A Per Model of Secure Information Flow in Sequential Programs

Andrei Sabelfeld and David Sands

Department of Computer Science
Chalmers University of Technology and the University of Göteborg
412 96 Göteborg, Sweden
{andrei,dave}@cs.chalmers.se

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 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 extend the approach to handle nondeterminism by using powerdomain semantics and show how *probabilistic security properties* can be formalised by using probabilistic powerdomain semantics.

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 emailed 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 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

S.D. Swierstra (Ed.): ESOP/ETAPS'99, LNCS 1576, pp. 40–58, 1999.

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 [HS91], 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 be extended to handle nondeterminism, and illustrate how the various choices of powerdomain semantics affects 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 [Den76,DD77] in an extension of Bell and LaPadula's early work [BL76]. 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 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 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 such 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* [VSI96] 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 e.g., [Ørb95] or denotational semantics e.g., [MS92], or on "axiomatic" approaches which define security in terms of a program logic [AR80] 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

[1] A Partial Equivalence relation is symmetric and transitive but not necessarily reflexive

mathematical justification. McLean points out [McL90] 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 in [GM82].

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 [McL94] for a tutorial overview. Many approaches have been phrased in terms of abstract, and sometimes rather ad hoc models of computation. Only more recently have attempts been made to rephrase and compare various security conditions in terms of well-known semantic models, e.g. the use of labelled transition systems and bisimulation semantics in [FG94]. In this paper we consider the problem of information-flow properties of sequential systems, and use the framework of *denotational semantics* as our formal model of computation. Along the way we consider some relations to specific static analyses, such as the ***S**ecurity **Lam**bda Calculus* [HR98] and an alternative semantic condition for secure information flow proposed by Leino and Joshi [LJ98].

1.5 Overview

The rest of the paper is organised as follows. **Section 2** shows how the per-based condition for soundness of binding times 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 [JP89]. **Section 6** 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 Hunts use of pers to model and construct abstract interpretations for strictness properties in higher-order functional programs [Hun90,Hun91], and in particular its use to model dependencies in *binding-time* analysis [HS91]. Related ideas already occur in the denotational formulation of live-variable analysis [Nie90].

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 [GM82]. Many problems in security come down to forms of dependency analysis. For example, in the case of *confidentiality*, the 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.

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 will be a low value, and, in the case that we have a pair of numbers, the result

[2] Unfortunately, from the perspective of a partial evaluator, BTA is not *purely* a matter of dependencies; in [HS95] it was shown that the pure dependency models of [Lau89] and [HS91] are not adequate to ensure the safety of partial evaluation.

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 [Den76].

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 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 [VSI96,SV98], and also for the extensional correctness proofs in core of the *Slam-calculus* [HR98].

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 $x, x' \in D$:

$$x\ All_D\ x' \tag{3}$$

$$x\ Id_D\ x' \iff x = x'. \tag{4}$$

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

$$\mathsf{snd} : All_D \times Id_E \multimap Id_E,$$

and (2) is given by

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

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" given in [HS91]. 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 folk-law (See e.g. [TK97] for a comparison of a particular security type system and a particular binding-time analysis, and [DRH95] which shows how the incorporation of indirect information flows from Dennings security analysis can improve binding time analyses).

It is worth highlighting a few of the pertinent ideas from [HS91]. 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 of [HS91] which rather straightforwardly extends Launchbury's projection-based binding-time analysis [Lau89] to higher types. To some extent [HS91] anticipates the treatment of partially-secure data types in the SLam calculus [HR98], 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* × *high*)*list*. Constructing equivalence relations which represent such properties is straightforward – see [HS91] for examples (which are adapted directly from Launchbury's work), and [Hun91] 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, [AP90]). As far as we are aware, the first use of pers in static program analysis is that presented in [Hun90].

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 $(D \multimap E) \in Per(D \to E)$ defined by:

$$\begin{array}{c} f\ (P \multimap Q)\ g \\ \iff \\ \forall x, x' \in D.\, x\ P\ x' \Longrightarrow (f\ x)\ Q\ (g\ x'). \end{array}$$

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

$$\begin{array}{rl} f : P \multimap Q & \iff f\ (P \multimap Q)\ f \\ & \iff \forall x, x' \in D.\, x\ P\ x' \Longrightarrow (f\ x)\ Q\ (f\ x'). \end{array}$$

Note that even if P and Q are both total (i.e., equivalence relations), $P \multimap Q$ may be partial. A simple example is $All \multimap Id$. If $f : All \multimap Id$ 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 address. The formulation of security properties given above is sensitive to termination. Consider, for example, the following function $\mathsf{f} : \mathbf{N}_\perp \to \mathbf{N}_\perp$

