaviours (the way that nondeterminism is resolved). Only more recently have attempts been made to rephrase and compare various security conditions in terms of well-known semantic models specifically designed for modelling concurrent systems e.g. the use of labelled transition systems and bisimulation semantics by Focardi and Gorrieri [11].

In this paper we consider the problem of information-flow properties of sequential systems. The systems are those concretely realised by programs, and we use the framework of *denotational semantics* as our formal model of computation. Our approach is based on standard semantic models rather than ad hoc nonstandard extensions specific to the problem of information flow – c.f., the addition of special "security variables" as found in the work of Banâtre et al. [4], or the "trust" tags in the nonstandard dependence models used by Ørbæk [36].

Along the way we consider some relations to specific static analyses, such as the ***S**ecure **Lam**bda Calculus* [15] and an alternative semantic condition for secure information flow proposed by Leino and Joshi [25].

1.5. Overview

The rest of the paper is organised as follows.

Section 2 shows how the per-based condition for soundness of binding-time analysis is also a model of secure information flow. We show how this provides insight into the treatment of higher-order functions and structured data.

Section 3 shows how the approach can be adapted to the setting of a nondeterministic imperative language by appropriate use of a powerdomain-based semantics. We show how the choice of powerdomain (upper, lower or convex) affects the nature of the security condition.

Section 4 focuses on an alternative semantic specification due to Leino and Joshi. Modulo some technicalities we show that Leino's condition—and a family of similar conditions—are in agreement with, and can be represented using our form of specification.

Section 5 considers the problem of preventing unwanted *probabilistic* information flows in programs. We show how this can be solved in the same framework by utilising a probabilistic semantics based on the probabilistic powerdomain [21].

Section 6 shows how the probabilistic security specification satisfies compositionality properties which facilitates straightforward proofs of correctness for compositional analyses. We illustrate the usefulness of the compositional nature of the security specifications by presenting a straightforward correctness proof for a type-based security analysis.

Section 7 concludes.

2. A Per Model of Information Flow

In this section we introduce the way that *partial equivalence relations* (pers) can be used to model dependencies in programs. The basic idea comes from Hunt's use of pers to model and construct abstract interpretations for strictness properties in higher-order functional programs [18, 17], and in particular its use to model dependencies in *binding-time* analysis [19]. Related ideas already occur in the denotational formulation of live-variable analysis [34].

2.1. Binding-time Analysis as Dependency Analysis

Given a description of the parameters in a program that will be known at partial evaluation time (called the *static* arguments), a binding-time analysis (BTA) must determine which parts of the program are dependent solely on these known parts (and therefore also known at partial evaluation time). The safety condition for binding-time analysis must ensure that there is no dependency between the *dynamic* (i.e., non-static) arguments and the parts of the program that are deemed to be static. Viewed in this way, binding-time analysis is purely an analysis of dependencies.[2]

Dependencies in Security In the security field, the property of absence of unwanted dependencies is often called *noninterference*, after Goguen and Meseguer [12]. Many problems in security come down to forms of dependency analysis. For example, in the case of *confidentiality*, the

[2] Unfortunately, from the perspective of a partial evaluator, BTA is not *purely* a matter of dependencies; it was shown [16] that the pure dependency models [24, 19] are not quite adequate to ensure the safety of partial evaluation.

aim is to show that the outputs of a program which are deemed to be of low confidentiality do not have any dependence on inputs of a higher degree of confidentiality. In the case of *integrity* (*trust*), one must ensure that the value of some trusted data does not depend on some untrusted source.

But what is the precise meaning of the phrase "does not depend"? This is formalised by asking whether *variation* in high input can ever cause variation in the low output [7]. If no variation can be conveyed, then there is no information flow from high inputs to low outputs. Put more simply, if we change high security inputs, the low security output is not affected. In the remainder of this section we show how these ideas can be conveniently expressed using *partial equivalence relations* to model the concept of variation, and we compare with Cohen's early formalisation of the problem [7].

Some intuitions about information flow Let us consider a program modelled as a function from some input domain to an output domain. Now consider the following simple functions mapping inputs to outputs: $\mathsf{snd} : D \times E \to E$ for some sets (or domains) D and E, and shift and test, functions in $\mathbf{N} \times \mathbf{N} \to \mathbf{N} \times \mathbf{N}$ and $\mathbf{N} \times \mathbf{N} \to \mathbf{N}$, defined by:

$$\begin{aligned} \mathsf{snd}(x, y) &= y \\ \mathsf{shift}(x, y) &= (x + y, y) \\ \mathsf{test}(x, y) &= \mathsf{if}\ x > 0\ \mathsf{then}\ y\ \mathsf{else}\ y + 1 \end{aligned}$$

Now suppose that (h, l) is a pair where h is some high security information, and l is low, "public domain", information. Without knowing about what the actual values h and l might be, we know about the result of applying function snd to (h, l) will be a low value, and, in the case that we have a pair of numbers, the result of applying shift will be a pair with a high first component and a low second component.

Note that the function test does not enjoy the same security property that snd does, since although it produces a value which is constructed from purely low-security components, the actual value is dependent on the first component of the input. This is what is known as an indirect information flow [9].

It is rather natural to think of these properties as "security types":

$$\begin{aligned} \mathsf{snd} &: \mathit{high} \times \mathit{low} \to \mathit{low} \\ \mathsf{shift} &: \mathit{high} \times \mathit{low} \to \mathit{high} \times \mathit{low} \\ \mathsf{test} &: \mathit{high} \times \mathit{low} \to \mathit{high} \end{aligned}$$

But what notion of "type", and what interpretation of "high" and "low" can formalise these more intuitive type statements? Interpreting types as sets of values is not adequate to model "high" and "low". To track degrees of dependence between inputs and outputs we need a more dynamic view of a type as a degree of variation. We must vary (parts of) the input and observe which (parts of) the output vary. For the application to confidentiality we want to determine if there is possible information leakage from a high level input to the parts of an output which are intended to be visible to a low security observer. We can detect this by observing whether the "low" parts of the output vary in any way as we vary the high input.

The simple properties of the functions snd and shift described above can be captured formally by the following formulae:

$$\forall x, x', y.\, \mathsf{snd}(x, y) = \mathsf{snd}(x', y) \tag{1}$$

$$\forall x, x', y.\, \mathsf{snd}(\mathsf{shift}(x, y)) = \mathsf{snd}(\mathsf{shift}(x', y)) \tag{2}$$

Indeed, this kind of formula forms the core of the correctness arguments for the security analyses proposed by e.g., Volpano and Smith et al. [48, 43], and also for the extensional correctness proofs in the core of the *SLam calculus* [15, 1].

High and Low as Equivalence Relations We show how we can interpret "security types" in general as partial equivalence relations. We will interpret *high* (for values in D) as the equivalence relation All_D, and *low* as the relation Id_D where for all $x, x' \in D$:

$$x \; All_D \; x'$$
$$x \; Id_D \; x' \iff x = x'$$

For a function $f : D \to E$ and binary relations $P \in Rel(D)$ and $Q \in Rel(E)$, we write $f : P \twoheadrightarrow Q$ iff

$$\forall x, x' \in D.\, x \; P \; x' \Longrightarrow (f\ x) \; Q \; (f\ x')$$

For binary relations P, Q we define the relation $P \times Q$ by:

$$(x, y) \; P \times Q \; (x', y') \iff x \; P \; x' \;\; \& \;\; y \; Q \; y'$$

Now the security property of snd described by (1) can be captured by:

$$\mathsf{snd} : All_D \times Id_E \twoheadrightarrow Id_E$$

and (2) is given by:

$$\mathsf{shift} : All_{\mathbf{N}} \times Id_{\mathbf{N}} \twoheadrightarrow All_{\mathbf{N}} \times Id_{\mathbf{N}}$$

An intuition for the relations *All* and *Id* is that they represent the perspective of the "low" user (the user who does not have access to the high information). The low user can see the difference between distinct low data elements, but one high datum is (or rather *should be*) indistinguishable from any other. This perspective is important when considering more complex lattices of security properties than just the simple two point lattice of low and high security levels. In the multi-level case one would model the perspective of a user at some security level k by using the relation *Id* for all data of security class less than or equal to k, and by *All* for all other classes—representing the indistiguishability of data items which are not classified as being visible at level k.

2.2. From Equivalence Relations to Pers

We have seen how the equivalence relations *All* and *Id* may be used to describe security "properties" *high* and *low*. It turns out that these are exactly the same as the interpretations given to the notions "dynamic" and "static" by the per-based model of BTA [19]. This means that the binding-time analysis for a higher-order functional language can also be read as a security information-flow analysis. This connection between security and binding-time analysis is already folklore (see e.g. a comparison [45] of a particular security type system and a particular binding-time analysis, and an investigation [8] which shows how the incorporation of indirect information flows from Denning's security analysis can improve binding-time analyses).

It is worth highlighting a few of the pertinent ideas from the per-based treatment of BTA [19]. Beginning with the equivalence relations *All* and *Id* to describe *high* and *low* respectively, there are two important extensions to the basic idea in order to handle *structured data types* and *higher-order functions*. Both of these ideas are handled by the analysis [19] which rather straightforwardly extends Launchbury's projection-based binding-time analysis [24] to higher types. To some extent the per model [19] anticipates the treatment of partially-secure data types in the SLam calculus [15], and the use of logical relations in their proof of noninterference.

For structured data it is useful to have more refined notions of security than just *high* and *low*; we would like to be able to model various *degrees* of security. For example, we may have a list of records containing name-password pairs. Assuming passwords are considered *high*, we might like to express the fact that although the whole list cannot be considered *low*, it can be considered as a $(low \times high)list$. Constructing equivalence relations which represent such properties is straightforward—see reference [19] for examples (which are adapted directly from Launchbury's work), and reference [17] for a more general treatment of finite lattices of "binding times" for recursive types.

To represent security properties of higher-order functions we use a less restricted class of relations than the equivalence relations. A *partial equivalence relation* (per) on a set D is a binary relation on D which is *symmetric* and *transitive*. If P is such a per let $|P|$ denote the domain of P, given by:

$$|P| = \{x \in D \mid x \; P \; x\}$$

Note that the domain and range of a per P are both equal to $|P|$ (so for any $x, y \in D$, if $x \; P \; y$ then $x \; P \; x$ and $y \; P \; y$), and that the restriction of P to $|P|$ is an equivalence relation. Clearly, an equivalence relation is just a per which is reflexive (so $|P| = D$). Partial equivalence relations over various applicative structures have been used to construct models of the polymorphic lambda calculus (see, for example, Abadi and Plotkin's paper [2]). As far as we are aware, the first use of pers in static program analysis is that presented in [18].

For a given set D let $Per(D)$ denote the partial equivalence relations over D. $Per(D)$ is a meet semi-lattice, with meets given by set-intersection, and top element All.

Given pers $P \in Per(D)$ and $Q \in Per(E)$, we may construct a new per $(P \rightarrow Q) \in Per(D \rightarrow E)$ defined by:

$$\begin{gathered} f \; (P \rightarrow Q) \; g \\ \iff \\ \forall x, x' \in D. \, x \; P \; x' \implies (f \; x) \; Q \; (g \; x') \end{gathered}$$

If P is a per, we will write $x : P$ to mean $x \in |P|$. This notation and the above definition of $P \rightarrow Q$ are consistent with the notation used previously, since now

$$\begin{aligned} & f : P \rightarrow Q \\ & \iff \; f \; (P \rightarrow Q) \; f \\ & \iff \; \forall x, x' \in D. \, x \; P \; x' \implies (f \; x) \; Q \; (f \; x') \end{aligned}$$

Note that even if P and Q are both total (i.e., equivalence relations), $P \rightarrow Q$ may be partial. A simple example is $All \rightarrow Id$. If $f : All \rightarrow Id$ then we know that given a *high* input, f returns a *low* output. A constant function $\lambda x.42$ has this property, but clearly not all functions satisfy this.

2.3. Observations on Strictness and Termination Properties

We are interested in the security properties of functions which are the denotations of programs (in a Scott-style denotational semantics), and so there are some termination issues which should be addressed. The formulation of security properties given above is sensitive to termination. Consider, for example, the following function $f : \mathbf{N}_\perp \to \mathbf{N}_\perp$

$$f = \lambda x.\mathsf{if}\ x > 0\ \mathsf{then}\ x\ \mathsf{else}\ fx$$

Clearly, if the argument is high then the result must be high. Now consider the security properties of the function $g \circ f$ where g is the constant function $g = \lambda x.2$. We might like to consider that g has type $high \to low$. However, if function application is considered to be strict (as in ML) then g is not in $|All_{\mathbf{N}_\perp} \twoheadrightarrow Id_{\mathbf{N}_\perp}|$ since $\perp\ All_{\mathbf{N}_\perp}\ 1$ but $g(\perp) \neq g(1)$. Hence the function $g \circ f$ does *not* have security type $high \to low$ (in our semantic interpretation). This is correct, since on termination of an application of this function, the low observer will have learned that the value of the high argument was positive.

The specific security analysis of e.g. the first calculus of Smith and Volpano [43] is termination sensitive—and this is enforced by a rather sweeping measure: all "while"-loop conditions must be low and all "while"-loop bodies must be low commands.

On the other hand, the type system of the SLam calculus [15] is not termination sensitive in general. This is due to the fact that it is based on a call-by-value semantics, and indeed the composition $g \circ f$ could be considered to have a security type corresponding to "$high \to low$". The correctness proof for noninterference carefully avoids saying anything about nonterminating executions. What is perhaps worth noting here is that had they chosen a non-strict semantics for application then the *same* type-system would yield termination sensitive security properties! So we might say that lazy programs are intrinsically more secure than strict ones. This phenomenon is closely related to properties of parametrically polymorphic functions [40][3]. From the type of a polymorphic function one can predict certain properties about its behaviour—the so-called "free theorems" of the type [49]. However, in a strict language one must add an additional condition in order that the theorems hold: the functions must be *bottom-reflecting* ($f(a) = \perp \implies a = \perp$). The same side condition can be added to make e.g. the type system of the SLam calculus termination-sensitive.

[3] Not forgetting that the use of pers in static analysis was inspired, in part, by Abadi and Plotkin's per model of polymorphic types [2].

To make this observation precise we introduce one further constructor for pers. If $R \in Per(D)$ then we will also let R denote the corresponding per on $D_\perp$ without explicit injection of elements from D into elements in $D_\perp$. We will write $R_\perp$ to denote the relation in $Per(D_\perp)$ which naturally extends R by $\perp R \perp$.

Now we can be more precise about the properties of g under a strict (call-by-value) interpretation: $g : (All_{\mathbf{N}})_\perp \twoheadrightarrow Id_{\mathbf{N}_\perp}$, which expresses that g is a constant function, modulo strictness. More informatively we can say that $g : All_{\mathbf{N}} \twoheadrightarrow Id_{\mathbf{N}}$ which expresses that g is a non-bottom constant function.

It is straightforward to express per properties in a subtype system of compositional rules (although we don't claim that such a system would be in any sense complete). Pleasantly, all the expected subtyping rules are sound when types are interpreted as pers and the subtyping relation is interpreted as subset inclusion of relations. For the abstract interpretation presented in [19] this has already been undertaken by e.g. Jensen [20] and Hankin and Le Métayer [14]. This natural subtyping relation also suggests a straightforward generalisation to arbitrary lattices of security levels.

2.4. Cohen's Formalisation of Flow Properties

In the case of information flow properties, the first programming-language-based formalisation of (in)security was due to Ellis Cohen [6, 7]. This was based on the view of programs as state transformers, as given by a denotational semantics (although there is no explicit discussion of nontermination issues). In [7], Cohen also considers "proof rules" for reasoning about flow properties of programs, and these are claimed to correspond to Denning's informal flow analysis conditions [10]. Here we recount the key definition of *strong dependency* from Cohen, and the related notion of *selective dependency*, and show how these notions can be expressed easily in the per setting.

For the sake of simplicity, we adapt Cohen's definitions slightly so as only to consider information flow between the initial value of a variable h, and the final value of some variable l. Assume that the computation state s is simply a record containing values $s.h$ and $s.l$ for the variables h and l respectively.

The notion of *strong dependency* between h and l, for a program C, written $h \rhd^C l$, is defined by:

$$h \rhd^C l \iff \exists s_1, s_2\, (s_1.l = s_2.l \;\;\&\;\; (\llbracket C \rrbracket s_1).l \neq (\llbracket C \rrbracket s_2).l)$$

This states that C conveys definite information flow from the initial value of h to the final value of l. Negating this condition we see that

$$\begin{aligned}\neg(h \rhd^{C} l) &\iff \neg\exists s_1, s_2\,(s_1.l = s_2.l \;\;\&\;\; ([\![C]\!]s_1).l \neq ([\![C]\!]s_2).l)\\ &\iff \forall s_1, s_2\,\neg(s_1.l = s_2.l \;\;\&\;\; ([\![C]\!]s_1).l \neq ([\![C]\!]s_2).l)\\ &\iff \forall s_1, s_2\,(s_1.l = s_2.l \Longrightarrow ([\![C]\!]s_1).l = ([\![C]\!]s_2).l)\end{aligned}$$

This corresponds to the expected noninterference condition. We have argued that this condition can be expressed using pers. But what is more interesting is Cohen's notion of *selective dependency*—since this can be used to model partial information flow.

Let us define the following strengthened form of non-dependency (corresponding to the negation of the predicate defined in [7], Definition 3-4). Let ϕ be a predicate on the value of h. The *selective independence* property, written $h \not\rhd^{C}_{\phi} l$, is defined by: $h \not\rhd^{C}_{\phi} l$ iff

$$\forall s_1, s_2\,(\phi(s_1.h) \;\;\&\;\; \phi(s_2.h) \;\;\&\;\; s_1.l = s_2.l \Longrightarrow ([\![C]\!]s_1).l = ([\![C]\!]s_2).l)$$

The role of ϕ is to place a restriction on the initial value of h, so as to ask whether, despite this constraint, there is information flow to l. Consider the program $C = \mathsf{if}\ h > 0\ \mathsf{then}\ l := h$. If we know that, for example, the value of h is negative, then we know that C cannot leak information, since the assignment $l := h$ can never be reached. This fact corresponds to the statement

$$h \not\rhd^{C}_{h<0} l$$

It is straightforward to express properties of this form using pers. The basic idea is to define the relation All^{ϕ} to be the relation All restricted to values satisfying ϕ. Define $u\ All^{\phi}\ v \iff \phi(u) \;\;\&\;\; \phi(v)$.

If we assume that a state s is simply a pair of the values of h and l respectively, then the information flow property $h \not\rhd^{C}_{\phi} l$ corresponds to

$$[\![C]\!] : (All^{\phi} \times Id) \rightarrow (All \times Id)_{\perp}$$

2.5. Partial Information Flow

The selective independence notation is used to make statements about partial information flow. We conclude with such an example (from [7]), and show how these statements can also be expressed (more) concisely with pers.

Let D be the program $l := h\ \mathsf{mod}\ 4$. Clearly there is information leakage, but the information leaked is only the two least significant bits of h. To express this using selective independence, Cohen suggests the following. For each $i \in \{0,1,2,3\}$, define the predicate

$$\phi_i(v_h) \Longleftrightarrow i = v_h \pmod 4$$

Now the collection of predicates $\{\phi_i\}_{i=0,1,2,3}$ can be used to define the flow property of the program:

$$\forall i \in \{0,1,2,3\} \,.\, h \not\rhd^{D}_{\phi_i} l$$

Clearly we can make a similar "for all" statement using pers. In the per setting this can be expressed as a single statement. Let $\hat{\phi}$ denote the equivalence relation formed by taking the set of states satisfying each ϕ_i to be the equivalence classes of the relation. This corresponds to the relation given by $\cup_i All^{\phi_i}$. Then we can express the selective independence property as:

$$[\![D]\!] : (\hat{\phi} \times Id) \twoheadrightarrow (All \times Id)_\perp$$

The equivalence of this statement to Cohen's formulation follows from the general fact that for any disjoint pers P and Q we have that $P \cup Q$ is a per, and that

$$f : P \twoheadrightarrow S \;\;\&\;\; f : Q \twoheadrightarrow S \;\Longleftrightarrow\; f : (P \cup Q) \twoheadrightarrow S$$

3. Nondeterministic Information Flow

In this section we show how the per model of security can be extended to describe nondeterministic computations. We see nondeterminism as an important feature as it arises naturally when considering the semantics of a concurrent language (although the treatment of a concurrent language remains outside the scope of the present paper.)

In order to focus on the essence of the problem we consider a very simplified setting—the analysis of commands in some simple imperative language containing a nondeterministic choice operator. We assume that there is some discrete (i.e., unordered) domain **St** of states (which might be viewed as finite maps from variables to discrete values, or simply just a tuple of values).

3.1. Secure Commands in a Deterministic Setting

In the deterministic setting we can take the denotation of a command C, written $[\![C]\!]$, to be a function in $[\mathbf{St}_\perp \to \mathbf{St}_\perp]$, where by $[D_\perp \to E_\perp]$

we mean the set of strict and continuous maps between domains $D_\bot$ and $E_\bot$. Note that we could equally well take the set of all (trivially continuous) functions in $\mathbf{St} \to \mathbf{St}_\bot$, which is isomorphic. The fact that $\mathbf{St}_\bot$ is no more than a flat domain (i.e., a domain of height two) will be important in the technical development.