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

Clearly, if the argument is high then the result must be high. Now consider the security properties of the function $\mathsf{g} \circ \mathsf{f}$ where g the constant function $\mathsf{g} = \lambda x.2$. 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 $|All_{\mathbf{N}_\perp} \multimap Id_{\mathbf{N}_\perp}|$ since $\perp\ All_{\mathbf{N}_\perp}\ 1$ but $\mathsf{g}(\perp) \neq \mathsf{g}(1)$. Hence the function $\mathsf{g} \circ \mathsf{f}$ 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 [SV98] 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 [HR98] 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 $\mathsf{g} \circ \mathsf{f}$ could be considered to have a security type corresponding to "*high* $\rightarrow$ *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 [Rey83][3]. From the type of a polymorphic function one can predict certain properties about its behaviour – the so-called "free theorems" of the type [Wad89]. However, in a strict language one must add an additional condition in order that the theorems hold: the functions must be *bottom-reflecting* ($f(a) = \bot \implies a = \bot$). The same side condition can be added to make the e.g. the type system of the Slam-calculus termination-sensitive.

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 $D_\bot$ without explicit injection of elements from D into elements in $D_\bot$. We will write $R_\bot$ to denote the relation in $Per(D_\bot)$ which naturally extends R by $\bot \; R \; \bot$.

Now we can be more precise about the properties of g under a strict (call-by-value) interpretation: $\mathsf{g} : (All_{\mathbf{N}})_\bot \multimap Id_{\mathbf{N}_\bot}$, which expresses that g is a constant function, modulo strictness. More informatively we can say that that $\mathsf{g} : (All_{\mathbf{N}}) \multimap Id_{\mathbf{N}}$ 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 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 [HS91] this has already been undertaken by e.g. Jensen [Jen92] and Hankin and Le Métayer [HL94].

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 $\mathbf{St}$ of states (which might be viewed as finite maps from variables to discrete values, or simply just a tuple of values).

[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 [AP90]

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 $D_\perp$ and $E_\perp$. Note that we could equally well take the set of all (trivially continuous) functions in $\mathbf{St} \to \mathbf{St}_\perp$, which is isomorphic.

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:

$$C \text{ is secure} \iff [\![C]\!] : (All \times Id)_\perp \twoheadrightarrow (All \times Id)_\perp \tag{5}$$

Which is equivalent to saying that $[\![C]\!] : (All \times Id) \twoheadrightarrow (All \times Id)_\perp$ since we only consider strict functions. Note that this does not imply that $[\![C]\!]$ terminates, but what it does imply is 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 $\mathbf{St}_\perp$ and the subset order of the powerset. This is resolved by considering a suitable *powerdomain* structure [Plo76,Smy78]. The powerdomains are built from a domain D by starting with the *finitely generated* (f.g.) subsets of $D_\perp$ (those non-empty subsets which are either finite, or contain $\perp$), and a preorder on these sets. Quotienting the f.g. sets using the associated equivalence relation yields the corresponding domain. We give each construction in turn, and give an idea about the corresponding *discrete powerdomain* $\mathcal{P}[\mathbf{St}_\perp]$.

- *Lower (Hoare) powerdomain* Let $u \preceq_{\mathsf{L}} v$ iff $\forall x \in u.\, \exists y \in v.\, x \sqsubseteq y$. 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.
- *Upper (Smyth) powerdomain* The upper ordering on f.g. sets u, v, is given by

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

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.

– *Convex (Plotkin) powerdomain* Let $u \preceq_{\mathsf{C}} v$ iff $u \preceq_{\mathsf{U}} v$ and $u \preceq_{\mathsf{L}} v$. 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:

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

A few basic properties and definitions on powerdomains will be needed. For each powerdomain constructor $\mathcal{P}[-]$ define the order-preserving "unit" map $\eta_D : D_\perp \to \mathcal{P}[D_\perp]$ which takes each element $a \in D$ into (the powerdomain equivalence class of) the singleton set $\{a\}$. For each function $f \in [D_\perp \to \mathcal{P}[E_\perp]]$ there exits a unique extension of f, denoted f^* where $f^* \in [\mathcal{P}[D_\perp] \to \mathcal{P}[E_\perp]]$ which is the unique mapping such that

$$f = f^* \circ \eta.$$

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

Give 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*

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

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

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

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 behaviours 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 [LJ98]. Following their approach, assume for simplicity we have programs with just two variables: h and l of high and low secrecy respectively. Assume that the state is simple a pair, where h refers to the first projection and l is the second projection.