Now suppose that the state is just a simple partition into a high-security half and a low-security half, so the set of states is the product $\mathbf{St}_{high} \times \mathbf{St}_{low}$. Then we might define a command C to be secure if no information from the high part of the state can leak into the low part:

$$C \text{ is secure } \iff$$
$$[\![C]\!] : (All \times Id)_\bot \twoheadrightarrow (All \times Id)_\bot$$

This is equivalent to saying that $[\![C]\!] : (All \times Id) \twoheadrightarrow (All \times Id)_\bot$ since we only consider strict functions. Note that this does not imply that $[\![C]\!]$ terminates, but that the termination behaviour is not influenced by the values of the high part of the state. It is easy to see that the sequential composition of secure commands is a secure command, since firstly, the denotation of the sequential composition of commands is just the function-composition of denotations, and secondly, in general for functions $g : D \to E$ and $f : E \to F$, and pers $P \in Per(D)$, $Q \in Per(E)$ and $R \in Per(F)$ it is easy to verify the soundness of the inference rule:

$$\frac{g : P \twoheadrightarrow Q \qquad f : Q \twoheadrightarrow R}{f \circ g : P \twoheadrightarrow R}$$

3.2. Powerdomain Semantics for Nondeterminism

A standard approach to giving meaning to a nondeterministic language—for example Dijkstra's guarded command language—is to interpret a command as a mapping which yields a *set* of results. However, when defining an ordering on the results in order to obtain a domain, there is a tension between the internal order of $\mathbf{St}_\bot$ and the subset order of the powerset. This is resolved by considering a suitable *powerdomain* structure [39, 44]. The idea is to define a preorder on the *finitely generated* (f.g.) subsets (those non-empty subsets which are either finite, or contain $\bot$) of $\mathbf{St}_\bot$ in terms of the order on their elements. By quotienting "equivalent sets" one obtains a partial ordering, each depending on a different view of what sets of values should be considered equivalent. Consider the following three programs (an example from Plotkin's notes on Domain Theory [38]):

$$(1)\ x := 1 \qquad (2)\ x := 1 \,[\!]\, \mathsf{loop} \qquad (3)\ \mathsf{loop}$$

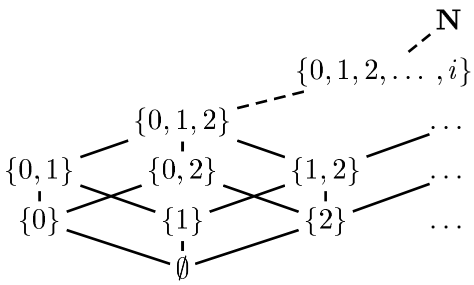
In the "Hoare" or *partial correctness* interpretation the first two programs are considered to be equal since, ignoring nontermination, they yield the same sets of outcomes. This view motivates the definition of the *Hoare* or *lower* powerdomain, $\mathcal{P}_{\mathsf{L}}[\mathbf{St}_\perp]$.

In the "Smyth" or *total correctness* interpretation, programs (2) and (3) are considered equal (equally bad!) because neither of them can guarantee an outcome. In the general case this view motivates the *Smyth* or *upper* powerdomain, $\mathcal{P}_{\mathsf{U}}[\mathbf{St}_\perp]$ [44].

In the "Egli-Milner" interpretation (leading to the *convex* or *Plotkin* powerdomain in the general case) all three programs are considered to have distinct denotations.

A domain D is *flat* whenever it has a trivial partial ordering ($\forall x, y \in D.\, x \sqsubseteq y \iff x = y$ or $x = \perp$). Examples of flat domains are $\mathbf{N}_\perp$ and $\mathbf{St}_\perp$. A *discrete powerdomain* is the powerdomain of a flat domain. We give a formal definition of each powerdomain construction in turn, and give an idea about the corresponding discrete powerdomain $\mathcal{P}[\mathbf{St}_\perp]$.

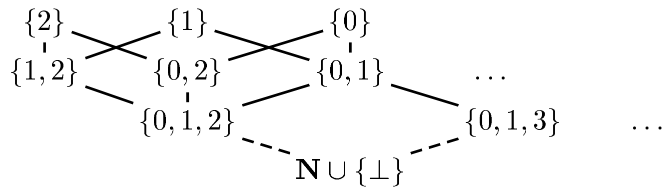
Lower powerdomain Let $A \preceq_{\mathsf{L}} B$ iff $\forall x \in A.\, \exists y \in B.\, x \sqsubseteq y$. In this case the induced discrete powerdomain $\mathcal{P}_{\mathsf{L}}[\mathbf{St}_\perp]$ is isomorphic to the powerset of $\mathbf{St}$ ordered by subset inclusion. This means that the domain $[\mathbf{St}_\perp \to \mathcal{P}_{\mathsf{L}}[\mathbf{St}_\perp]]$ is isomorphic to all subsets of $\mathbf{St} \times \mathbf{St}$— i.e. the *relational semantics.* The Hasse diagram for the construction isomorphic to $\mathcal{P}_{\mathsf{L}}[\mathbf{N}_\perp]$ is:



Upper powerdomain The upper ordering on f.g. sets A, B, is given by:

$$A \preceq_{\mathsf{U}} B \iff \forall y \in B.\, \exists x \in A.\, x \sqsubseteq y$$

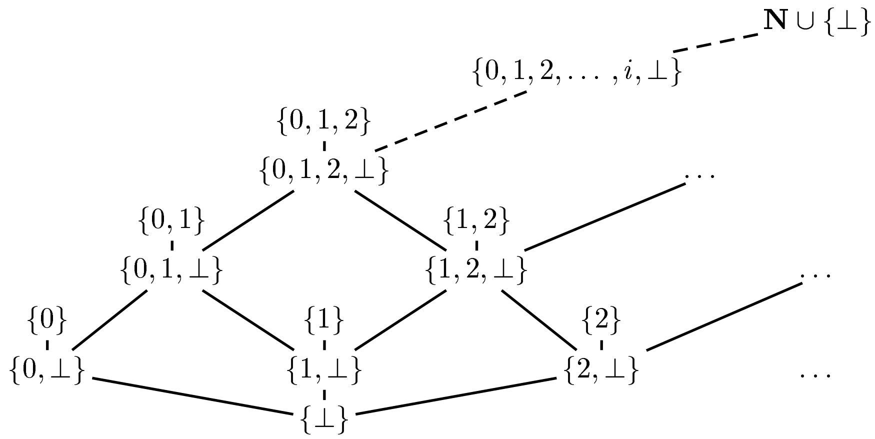
Here the induced discrete powerdomain $\mathcal{P}_{\mathsf{U}}[\mathbf{St}_\perp]$ is isomorphic to the set of finite non-empty subsets of $\mathbf{St}$ together with $\mathbf{St}_\perp$ itself, ordered by superset inclusion. The Hasse diagram for the construction isomorphic to $\mathcal{P}_{\mathsf{U}}[\mathbf{N}_\perp]$ is:



Convex powerdomain Let $A \preceq_{\mathsf{C}} B$ iff $A \preceq_{\mathsf{U}} B$ and $A \preceq_{\mathsf{L}} B$. This is also known as the Egli-Milner ordering. The resulting powerdomain $\mathcal{P}_{\mathsf{C}}[\mathbf{St}_{\perp}]$ is isomorphic to the f.g. subsets of $\mathbf{St}_{\perp}$, ordered by:

$$A \sqsubseteq_{\mathsf{C}} B \iff \begin{array}{l} \text{either } \perp \notin A \;\;\&\;\; A = B \\ \text{or } \perp \in A \;\;\&\;\; A \setminus \{\perp\} \subseteq B \end{array}$$

The Hasse diagram for $\mathcal{P}_{\mathsf{C}}[\mathbf{N}_{\perp}]$ is:



NOTE 1. *The isomorphisms mentioned and depicted in the diagrams above will be used implicitly in a number of constructions in this section in order to give pointwise definitions and proofs about powerdomains. (This is also the reason that the following development does not directly generalise from discrete to arbitrary powerdomains.) We will use the subsets from the isomorphic constructions to represent elements of the powerdomains $\mathcal{P}[\mathbf{St}_{\perp}]$ (the equivalence classes of f.g. subsets). In the case of $\mathcal{P}_{\mathsf{C}}[\mathbf{St}_{\perp}]$ (or $\mathcal{P}_{\mathsf{C}}[\mathbf{N}_{\perp}]$) it is straightforward—the equivalence classes are singleton sets each containing an f.g. set (see the above diagram for $\mathcal{P}_{\mathsf{C}}[\mathbf{N}_{\perp}]$).*

A few basic properties and definitions on powerdomains will be needed. For each powerdomain constructor $\mathcal{P}[-]$ define the order-preserving "unit" map $\eta_D : D \to \mathcal{P}[D]$ which takes each element $a \in D$

into (the powerdomain equivalence class of) the singleton set $\{a\}$. For each function $f \in [D \to \mathcal{P}[E]]$ we define the *extension* of f, denoted f^*, where $f^* \in [\mathcal{P}[D] \to \mathcal{P}[E]]$ by:

$$f^*(A) = \cup_{x \in A} f(x)$$

For each powerdomain constructor $\mathcal{P}[-]$ the structure $(\mathcal{P}[D], \eta_D(x), *)$ is a *Kleisli triple* or, alternatively, a *monad*, and thus satisfies the Kleisli triple laws [27]:

$$\begin{aligned} \eta_D^* &= id_{\mathcal{P}[D]} \\ f &= f^* \circ \eta_D \\ g^* \circ f^* &= (g^* \circ f)^* \end{aligned}$$

This provides a canonical way of composing the semantics of programs. In the particular setting of the denotations of commands, it is worth noting that $[\![C_1; C_2]\!]$ would be given by:

$$[\![C_1; C_2]\!] = [\![C_2]\!]^* \circ [\![C_1]\!]$$

3.3. Pers on Powerdomains

Given one of the discrete powerdomains, $\mathcal{P}[\mathbf{St}_\perp]$, we will need a "logical" way to lift a per $P \in Per(\mathbf{St}_\perp)$ to a per in $Per(\mathcal{P}[\mathbf{St}_\perp])$.

DEFINITION 1. *For each $R \in Per(D_\perp)$ and each choice of power domain $\mathcal{P}[-]$, let $\mathcal{P}[R]$ denote the relation on $\mathcal{P}[D_\perp]$ given by:*

$$\begin{aligned} A\, \mathcal{P}[R]\, B \iff & \forall a \in A.\, \exists b \in B.\, a\, R\, b \\ \& \quad & \forall b \in B.\, \exists a \in A.\, a\, R\, b \end{aligned}$$

where we implicitly adopt the set-based representation of the elements of the powerdomain (Note 1).

It is easy to check that $\mathcal{P}[R]$ is a per, and in particular that $\mathcal{P}[Id_{D_\perp}] = Id_{\mathcal{P}[D_\perp]}$.

Henceforth we shall restrict our attention to the semantics of simple commands, and hence the three discrete powerdomains $\mathcal{P}[\mathbf{St}_\perp]$.

PROPOSITION 1. *For any $f \in [\mathbf{St}_\perp \to \mathcal{P}[\mathbf{St}_\perp]]$ and any R, $S \in Per(\mathbf{St}_\perp)$,*

$$f : R \twoheadrightarrow \mathcal{P}[S] \iff f^* : \mathcal{P}[R] \twoheadrightarrow \mathcal{P}[S]$$

From this it easily follows that the following inference rule is sound:

$$\frac{[\![C_1]\!] : P \twoheadrightarrow \mathcal{P}[Q] \qquad [\![C_2]\!] : Q \twoheadrightarrow \mathcal{P}[R]}{[\![C_1; C_2]\!] : P \twoheadrightarrow \mathcal{P}[R]}$$

3.4. The Security Condition

We will investigate the implications of the security condition under each of the powerdomain interpretations. Let us suppose that, as before the state is partitioned into a high part and a low part: $\mathbf{St} = \mathbf{St}_{high} \times \mathbf{St}_{low}$. With respect to a particular choice of powerdomain let the security "type" $C : high \times low \to high \times low$ denote the property

$$[\![C]\!] : (All \times Id)_\perp \twoheadrightarrow \mathcal{P}[(All \times Id)_\perp]$$

In this case we say that C is secure. Now we explore the implications of this definition on each of the possible choices of powerdomain:

1. In the lower powerdomain, the security condition describes in a weak sense termination-insensitive information flow. For example, the program

   $$\mathsf{if}\ h = 0\ \mathsf{then\ skip}\ [\!]\ \mathsf{loop\ else\ skip}$$

   (h is the high part of the state) is considered secure under this interpretation but the termination behaviour is influenced by h (it can fail to terminate only when $h = 0$).

2. In the upper powerdomain nontermination is considered catastrophic. This interpretation seems completely unsuitable for security unless one only considers programs which are "totally correct"— i.e. which must terminate on their intended domain. Otherwise, a possible nonterminating computation path will mask any other insecure behaviours a term might exhibit. This means that for *any* program C, the program $C\ [\!]\ \mathsf{loop}$ is secure!

3. The convex powerdomain gives the appropriate generalisation of the deterministic case in the sense that it is termination sensitive, and does not have the shortcomings of the upper powerdomain interpretation.

4. Relation to an Equational Characterisation

In this section we relate the per-based security condition to a proposal by Leino and Joshi [25]. As before, assume for simplicity we have programs with just two variables: h and l of high and low secrecy respectively. Assume that the state is simply a pair, where h refers to the first projection and l is the second projection.

Leino and Joshi [25] define the security condition for a program C by:

$$HH; C; HH = C; HH$$

where "=" stands for semantic equality (the style of semantic specification is left unfixed), and HH is the program that "assigns to h arbitrary values"—aka "Havoc on H". We will refer to this equation as the equational security condition. Intuitively, the equation says that we cannot learn anything about the initial values of the high variables by variation of the low security variables. The postfix occurrences of HH on each side mean that we are only interested in the final value of l. The prefix HH on the left-hand side means that the two programs are equal if the final value of l does not depend on the initial value of h.

In relating the equational security condition to pers we must first decide upon the denotation of HH. Here [25] we run into some potential problems since it is necessary that HH always terminates, but nevertheless exhibits unbounded nondeterminism. Although this appears to pose no problems in [25] (in fact it goes without mention), to handle this we would need to work with non-ω-continuous semantics, and powerdomains for unbounded nondeterminism. Instead, we side-step the issue by assuming that the domain of h, $\mathbf{St}_{high}$, is finite. Suppose that $\mathbf{St}_{high} = \{v_1, \ldots, v_n\}$. Under this assumption, the definition of HH can be given by the command:

$$h := v_1 \, [\!] \cdots [\!] \, h := v_n$$

and its denotation can be written:

$$[\![HH]\!](\bot) = \{\bot\} \qquad [\![HH]\!](h, l) = \{(h', l) \mid h' \in \mathbf{St}_{high}\}$$

4.1. Equational Security and Projection Analysis

A first observation is that the the equational security condition is strikingly similar to the well-known form of static analysis for functional programs known as projection analysis [50]. Given a function f, a projection analysis aims to find projections (continuous lower closure operators on the domain) α and β such that

$$\beta \circ f \circ \alpha = \beta \circ f$$

For (generalised) strictness analysis and dead-variable analysis, one is given β, and α is to be determined; for binding time analysis [24] it is a forwards analysis problem: given α one must determine some β.

For strict functions (e.g., the denotations of commands) projection analysis is not so readily applicable. However, in the convex power-domain *HH* is rather projection-like, since it effectively hides all information about the high variable; in fact it is an embedding (an upper closure operator) so the connection is rather close.

4.2. The Equational Security Condition Is Subsumed by the Per Security Condition

Hunt [18] showed that projection properties of the form $\beta \circ f \circ \alpha = \beta \circ f$ could be expressed naturally as a per property of the form $f : R_\alpha \multimap R_\beta$ for equivalence relations derived from α and β by relating elements which get mapped to the same point by the corresponding projection.

Using the same idea we can show that the per-based security condition subsumes the equation specification in a similar manner.

We will establish the following:

THEOREM 1. *For any command C*

$$\llbracket HH; C; HH \rrbracket = \llbracket C; HH \rrbracket$$
$$\textit{iff}$$
$$C : \mathit{high} \times \mathit{low} \to \mathit{high} \times \mathit{low}$$

The idea will be to associate an equivalence relation to the function HH. More generally, for any command C let $\mathit{ker}(C)$, the *kernel* of C, denote the relation on $\mathbf{St}_\perp$ satisfying

$$s_1 \; \mathit{ker}(C) \; s_2 \iff \llbracket C \rrbracket s_1 = \llbracket C \rrbracket s_2$$

Extend $\mathit{ker}(C)$ to be the relation $\mathit{ker}^*(C)$ on $\mathcal{P}[\mathbf{St}_\perp]$ given by:

$$A \; \mathit{ker}^*(C) \; B \iff \llbracket C \rrbracket^* A = \llbracket C \rrbracket^* B$$

Recall the per interpretation of the type signature of C.

$$C : \mathit{high} \times \mathit{low} \to \mathit{high} \times \mathit{low}$$
$$\iff$$
$$\llbracket C \rrbracket : (\mathit{All} \times \mathit{Id})_\perp \multimap \mathcal{P}[(\mathit{All} \times \mathit{Id})_\perp]$$

Observe that $(All \times Id)_\perp = ker(HH)$ since for any h, l, h', l' it holds $[\![HH]\!](h,l) = [\![HH]\!](h',l')$ iff $l = l'$ iff $(h,l)(All \times Id)_\perp(h',l')$.

The proof of the theorem is based on this observation and on the following two facts:

- $\mathcal{P}[(All \times Id)_\perp] = ker^*(HH)$ and
- $[\![HH; C; HH]\!] = [\![C; HH]\!] \iff [\![C]\!] : ker(HH) \twoheadrightarrow ker^*(HH)$.

Let us first prove the latter fact by proving a more general statement similar to Proposition 3.1.5 from Hunt's thesis [17] (the correspondence between projections and per-analysis). Note that we do not use the specifics of the convex powerdomain semantics here, so the proof is valid for any of the three choices of powerdomain.

THEOREM 2. *Let us say that a command B is* idempotent *iff* $[\![B; B]\!] = [\![B]\!]$. *For any commands C and D, and any idempotent command B*

$$[\![B; C; D]\!] = [\![C; D]\!] \iff [\![C]\!] : ker(B) \twoheadrightarrow ker^*(D)$$

Proof. $(\Rightarrow)$: Assume $[\![B; C; D]\!] = [\![C; D]\!]$. Let s_1, s_2 denote arbitrary states such that $s_1 \; ker(B) \; s_2$. We are required to show that $[\![C]\!]s_1 \; ker^*(D) \; [\![C]\!]s_2$. Now

$$\begin{array}{ll}
[\![C; D]\!]s_1 & \\
= [\![B; C; D]\!]s_1 & \text{assn.} \\
= [\![C; D]\!]^* \circ [\![B]\!]s_1 & \\
= [\![C; D]\!]^* \circ [\![B]\!]s_2 & ker(B) \text{ defn.} \\
= [\![B; C; D]\!]s_2 & \\
= [\![C; D]\!]s_2. &
\end{array}$$

Now $[\![C; D]\!]s_1 = [\![C; D]\!]s_2$ implies

$$[\![D]\!]^*([\![C]\!]s_1) = [\![D]\!]^*([\![C]\!]s_2)$$

i.e.,

$$[\![C]\!]s_1 \; ker^*(D) \; [\![C]\!]s_2$$

as required.

$(\Leftarrow$:) Assume $[\![C]\!] : ker(B) \twoheadrightarrow ker^*(D)$. Now $[\![B]\!]^*([\![B]\!]^*A) = [\![B]\!]^*A$ by idempotence, which implies

$$\begin{array}{lll}
[\![B]\!]^*A \; ker^*(B) \; A & & \\
\Longrightarrow & [\![C]\!]^*([\![B]\!]^*A) \; ker^*(D) \; [\![C]\!]^*A & \text{assn.} \\
\iff & [\![D]\!]^*([\![C]\!]^*([\![B]\!]^*A)) = [\![D]\!]^*([\![C]\!]^*A) & \text{defn. } ker^*(D) \\
\Longrightarrow & [\![B; C; D]\!]^*A = [\![C; D]\!]^*A & \Box
\end{array}$$

Corollary. Since $[\![HH]\!]$ is idempotent we can conclude that

$$[\![HH;C;HH]\!] = [\![C;HH]\!] \iff [\![C]\!] : ker(HH) \twoheadrightarrow ker^*(HH)$$

It remains to establish the first fact.

THEOREM 3. $\mathcal{P}[(All \times Id)_\perp] = ker^*(HH)$

Proof. We reason "pointwise" using the implicit representation of A and B as sets of states (cf. Note 1). ($\subseteq$): Suppose $A\,\mathcal{P}[(All \times Id)_\perp]\,B$, we need to show $[\![HH]\!]^*A = [\![HH]\!]^*B$. Cases

$$\begin{array}{lll}
\perp \in [\![HH]\!]^*A & & \\
\quad \iff & \perp \in A & HH\ \perp\text{-reflecting} \\
\quad \implies & \perp \in B & \\
\quad \iff & \perp \in [\![HH]\!]^*B & \\
(h,l) \in [\![HH]\!]^*A & & \\
\quad \iff & \exists h_0.\,(h_0,l) \in A & \text{defn. } (\cdot)^* \\
\quad \implies & \exists h_1.\,(h_1,l) \in B & \\
\quad \iff & \forall h'.\,(h',l) \in [\![HH]\!]^*B &
\end{array}$$

($\supseteq$): For the other direction assume $[\![HH]\!]^*A = [\![HH]\!]^*B$.

$$\begin{array}{rcl}
\perp \in A & \implies & \perp \in B \\
(h,l) \in A & \iff & \exists h_0.\,(h_0,l) \in [\![HH]\!]^*A \\
& \implies & \exists h_0.\,(h_0,l) \in [\![HH]\!]^*B \\
& \iff & \exists h_1.\,(h_1,l) \in B \qquad \square
\end{array}$$

Thus, the equational and per security conditions in this simple case are equivalent.

4.3. Abstract Variables

In a more recent extension of the paper, [26], Leino and Joshi update their relational semantics to handle termination-sensitive leakages and introduce abstract variables—a way to support partially confidential data. Abstract variables h and l are defined as functions of the concrete variables in a program. For example, for a list of low length and high elements, l would be the length of the list and h would be the list itself. In the general case the choice of h and l could be dependent, so an *independence condition* must be verified.

Abstract variables are easily represented in our setting. Suppose that some function $g \in \mathbf{St} \to D$ yields the value (in some domain D) of the abstract low variable from any given state, then we can represent the security condition on abstract variables by: $[\![C]\!] : R_g \twoheadrightarrow \mathcal{P}[(All \times Id)_\perp]$ where $s_1 R_g s_2 \iff g\, s_1 = g\, s_2$.