In [LJ98] the security condition for a program C is defined 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 we run into some potential problems since it is necessary in [LJ98] that HH always terminates, but nevertheless exhibits unbounded nondeterminism. Although this appears to pose no problems in [LJ98] (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.

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 [WH87]. 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 [Lau89] 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 powerdomain 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 [Hun90] 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 $f : R_\alpha \twoheadrightarrow R_\beta$ 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*

$$[\![HH; C; HH]\!] = [\![C; HH]\!] \quad \textit{iff} \quad C : \mathit{high} \times \mathit{low} \to \mathit{high} \times \mathit{low}.$$

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

$$s_1 \; ker(C) \; s_2 \iff [\![C]\!]s_1 = [\![C]\!]s_2.$$

Define the extension of $ker(C)$ by

$$A \; ker^*(C) \; B \iff [\![C]\!]^*A = [\![C]\!]^*B.$$

Recall the per interpretation of the type signature of C.

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

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

Let us first prove the latter fact by proving a more general statement similar to Proposition 3.1.5 from [Hun91] (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*

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

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

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

It remains to establish the first fact.

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

The proofs are given in the full version of the paper [SS99]. Thus, the equational and per security conditions in this simple case are equivalent.

In a more recent extension of the paper, [LJ99], 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 independent, 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: $[\![C]\!] : R_g \multimap \mathcal{P}_{\mathsf{C}}[(All \times Id)_\perp]$ 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 indiction of its value by observing only the possible final values of l, e.g., $2, 17, 2, 45, 2, 2, 33, 2, 97, 2, 8, 57, 2, 2, 66, \ldots$ from which we might reasonably conclude that the value of h was 2.

To counter this problem we consider *probabilistic powerdomains* [JP89] 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.

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 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 from [JP89] 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 domains, and hence a simplified notion of probabilistic powerdomain in terms of distributions.

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, to be 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 from [JP89], one can easily see that $\mathcal{E}[S]$ is an inductively complete partial order with directed lubs defined pointwise, and with a least element $\omega = \eta_S(\perp)$, where η_S is the *point-mass distribution* defined for an $x \in S$ by

$$\eta_S(x)d = \begin{cases} 1, \text{ if } d = x, \\ 0, \text{ otherwise.} \end{cases}$$

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

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

The structure $(\mathcal{E}[D], \eta_D(x), *)$ is a *Kleisli triple*, and thus we have a canonical way of 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 $g^* \circ f^*$.

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.

Define the per relation $\mathcal{E}[R]$ on $\mathcal{E}[D]$ for $\mu, \nu \in \mathcal{E}[D]$ by

$$\mu \; \mathcal{E}[R] \; \nu \text{ iff } \forall d \in |R|. \sum_{e \in [d]_R} \mu e = \sum_{e \in [d]_R} \nu e,$$

where $[d]_R$ stands for the $R-$equivalence class which contains d. 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 $\mathcal{E}[(All \times Id)_\perp]$. Two distributions μ and ν in $(All \times Id)_\perp \to [0, 1]$ are equal if 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 $[\![C]\!]_{\mathsf{C}} : R \twoheadrightarrow \mathcal{P}_{\mathsf{C}}[S]$.

In the case that the state is modelled by a pair representing a high variable and a low variable respectively, it is easy to see that a command C is secure ($[\![C]\!]_{\mathcal{E}} : (All \times Id)_\perp \twoheadrightarrow \mathcal{E}[(All \times Id)_\perp]$) if and only if

$$\begin{gathered} [\![C]\!]_{\mathcal{E}}(i_h, i_l) \perp = [\![C]\!]_{\mathcal{E}}(i'_h, i_l) \perp \quad \text{and} \\ \textstyle\sum_{h \in \mathbf{St}_{high}} [\![C]\!]_{\mathcal{E}}(i_h, i_l)(h, o_l) = \sum_{h \in \mathbf{St}_{high}} [\![C]\!]_{\mathcal{E}}(i'_h, i_l)(h, o_l) \end{gathered}$$

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 (the sums provide "forgetting" the highs) does not change.

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:

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

The semantics of the uniformly distributed nondeterministic choice $C_1 \,[\!]\, C_1$ is defined by $[\![C_1 \,[\!]\, C_2]\!]\ i\ o = 0.5[\![C_1]\!]\ i\ o + 0.5[\![C_2]\!]\ i\ o$. Consult [JP89] for an account of how to define the semantics of other language constructs.