5. A Probabilistic Security Condition

There are still some weaknesses in the security condition when interpreted in the convex powerdomain when it comes to the consideration of nondeterministic programs. In the usual terminology of information flow, we have considered *possibilistic information flows.* The probabilistic nature of an implementation may allow *probabilistic information flows* for "secure" programs. Consider the program

$$h := h \;\texttt{mod}\; 100; (l := h \;[\!]\; l := \texttt{rand}(99))$$

This program is secure in the convex powerdomain interpretation since regardless of the initial value of h, the final value of l can be any value in the range $\{0 \ldots 99\}$. But with a reasonably fair implementation of the nondeterministic choice and of the randomised assignment, it is clear that a few runs of the program, for a fixed input value of h, could yield a rather clear indication of its value by observing only the possible final values of l, e.g.:

$$17, 2, 45, 2, 2, 33, 2, 97, 2, 8, 57, 2, 2, 66, \ldots$$

—from which we might reasonably conclude that the value of h was 2. This observation is not new to this paper. It was first addressed in an abstract setting by Gray [13]. The above example is adapted from McLean [29]. In the setting of a concrete imperative language, Volpano and Smith recently devised a probabilistic security type-system [47] with a soundness proof based on an adaptation of Kozen's probabilistic operational semantics [22, 23]. They do not provide a *definition* of what it means for a program to be secure, and the security condition implicit in their correctness argument is not directly comparable—due to the fact that they consider parallel deterministic threads and a non-compositional semantics.

To counter the problem indicated by the example we consider *probabilistic powerdomains* [21] which allow the probabilistic nature of choice to be reflected in the semantics of programs, and hence enable us to capture the fact that varying the value of h causes *a change in the probability distribution* of values of l.

5.1. Probabilistic Powerdomain of Distributions

In the "possibilistic" setting we had the denotation of a command C to be a continuous function in $[\mathbf{St}_\perp \to \mathcal{P}_\mathsf{C}[\mathbf{St}_\perp]]$. In the probabilistic case, given an input to C not only do we keep track of possible outputs, but also of probabilities at which they appear. Thus, we consider a domain $\mathcal{E}[\mathbf{St}_\perp]$ of distributions over $\mathbf{St}_\perp$. The denotation of C is going to be a function in $[\mathbf{St}_\perp \to \mathcal{E}[\mathbf{St}_\perp]]$.

The general probabilistic powerdomain construction [21] on an inductive partial order $\mathcal{E}[D]$ is taken to be the domain of *evaluations*, which are certain continuous functions on $\Omega(D) \to [0,1]$, where $\Omega(D)$ is the lattice of open subsets of D. We will omit a description of the general probabilistic powerdomain of evaluations since for the present paper it is sufficient and more intuitive to work with discrete powerdomains, and hence a simplified notion of probabilistic powerdomain in terms of distributions. Evaluations rather than distributions would be needed to give a probabilistic semantics to a concurrent language based on the domain of *resumptions* [21]—but this is beyond the scope of the present article.

If S is a set (e.g., the domain of states for a simple sequential language) then we define the probabilistic powerdomain of $S_\perp$, written $\mathcal{E}[S_\perp]$ to be the domain of *distributions* on $S_\perp$, where a distribution μ is a function from $S_\perp$ to $[0,1]$ such that $\sum_{d \in S_\perp} \mu d = 1$. The ordering on $\mathcal{E}[S_\perp]$ is defined pointwise by $\mu \leq \nu$ iff $\forall d \neq \perp. \, \mu d \leq \nu d$. This structure is isomorphic to Jones and Plotkin's probabilistic powerdomain of evaluations for this special case.

As a simple instance of the probabilistic powerdomain construction [21], one can easily see that $\mathcal{E}[S_\perp]$ is an inductively complete partial order with directed lubs defined pointwise, and with a least element $\eta_{S_\perp}(\perp)$, where $\eta_{S_\perp}$ is the *point-mass distribution* defined for an $x \in S_\perp$ by:

$$\eta_{S_\perp}(x)d = \begin{cases} 1, & \text{if } d = x \\ 0, & \text{otherwise} \end{cases}$$

To lift a function $f : D_1 \to \mathcal{E}[D_2]$ to type $\mathcal{E}[D_1] \to \mathcal{E}[D_2]$ we define the *extension* of f by:

$$f^*(\mu)(y) = \sum_{x \in D_1} f(x)(y) \cdot \mu(x)$$

As in the case of the powerdomains for nondeterminism, the structure $(\mathcal{E}[D], \eta_D(x), *)$ is a Kleisli triple, and thus, similarly to Section 3.2, we have a mechanism for composing the probabilistic semantics of any two given programs. Suppose $f : D_1 \to \mathcal{E}[D_2]$ and $g : D_2 \to \mathcal{E}[D_3]$ are such. Then the lifted composition $(g^* \circ f)^*$ can be computed by one of the Kleisli triple laws as $g^* \circ f^*$.

5.2. Pers on Probabilistic Powerdomains

The next step towards the security condition is to define how pers work on discrete probabilistic powerdomains. To lift pers to $\mathcal{E}[D]$ we need to consider a definition which takes into consideration the whole of each R-equivalence class in one go. The intuition is that an equivalence class of a per is a set of points that are indistinguishable by a low-level observer. For a given evaluation, the probability of a given observation by a low level user is thus the sum of probabilities over all elements of the equivalence class.

For a per $R \in Per(D)$ we define D/R to be the partition of $|D|$ onto R-equivalence classes. Given $e_R \in D/R$ and a distribution μ on D, let $\mu(e_R)$ be short-hand for $\sum_{x \in e_R} \mu(x)$. Define the per relation $\mathcal{E}[R]$ on $\mathcal{E}[D]$ for $\mu, \nu \in \mathcal{E}[D]$ by:

$$\mu \; \mathcal{E}[R] \; \nu \iff \forall e_R \in D/R. \, \mu(e_R) = \nu(e_R)$$

Naturally, $\mu \; \mathcal{E}[Id] \; \nu \iff \mu = \nu$ and $\forall \mu, \nu \in \mathcal{E}[D]. \, \mu \; \mathcal{E}[All] \; \nu$.

As an example, consider $\mathcal{E}[(All \times Id)_\perp]$. For two distributions μ and ν on $\mathbf{St}_\perp$ we have $\mu \; \mathcal{E}[(All \times Id)_\perp] \; \nu$ iff the probability of any given low value l in the left-hand distribution, given by $\sum_h \mu(h, l)$, is equal to the probability in the right-hand distribution, namely $\sum_h \nu(h, l)$.

The probabilistic security condition is indeed a strengthening of the possibilistic one—when we consider programs whose possibilistic and probabilistic semantics are in agreement.

THEOREM 4. *Suppose we have a possibilistic (convex) semantics* $[\![\cdot]\!]_{\mathsf{C}}$ *and a probabilistic semantics* $[\![\cdot]\!]_{\mathcal{E}}$*, which satisfy a basic consistency property that for any command* C*, if* $[\![C]\!]_{\mathcal{E}} i \, o > 0$ *then* $o \in [\![C]\!]_{\mathsf{C}} i$.

Now suppose that R *and* S *are equivalence relations on* D*. Suppose further that* C *is any command such that possibilistic behaviour agrees with its probabilistic behaviour, i.e.,* $o \in [\![C]\!]_{\mathsf{C}} i \implies [\![C]\!]_{\mathcal{E}} i \, o > 0$*. Then we have that* $[\![C]\!]_{\mathcal{E}} : R \twoheadrightarrow \mathcal{E}[S]$ *implies* $[\![C]\!]_{\mathsf{C}} : R \twoheadrightarrow \mathcal{P}_{\mathsf{C}}[S]$.

Proof. For some $a, b \in D$ let $\mu = [\![C]\!]_{\mathcal{E}} a$ and $\nu = [\![C]\!]_{\mathcal{E}} b$. By $[\![C]\!]_{\mathcal{E}} : R \twoheadrightarrow \mathcal{E}[S]$ deduce $a \; R \; b \implies \forall d \in D. \, \mu([d]_S) = \nu([d]_S)$. What we need to prove is $a \; R \; b \implies [\![C]\!]_{\mathsf{C}} a \; \mathcal{P}_{\mathsf{C}}[S] \; [\![C]\!]_{\mathsf{C}} b$. So, assume $a \; R \; b$ and let us show that

$$\forall x \in [\![C]\!]_{\mathsf{C}} a. \, \exists y \in [\![C]\!]_{\mathsf{C}} b. \, x \; S \; y$$

Take any $x \in [\![C]\!]_{\mathsf{C}} a$. By the agreement hypothesis $\mu x > 0$, since if x is a possible output of program C run on data a, then the probability of getting this output must be greater then 0. So $\mu([x]_S) > 0$ and therefore $\nu([x]_S) > 0$. There must exist a $y \in [x]_S$ such that $\nu y > 0$. Thus, by the consistency property of the theorem's assumption, y is a possible output of program C run on data b, and $y \in [\![C]\!]_{\mathsf{C}} b$. The fact that $y \in [x]_S$ implies $x \; S \; y$. $\square$

5.3. Probabilistic Security Condition for the Case of Two Variables

Let us derive the probabilistic powerdomain security condition for the case of two variables h and l and domain $D = \mathbf{St}_\perp$. C is secure iff

$$
\begin{aligned}
&[\![C]\!]_{\mathcal{E}} : (All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp] \\
&\iff (i_h, i_l) \; (All \times Id)_\perp \; (i'_h, i'_l) \implies [\![C]\!]_{\mathcal{E}}(i_h, i_l) \; \mathcal{E}[(All \times Id)_\perp] \; [\![C]\!]_{\mathcal{E}}(i'_h, i'_l) \\
&\iff \forall i_h, i'_h, i_l . \, [\![C]\!]_{\mathcal{E}}(i_h, i_l) \; \mathcal{E}[(All \times Id)_\perp] \; [\![C]\!]_{\mathcal{E}}(i'_h, i_l) \\
&\text{Let } \mu = [\![C]\!]_{\mathcal{E}}(i_h, i_l) \text{ and } \nu = [\![C]\!]_{\mathcal{E}}(i'_h, i_l) \\
&\iff \forall (o_h, o_l) \in \mathbf{St}_\perp . \, \mu([(o_h, o_l)]_{(All \times Id)_\perp}) = \nu([(o_h, o_l)]_{(All \times Id)_\perp}) \\
&\iff \forall o_l \in \mathbf{St}_{low} . \mu\perp = \nu\perp \;\; \& \;\; \textstyle\sum_{o_h \in \mathbf{St}_{high}} \mu(o_h, o_l) = \sum_{o_h \in \mathbf{St}_{high}} \nu(o_h, o_l)
\end{aligned}
$$

So, a command C is secure iff

$$[\![C]\!]_{\mathcal{E}}(i_h, i_l) \perp = [\![C]\!]_{\mathcal{E}}(i'_h, i_l) \perp \;\; \&$$

$$\sum_{o_h \in \mathbf{St}_{high}} [\![C]\!]_{\mathcal{E}}(i_h, i_l)(o_h, o_l) = \sum_{o_h \in \mathbf{St}_{high}} [\![C]\!]_{\mathcal{E}}(i'_h, i_l)(o_h, o_l)$$

for any i_l, i_h, i'_h and o_l. Intuitively the equation means that if you vary i_h the distribution of low variables does not change. The sums provide projecting out the high variable.

Let us introduce probabilistic powerdomain semantics definitions for some language constructs. Here we omit the $\mathcal{E}$-subscripts to mean the probabilistic semantics. Given two programs C_1, C_2 such that $[\![C_1]\!] : \mathbf{St}_\perp \to \mathcal{E}[\mathbf{St}_\perp]$ and $[\![C_2]\!] : \mathbf{St}_\perp \to \mathcal{E}[\mathbf{St}_\perp]$ the composition of two program semantics is defined by:

$$[\![C_1; C_2]\!] \; i \; o = \sum_{s \in \mathbf{St}_\perp} ([\![C_1]\!] \; i \; s) \cdot ([\![C_2]\!] \; s \; o)$$

The semantics of the uniformly distributed choice $C_1 [\!] \, C_1$ is defined by:

$$[\![C_1 \; [\!] \; C_2]\!] \; i \; o = 0.5 [\![C_1]\!] \; i \; o + 0.5 [\![C_2]\!] \; i \; o$$

$$\begin{aligned} C ::= \;& \mathsf{skip} \mid Id := Exp \mid C_1; C_2 \mid C_1 \mathbin{[\!]_p} C_2 \\ & \mid \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid \mathsf{while}\ B\ \mathsf{do}\ C \end{aligned}$$

Figure 1. Command syntax

In Section 6 we give the semantics of other language constructs.

Example. Recall the program

$$h := h\ \mathsf{mod}\ 100; (l := h \mathbin{[\!]} l := \mathsf{rand}(99))$$

Now we investigate the security condition by varying the initial value of h from 0 to 1. Take $i_l = 0, i_h = 0, i'_h = 1$ and $o_l = 0$. The left-hand side is:

$$\sum_{o_h \in [0,\ldots,99]} [\![C]\!]_{\mathcal{E}}(0,0)(o_h,0) = 0.5 \cdot 1 + 0.5 \cdot 0.01 = 0.505$$

whereas the right-hand side is:

$$\sum_{o_h \in [0,\ldots,99]} [\![C]\!]_{\mathcal{E}}(1,0)(o_h,0) = 0.5 \cdot 0 + 0.5 \cdot 0.01 = 0.005$$

So, the security condition does not hold and the program must be rejected.

6. Analysis for Security Properties

The compositionality of the security condition can be fruitfully exploited when proving correctness of various compositional analyses. Instances of such are type-system-based analyses. In this section we investigate the compositionality properties of the security condition and show how an example type system [46], compositionally extended to handle probabilistic choice, can be proved correct with respect to the probabilistic security condition of Section 5.

6.1. A Simple Sequential Language with Probabilistic Choice

Consider a very simple sequential language with probabilistic choice ($[\!]_p$), described by the grammar in Figure 1. The semantics of the language is defined in Figure 2. The definition is an adaptation of

$$
\begin{aligned}
[\![\mathsf{skip}]\!] &= \lambda s.\eta_s \\
[\![Id := Exp]\!] &= \lambda s.\eta_{s'},\ \text{where } s' = [eval(Exp,s)/Id]s \\
[\![C_1; C_2]\!] &= [\![C_2]\!]^* \circ [\![C_1]\!] \\
[\![C_1 \,[\!]_p\, C_2]\!] &= p \cdot [\![C_1]\!] + (1-p) \cdot [\![C_2]\!] \\
[\![\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2]\!] &= \lambda s.\mathbf{if}\ eval(B,s)\ \mathbf{then}\ [\![C_1]\!]\ s\ \mathbf{else}\ [\![C_2]\!]\ s \\
[\![\mathsf{while}\ B\ \mathsf{do}\ C]\!] &= \mathsf{fix}\, f.\lambda s.\mathbf{if}\ eval(B,s)\ \mathbf{then}\ (f^* \circ [\![C]\!])s\ \mathbf{else}\ \eta_s
\end{aligned}
$$

Figure 2. Probabilistic semantics of commands

the probabilistic semantics of Jones and Plotkin [21]. The $eval(Exp, s)$ function evaluates a boolean or arithmetic expression Exp in a state s. A state is a tuple of variable values. Given an input state, executing commands skip and $Id := Exp$ results in the point-mass distribution for the input state (updated in the case of assignment $Id := Exp$).

The semantics of the sequential composition is defined using the extension operator introduced in Section 5.1.

When running $C_1 \,[\!]_p\, C_2$, command C_1 is chosen for execution with probability p and C_2 is chosen with probability $1 - p$. The existence of the least fixpoint fix in the definition of the semantics for while is guaranteed by the inductive partial order construction of Jones and Plotkin [21].

Let us first establish the compositional nature of the probabilistic security condition and then go on with a straightforward proof of correctness of the type system.

6.2. Hook-up Properties

We start with a proposition that will be useful when reasoning about sequential composition of security types.

PROPOSITION 2. *For any* $f \in [\mathbf{St}_\perp \to \mathcal{E}[\mathbf{St}_\perp]]$ *and any* R, $S \in Per(\mathbf{St}_\perp)$,

$$f : R \twoheadrightarrow \mathcal{E}[S] \iff f^* : \mathcal{E}[R] \twoheadrightarrow \mathcal{E}[S]$$

Proof. ($\Rightarrow$): Assume $f : R \twoheadrightarrow \mathcal{E}[S]$. It means $s_1\ R\ s_2 \Longrightarrow f(s_1)\ \mathcal{E}[S]\ f(s_2)$ or, using the law $f = f^* \circ \eta$:

$$s_1\ R\ s_2 \Longrightarrow \forall e_S \in \mathbf{St}_\perp/S.\, (f^*\eta_{s_1})e_S = (f^*\eta_{s_2})e_S.$$

Suppose $\mu\ \mathcal{E}[R]\ \nu$. We need to show $(f^*\mu)\ \mathcal{E}[S]\ (f^*\nu)$ or, unwinding the definition of the per $\mathcal{E}[S]$: $\forall e_S \in \mathbf{St}_\perp/S.\, (f^*\mu)e_S = (f^*\nu)e_S$. For any $e_S \in \mathbf{St}_\perp/S$:

$$
\begin{array}{ll}
& (f^*\mu)e_S \\
& \quad \text{rewrite } \mu(x) = \sum_{e_R \in \mathbf{St}_\perp/R} \sum_{r \in e_R} \mu(r) \cdot \eta_r(x) \\
= & (f^*(\sum_{e_R \in \mathbf{St}_\perp/R} \sum_{r \in e_R} \mu(r) \cdot \eta_r))e_S \\
& \quad \text{distributivity of } \cdot \text{ and } + \text{ over } f^* \\
= & (\sum_{e_R \in \mathbf{St}_\perp/R} \sum_{r \in e_R} \mu(r) \cdot (f^*\eta_r))e_S \\
& \quad \text{distributivity of } \cdot \text{ and } + \\
= & \sum_{e_R \in \mathbf{St}_\perp/R} \sum_{r \in e_R} \mu(r) \cdot ((f^*\eta_r)e_S) \\
& \quad \text{lhs of proposition} \\
= & \sum_{e_R \in \mathbf{St}_\perp/R} \sum_{r \in e_R} \mu(r) \cdot ((f^*\eta_{r_0})e_S) \\
& \quad \text{for some } r_0 \text{ such that } [r_0]_R = e_R \\
= & \sum_{e_R \in \mathbf{St}_\perp/R}((f^*\eta_{r_0})e_S) \cdot \sum_{r \in e_R} \mu(r) \\
= & \sum_{e_R \in \mathbf{St}_\perp/R}((f^*\eta_{r_0})e_S) \cdot \mu(e_R) \\
& \quad \text{use } \forall e_R.\, \mu(e_R) = \nu(e_R) \\
= & \sum_{e_R \in \mathbf{St}_\perp/R}((f^*\eta_{r_0})e_S) \cdot \nu(e_R) \\
= & (f^*\nu)e_S
\end{array}
$$

($\Leftarrow$): Suppose $r_1 \, R \, r_2$. Need to show $f(r_1) \; \mathcal{E}[S] \; f(r_2)$.

$$
\begin{array}{lll}
r_1 \, R \, r_2 & & \\
\quad \Longrightarrow & \eta_{r_1} \; \mathcal{E}[R] \; \eta_{r_2} & \\
\quad \Longrightarrow & f^*(\eta_{r_1}) \; \mathcal{E}[S] \; f^*(\eta_{r_2}) & \text{rhs of proposition} \\
\quad \Longrightarrow & f(r_1) \; \mathcal{E}[S] \; f(r_2) & f = f^* \circ \eta \quad \Box
\end{array}
$$

Corollary. From this proposition it easily follows that the following inference rule is sound:

$$
\frac{[\![C_1]\!] : P \twoheadrightarrow \mathcal{E}[Q] \qquad [\![C_2]\!] : Q \twoheadrightarrow \mathcal{E}[R]}{[\![C_1; C_2]\!] : P \twoheadrightarrow \mathcal{E}[R]}
$$

In the standard security terminology, the *hook-up* property [28] says that if two programs are secure then their composition (sequential or some other kind) is secure as well. We will investigate what kinds of composition are allowed. These properties are important since they are the key to the utility of the specification for the purpose of proving the correctness of syntax-directed program analyses.

Recall the probabilistic powerdomain security condition for the case of two variables h and l and domain $D = \mathbf{St}_\perp$. A command C is secure iff

$$\llbracket C \rrbracket_{\mathcal{E}}(i_h, i_l) \perp = \llbracket C \rrbracket_{\mathcal{E}}(i'_h, i_l) \perp \ \&$$
$$\sum_{o_h \in \mathbf{St}_{high}} \llbracket C \rrbracket_{\mathcal{E}}(i_h, i_l)(o_h, o_l) = \sum_{o_h \in \mathbf{St}_{high}} \llbracket C \rrbracket_{\mathcal{E}}(i'_h, i_l)(o_h, o_l)$$

for any i_l, i_h, i'_h and o_l.

We will need some definitions to study various kinds of composition. A (boolean or arithmetic) expression Exp is *low* iff $\lambda s.eval(Exp, s) : All \times Id \twoheadrightarrow Id$. Otherwise, the expression is *high*.

We cannot plug a secure program into an arbitrary context to obtain a secure program. (Take, for example, any ground context that is insecure.) The security property is preserved by *secure contexts*, i.e. contexts built with secure components. Let $[\bullet]$ be a hole for a command. A context $\mathbb{C}[\bullet_1, \bullet_2]$ is secure iff it has one of the following forms:

$$\begin{aligned} \mathbb{C}[\bullet_1, \bullet_2] ::= \ & \mathsf{skip} \mid h := Exp \mid l := Exp \ (Exp \text{ is low}) \\ & \mid [\bullet_1]; [\bullet_2] \mid [\bullet_1] \ [\!]_p \ [\bullet_2] \\ & \mid \mathsf{if}\ B\ \mathsf{then}\ [\bullet_1]\ \mathsf{else}\ [\bullet_2]\ (B \text{ is low }) \\ & \mid \mathsf{while}\ B\ \mathsf{do}\ [\bullet_1]\ (B \text{ is low }) \end{aligned}$$

Suppose $f : D \to \mathcal{E}[D]$. Define f's i'th *iteration* $f^i : D \to \mathcal{E}[D]$ by:

$$f^0 = \eta_D \qquad\qquad f^{n+1} = f^* \circ f^n$$

THEOREM 5 (Hook-up).

- *If C_1, C_2 are secure and $\mathbb{C}[\bullet_1, \bullet_2]$ is a secure context, then $\mathbb{C}[C_1, C_2]$ is secure.*

- *If $\mathbb{C}[\bullet_1, \bullet_2] = \mathsf{if}\ B\ \mathsf{then}\ [\bullet_1]\ \mathsf{else}\ [\bullet_2]$ (B is high), then $\mathbb{C}[C_1, C_2]$ is secure provided*

$$\llbracket C_1 \rrbracket \ ((All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp]) \ \llbracket C_2 \rrbracket$$

- *If $\mathbb{C}[\bullet_1, \bullet_2] = \mathsf{while}\ B\ \mathsf{do}\ [\bullet_1]$ (B is high), then $\mathbb{C}[C_1, C_2]$ is secure provided*

$$(i) \ \ \forall i, j \in \mathbf{N}.\ \llbracket C_1 \rrbracket^i \ ((All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp]) \ \llbracket C_1 \rrbracket^j$$
$$(ii) \ \ \forall s.\ \llbracket \mathbb{C}[C_1, C_2] \rrbracket \ s \perp = 0$$

Proof. Cases on $\mathbb{C}[C_1, C_2] = C$. In each case we inspect the behaviour of the program fed with two *low-equal* input states s_1 and s_2 (the low components of the states are the same but the high components are possibly different).

skip

Start with low-equal s_1 and s_2. With either state, the computation of skip terminates in one step with the same states. So, the final states are low-equal as well.

$l := Exp$ **(**Exp **is low)**

Since Exp is a low expression, the computation of the command with low-equal initial states s_1 and s_2 terminates with the same states that are still low-equal.

$h := Exp$

Exp can be arbitrary. Computing the command will not change the low component of the states.

$C_1; C_2$

We know that $[\![C_i]\!] : (All \times Id)_\perp \rightarrow\!\!\triangleright \mathcal{E}[(All \times Id)_\perp], i = \{1, 2\}$. Applying the inference rule of the corollary of Proposition 2 yields $[\![C_2]\!]^* \circ [\![C_1]\!] : (All \times Id)_\perp \rightarrow\!\!\triangleright \mathcal{E}[(All \times Id)_\perp]$. Thus, C is secure.

$C_1 \, [\!]_p \, C_2$

Trivial, as computation does not depend on h.

if B **then** C_1 **else** C_2 **(**B **is low)**

Start with low-equal s_1 and s_2. The low expression B has the same value in either state. Thus, one step of computation will lead to a configuration with the same command (C_1 or C_2 in both cases) and low-equal states. C_1 and C_2 are secure and, therefore so is C.

if B **then** C_1 **else** C_2 **(**B **is high)**

Start with low-equal s_1 and s_2. Here the value of B might differ for s_1 and s_2, but, due to the assumption of the theorem, we have $[\![C_i]\!] \, (\mathcal{E}[(All \times Id)_\perp]) \, [\![C_j]\!]$ for $i, j \in \{1, 2\}$. This guarantees the security of C.

while B **do** C_1 **(**B **is low)**

Start with low-equal s_1 and s_2. The low expression B has the same value in either state. Thus, the fixpoints in the semantics of the while under both initial states will be the same.

while B **do** C_1 **(**B **is high)**

Start with low-equal s_1 and s_2. The value of B might differ for s_1 and s_2. However, from the assumption for this case we know that (ii) the while-loop terminates and (i) any number of C_1-iterations will result in the same distribution of the final values of l. □

6.3. Type-system-based Analysis

Let us exemplify how the compositional nature of the security condition can be used for straightforward proofs of correctness of compositional analyses. In this subsection we consider a nontermination-covert-flow-sensitive type system by Volpano and Smith's [46], adapted to our language and extended to handle probabilistic choice. Note that this is an instance of a particular analysis that will reject some secure programs. The compositionality of the security condition can be used for justification of any compositional analysis of arbitrary precision. The point of this subsection is not to develop a powerful analysis, but rather to illustrate how simple the proof technique is using the compositional security condition.

Let us now describe the type system. The basic idea is to give security types (note the difference between these types and semantic security types used in security specification) to expressions (*high* or *low*) and commands (*high cmd* or *low cmd*) and make sure that these types do not mismatch when composing the commands. The main job of the types is to make sure that (i) an expression of type *low* does not contain an occurrence of the high variable h, (ii) a command of type *high cmd* does not have an occurrence of assignment to the low variable l and (iii) while-loops have the type *low cmd*. These properties can be easily proved by induction.

The type system is presented in Figure 3. The rule Assign_{low} prevents direct insecure information flows—for example, the assignment $l := h$ is not typable. The rule If prevents indirect insecure flows—the command if $h = 0$ then $l := 0$ else $l := 1$ cannot be typed since the guard has type *high* and the branches have type *low cmd*. In this particular system, the guard of the while-loop has to be low to prevent nontermination leaks. The while-loop has to have type *low cmd*. This guarantees that a while cannot be placed in a branch of a high conditional, with could also introduce a potential nontermination leak.

Let us now prove the security of the type system. We start off with a simple lemma. The lemma intuitively says that the behaviour of two commands with no loops or assignments to l can only differ in the high output as one varies the high input.

LEMMA 1. $C_1 : high\ cmd, C_2 : high\ cmd \Longrightarrow$
$[\![C_1]\!]\ ((All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp])\ [\![C_2]\!]$

$$
\begin{array}{rl}
[\text{Var}] & h : \mathit{high} \qquad l : \mathit{low} \\[2ex]
[\text{Exp}] & n : \tau \qquad \mathit{Exp} : \mathit{high} \\[2ex]
[\text{Arithm}_{low}] & \dfrac{\mathit{Exp}_1 : \mathit{low} \quad \mathit{Exp}_2 : \mathit{low}}{\mathit{op}(\mathit{Exp}_1, \mathit{Exp}_2) : \mathit{low}} \\[2ex]
[\text{Skip}] & \mathsf{skip} : \tau\ \mathit{cmd} \\[2ex]
[\text{Assign}_{low}] & \dfrac{\mathit{Exp} : \mathit{low}}{l := \mathit{Exp} : \mathit{low}\ \mathit{cmd}} \\[2ex]
[\text{Assign}_{high}] & h := \mathit{Exp} : \tau\ \mathit{cmd} \\[2ex]
[\text{Seq}] & \dfrac{C_1 : \tau\ \mathit{cmd} \quad C_2 : \tau\ \mathit{cmd}}{C_1 ; C_2 : \tau\ \mathit{cmd}} \\[2ex]
[\text{Choice}] & \dfrac{C_1 : \tau\ \mathit{cmd} \quad C_2 : \tau\ \mathit{cmd}}{C_1 \, []_p \, C_2 : \tau\ \mathit{cmd}} \\[2ex]
[\text{If}] & \dfrac{B : \tau \quad C_1 : \tau\ \mathit{cmd} \quad C_2 : \tau\ \mathit{cmd}}{\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 : \tau\ \mathit{cmd}} \\[2ex]
[\text{While}] & \dfrac{B : \mathit{low} \quad C : \tau\ \mathit{cmd}}{\mathsf{while}\ B\ \mathsf{do}\ C : \mathit{low}\ \mathit{cmd}}
\end{array}
$$

Figure 3. Type system for security

Proof. We need to show

$$
\begin{gathered}
[\![C_1]\!]_{\mathcal{E}}(i_h, i_l)\ \bot = [\![C_2]\!]_{\mathcal{E}}(i'_h, i_l)\ \bot\ \ \& \\
\sum_{o_h \in \mathbf{St}_{high}} [\![C_1]\!]_{\mathcal{E}}(i_h, i_l)(o_h, o_l) = \sum_{o_h \in \mathbf{St}_{high}} [\![C_2]\!]_{\mathcal{E}}(i'_h, i_l)(o_h, o_l)
\end{gathered}
$$

for any i_l, i_h, i'_h and o_l. Clearly, $[\![C_1]\!]_{\mathcal{E}}(i_h, i_l)\ \bot\ = 0 = [\![C_2]\!]_{\mathcal{E}}(i'_h, i_l)\ \bot$ since nontermination is only possible in the presence of a while-loop. By property (iii) of the type system, no while-loop can occur in a high command. From the fact that there are no assignments to l ((ii)-property), deduce

$$
\sum_{o_h \in \mathbf{St}_{high}} [\![C_i]\!]_{\mathcal{E}}(i_h, i_l)(o_h, o_l) = \begin{cases} 1, & \text{if } i_l = o_l \\ 0, & \text{otherwise} \end{cases}
$$

for both $i = 1, 2$. This implies the second equality. □

The following theorem is a straightforward utilisation of the hook-up properties shown in Theorem 5.

THEOREM 6 (Security of the Analysis).

$C : \tau\ cmd \implies C\ \textit{is secure.}$

Proof. Induction on C and direct application of the hook-up theorem. Let us consider the cases by which $C : \tau\ cmd$. The cases $\mathsf{skip} : \tau\ cmd$, $l := Exp : low\ cmd\ (Exp : low)$ and $h := Exp : \tau\ cmd$ are immediate since C is a ground secure context. (The case $l := Exp : low\ cmd\ (Exp : low)$ relies on the (i)-property of the type system which guarantees $Exp : low \Longrightarrow Exp$ is low.)

The cases $C_1; C_2 : \tau\ cmd$, $C_1 \mathbin{[]_p} C_2 : \tau\ cmd$, $\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 : low\ cmd\ (B : low)$ and $\mathsf{while}\ B\ \mathsf{do}\ C : low\ cmd\ (B : low)$ provide secure contexts $[\bullet_1]; [\bullet_2]$, $[\bullet_1] \mathbin{[]_p} [\bullet_2]$, $\mathsf{if}\ B\ \mathsf{then}\ [\bullet_1]\ \mathsf{else}\ [\bullet_2]$ and $\mathsf{while}\ B\ \mathsf{do}\ [\bullet]$ (B is low by in the two latter by the (i)-property) respectively. By the induction hypothesis the sub-commands are secure, so the conclusion follows by the hook-up theorem.

The last case is $\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 : high\ cmd$. Both C_1 and C_2 have type $high\ cmd$. To apply the if-on-high case of Theorem 5 note that by Lemma 1 we have $[\![C_1]\!]\ ((All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp])\ [\![C_2]\!]$. $\Box$

7. Conclusions

We have developed an extensional semantics-based specification of secure information flow in sequential programs, by embracing and extending earlier work on the use of partial equivalence relations to model binding times [19]. We have shown how this idea can be extended to handle nondeterminism and also probabilistic information flow.

A final remark on some recent related work: Abadi, Banerjee, Heintze and Riecke [1] show that a single calculus (DCC), based on Moggi's computational lambda calculus, can capture a number of specific static analyses for security, binding-time analysis, program slicing and call-tracking. Although their calculus does not handle nondeterministic language features, it is notable that the semantic model given to DCC is per-based, and the logical presentations of the abstract interpretation for per-based BTA [19, 20, 14] readily fit this framework (although this specific analysis is not one of those considered for DCC). They also show that what we have called "termination insensitive" analyses (Section 2.3) can be modelled by extending the semantic relations to relate bottom (nontermination) to every other domain point (without insisting on transitivity). It is encouraging to note that—at least in the deterministic setting—the per-based approach can be weakened to a termination-insensitive condition without significant technical difficulties. We do not, however, see any obvious way to make the probabilistic security condition insensitive to termination in a similar manner.

We conclude by considering a few possible extensions and limitations:

Multi-level security There is no problem with handling lattices of security levels rather than the simple *high-low* distinction. But one cannot expect to assign any intrinsic semantic meaning to such lattices of security levels, since they represent a "social phenomenon" which is external to the programming language semantics. In the presence of multiple security levels one must simply formulate conditions for security by considering information flows between levels in a pairwise fashion (although of course a specific static analysis is able to do something much more efficient).

Downgrading and Trusting There are operations which are natural to consider but which cannot be modelled in an obvious way in an extensional framework. One such operation is the downgrading of information from high to low without losing information—for example representing the secure encryption of high level information. This seems impossible since an encryption operation does not lose information about a value and yet should have type $high \to low$—but the only functions of type $high \to low$ are the constant functions. An analogous problem arises with Ørbæk and Palsberg's `trust` primitive if we try to use pers to model their *integrity analysis* [37].

Operational Semantics We are not particularly married to the denotational perspective on programming language semantics. We have reported an examination of operational formulations of pers on a multi-threaded language [42], based on partial bisimulations.

Constructing Program Analyses Although the model seems useful to compare other formalisations, further work is needed to show that it can assist in the systematic design of program analyses.

Concurrency Handling nondeterminism can be viewed as the main stepping stone to formulating a language-based security condition for concurrent languages. We have made an investigation of per-based security conditions for a multi-threaded language [42], using an operational rather than denotational semantic model of security.

Acknowledgements

We would like to thank the anonymous referees for valuable feedback.

References

1. Abadi, M., A. Banerjee, N. Heintze, and J. Riecke: 1999, 'A Core calculus of Dependency'. In: *POPL '99, Proceedings of the 26th Annual ACM Symposium on Principles of Programming Languages (January 1999).*
2. Abadi, M. and G. Plotkin: 1990, 'A Per Model of Polymorphism and Recursive Types'. In: *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*. pp. 355–365.
3. Andrews, G. R. and R. P. Reitman: 1980, 'An axiomatic approach to information flow in programs'. *ACM TOPLAS* **2**(1), 56–75.
4. Banâtre, J.-P., C. Bryce, and D. Le Métayer: 1994, 'Compile-time detection of information flow in sequential programs'. In: D. Gollmann (ed.): *Computer Security—ESORICS '94, 3rd European Symposium on Research in Computer Security*, Vol. 875 of *Lecture Notes in Computer Science*. pp. 55–73.
5. Bell, D. and L. LaPadula: 1976, 'Secure Computer Systems: Unified Exposition and Multics Interpretation'. MTR-2997, Rev. 1, The MITRE Corporation, Bedford, Mass.
6. Cohen, E. S.: 1977, 'Information Transmission in Computational Systems'. *ACM SIGOPS Operating Systems Review* **11**(5), 133–139.
7. Cohen, E. S.: 1978, 'Information Transmission in Sequential Programs'. In: R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton (eds.): *Foundations of Secure Computation*. Academic Press, pp. 297–335.
8. Das, M., T. Reps, and P. V. Hentenryck: 1995, 'Semantic Foundations of Binding-Time Analysis for Imperative Programs'. In: *Partial Evaluation and Semantics-Based Program Manipulation*. La Jolla, California, pp. 100–110.
9. Denning, D. E.: 1976, 'A Lattice Model of Secure Information Flow'. *Communications of the ACM* **19**(5), 236–243.
10. Denning, D. E. and P. J. Denning: 1977, 'Certification of Programs for Secure Information Flow'. *Communications of the ACM* **20**(7), 504–513.
11. Focardi, R. and R. Gorrieri: 1994, 'A Classification of Security Properties for Process Algebra'. *J. Computer Security* **3**(1), 5–33.
12. Goguen, J. and J. Meseguer: 1982, 'Security Policies and Security Models'. In: *Proceedings of the IEEE Symposium on Security and Privacy.*
13. Gray III, J.: 1990, 'Probabilistic Interference'. In: *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, California, pp. 170–179.
14. Hankin, C. L. and D. Le Métayer: 1994, 'A Type-based Framework for Program Analysis'. In: *Proceedings of the First Static Analysis Symposium*, Vol. 864 of *LNCS*.
15. Heintze, N. and J. G. Riecke: 1998, 'The SLam Calculus: Programming with Secrecy and Integrity'. In: *Conference Record of POPL'98: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. San Diego, California, pp. 365–377.
16. Henglein, F. and D. Sands: 1995, 'A Semantic Model of Binding Times for Safe Partial Evaluation'. In: M. Hermenegildo and S. D. Swierstra (eds.): *Proc.*

Programming Languages: Implementations, Logics and Programs (PLILP), Utrecht, The Netherlands, Vol. 982 of *Lecture Notes in Computer Science*. pp. 299–320.

17. Hunt, L. S.: 1991, 'Abstract Interpretation of Functional Languages: From Theory to Practice'. Ph.D. thesis, Department of Computing, Imperial College of Science, Technology and Medicine.
18. Hunt, S.: 1990, 'PERs generalise projections for strictness analysis'. In: *Draft Proceedings of the Third Glasgow Functional Programming Workshop*. Ullapool.
19. Hunt, S. and D. Sands: 1991, 'Binding Time Analysis: A New PERspective'. In: *Proceedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'91)*. pp. 154–164. ACM SIGPLAN Notices 26(9).
20. Jensen, T. P.: 1992, 'Abstract Interpretation in Logical Form'. Ph.D. thesis, Imperial College, University of London. Available as DIKU Report 93/11 from DIKU, University of Copenhagen.
21. Jones, C. and G. D. Plotkin: 1989, 'A Probabilistic Powerdomain of Evaluations'. In: *Proceedings, Fourth Annual Symposium on Logic in Computer Science*. Asilomar Conference Center, Pacific Grove, California, pp. 186–195.
22. Kozen, D.: 1981, 'Semantics of Probabilistic Programs'. *Journal of Computer and System Sciences* **22**, 328–350.
23. Kozen, D.: 1985, 'A Probabilistic PDL'. *Journal of Computer and System Sciences* **30**, 162–178.
24. Launchbury, J.: 1989, 'Projection Factorisations in Partial Evaluation'. Ph.D. thesis, Department of Computing, University of Glasgow.
25. Leino, K. R. M. and R. Joshi: 1998, 'A Semantic Approach to Secure Information Flow'. In: *MPC'98*.
26. Leino, K. R. M. and R. Joshi: 2000, 'A Semantic Approach to Secure Information Flow'. *Science of Computer Programming* **37**(1).
27. Manes, E.: 1976, *Graduate Texts in Mathematics*, Vol. 26. Springer-Verlag.
28. McCullough, D.: 1987, 'Specifications for Multi-level Security and Hook-Up Property'. In: *Proceedings of the IEEE Symposium on Security and Privacy*. pp. 161–166.
29. McLean, J.: 1990a, 'Security Models and Information Flow'. In: *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, California, pp. 180–187.
30. McLean, J.: 1990b, 'The Specification and Modeling of Computer Security'. *Computer* **23**(1), 9–16.
31. McLean, J.: 1994, 'Security Models'. In: J. Marciniak (ed.): *Encyclopedia of Software Engineering*. Wiley & Sons.
32. Mizuno, M. and D. Schmidt: 1992, 'A Security Flow Control Algorithm and Its Denotational Semantics Correctness Proof'. *Formal Aspects of Computing* **4**(6A), 727–754.
33. Moskowitz, I. S. and O. L. Costich: 1992, 'A Classical Automata Approach to Noninterference Type Problems'. In: *The Computer Security Foundations Workshop V proceedings: June 16–18, 1992, the Franconia Inn, Franconia, New Hampshire*. pp. 2–8.
34. Nielson, F.: 1989, 'Two-Level Semantics and Abstract Interpretation'. *Theoretical Computer Science—Fundamental Studies* **69**, 117–242.

35. Ørbæk, P.: 1995, 'Can you Trust your Data?'. In: P. D. Mosses, M. I. Schwartzbach, and M. Nielsen (eds.): *Proceedings of the TAPSOFT/FASE'95 Conference*. Aarhus, Denmark, pp. 575–590.
36. Ørbæk, P.: 1997, 'Trust and Dependence Analysis'. Ph.D. thesis, Dept. of Computer Science, Univ. of Aarhus. BRICS report DS-97-2.
37. Ørbæk, P. and J. Palsberg: 1997, 'Trust in the λ-calculus'. *Journal of Functional Programming* **7**(4).
38. Plotkin, G.: 1981, 'Post-graduate Lecture Notes in Advanced Domain Theory (incorporating the "Pisa Notes")'. Dept. of Computer Science, Univ. of Edinburgh.
39. Plotkin, G. D.: 1976, 'A Powerdomain Construction'. *SIAM Journal on Computing* **5**(3), 452–487.
40. Reynolds, J. C.: 1983, 'Types, Abstraction and Parametric Polymorphism'. In: R. E. A. Mason (ed.): *Proceedings 9th IFIP World Computer Congress, Information Processing '83, Paris, France, 19–23 Sept 1983.* Amsterdam: North-Holland, pp. 513–523.
41. Sabelfeld, A. and D. Sands: 1999, 'A Per Model of Secure Information Flow in Sequential Programs'. In: *Proceedings of the 8th European Symposium on Programming, ESOP'99*. Amsterdam, pp. 40–58.
42. Sabelfeld, A. and D. Sands: 2000, 'Probabilistic Noninterference for Multi-threaded Programs'. In: *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. Cambridge, England.
43. Smith, G. and D. Volpano: 1998, 'Secure Information Flow in a Multi-threaded Imperative Language'. In: *Conference Record of POPL '98: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. pp. 355–364.
44. Smyth, M. B.: 1978, 'Powerdomains'. *Journal of Computer and Systems Sciences* **16**(1), 23–36.
45. Thiemann, P. and H. Klaeren: 1997, 'Binding-Time Analysis by Security Analysis'. Universitt Tübingen.
46. Volpano, D. and G. Smith: 1997, 'Eliminating Covert Flows with Minimum Typings'. *Proc. 10th IEEE Computer Security Foundations Workshop* pp. 156–168.
47. Volpano, D. and G. Smith: 1999, 'Probabilistic Noninterference in a Concurrent Language'. *Journal of Computer Security* **7**(2,3), 231–253.
48. Volpano, D., G. Smith, and C. Irvine: 1996, 'A Sound Type System for Secure Flow Analysis'. *J. Computer Security* **4(3)**, 1–21.
49. Wadler, P.: 1989, 'Theorems for Free'. In: *Functional Programming Languages and Computer Architecture*. pp. 347–359.
50. Wadler, P. and R. J. M. Hughes: 1987, 'Projections for Strictness Analysis'. In: *1987 Conference on Functional Programming and Computer Architecture*. Portland, Oregon, pp. 385–407.