Example. Recall the program

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

Now we investigate the security condition by varying the 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

$$\sum_{h \in [0,\ldots,100]} [\![C]\!]\mathcal{E}(0,0)(h,0) = 0.5 * 1 + 0.5 * 0.01 = 0.505,$$

whereas the right-hand side is

$$\sum_{h \in [0,\ldots,100]} [\![C]\!]\mathcal{E}(1,0)(h,0) = 0.5 * 0 + 0.5 * 0.01 = 0.005.$$

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

Volpano and Smith recently devised a probabilistic security type-system [VS98] with a soundness proof based on a probabilistic operational semantics. Although the security condition that they use in their correctness argument is not directly comparable – due to the fact that they consider parallel deterministic threads and a non-compositional semantics – we can easily turn their examples into nondeterministic sequential programs with the same probabilistic behaviours. In the extended version of this paper [SS99] we show how their examples can all be verified using our security condition.

6 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 in [HS91]. We have shown how this idea can be extended to handle nondeterminism and also probabilistic information flow.

We recently became aware of work by Abadi, Banerjee, Heintze and Riecke [ABHR99] which shows 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 from [HS91,Jen92,HL94] readily fit this framework (although this specific analysis is not one of those considered in [ABHR99]). They also show that what we have called "termination insensitive" analyses 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 – this appears to create no 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* [ØP97].

Concurrency Handling nondeterminism can be viewed as the main stepping stone to formulating a language-based security condition for concurrent languages, but this remains a topic for further work.

References

ABHR99. M. Abadi, A. Banerjee, N. Heintze, and J. Riecke. A core calculus of dependency. In *POPL '99, Proceedings of the 26th Annual ACM Symposium on Principles of Programming Languages (January 1999)*, 1999.

AP90. M. Abadi and G. Plotkin. A per model of polymorphism and recursive types. In *Logic in Computer Science*. IEEE, 1990.

AR80. G. R. Andrews and R. P. Reitman. An axiomatic approach to information flow in programs. *ACM TOPLAS*, 2(1):56–75, January 1980.

BL76. D.E. Bell and L.J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. MTR-2997, Rev. 1, The MITRE Corporation, Bedford, Mass., 1976.

DD77. Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, July 1977.

Den76. Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

DRH95. M. Das, T. Reps, and P. Van Hentenryck. Semantic foundations of binding-time analysis for imperative programs. In *Partial Evaluation and Semantics-Based Program Manipulation*, pages 100–110, La Jolla, California, June 1995. ACM.

FG94. R. Focardi and R. Gorrieri. A classification of security properties for process algebra. *J. Computer Security*, 3(1):5–33, 1994.

GM82. Joseph Goguen and José Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, April 1982.

HL94. C. L. Hankin and D. Le Métayer. A type-based framework for program analysis. In *Proceedings of the First Static Analysis Symposium*, volume 864 of *LNCS*. Springer-Verlag, 1994.

HR98. Nevin Heintze and Jon G. Riecke. 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*, pages 365–377, San Diego, California, January 19–21, 1998.

HS91. S. Hunt and D. Sands. Binding Time Analysis: A New PERspective. In *Proceedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'91)*, pages 154–164, September 1991. ACM SIGPLAN Notices 26(9).

HS95. F. Henglein and D. Sands. A semantic model of binding times for safe partial evaluation. In Manuel Hermenegildo and S.D̃oaitse Swierstra, editors, *Proc. Programming Languages: Implementations, Logics and Programs (PLILP), Utrecht, The Netherlands*, volume 982 of *Lecture Notes in Computer Science*, pages 299–320. Springer-Verlag, September 1995.

Hun90. S. Hunt. PERs generalise projections for strictness analysis. In *Draft Proceedings of the Third Glasgow Functional Programming Workshop*, Ullapool, 1990.

Hun91. L. S. Hunt. *Abstract Interpretation of Functional Languages: From Theory to Practice*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, 1991.

Jen92. T. P. Jensen. *Abstract Interpretation in Logical Form*. PhD thesis, Imperial College, University of London, November 1992. Available as DIKU Report 93/11 from DIKU, University of Copenhagen.

JP89. C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 186–195, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.

Lau89. J. Launchbury. *Projection Factorisations in Partial Evaluation*. PhD thesis, Department of Computing, University of Glasgow, 1989.

LJ98. K. R. M. Leino and Rajeev Joshi. A semantic approach to secure information flow. In *MPC'98*, Springer Verlag LNCS, 1998.

LJ99. K. R. M. Leino and Rajeev Joshi. A semantic approach to secure information flow. *Science of Computer Programming*, 1999. To appear.

McL90. John McLean. The specification and modeling of computer security. *Computer*, 23(1):9–16, January 1990.

McL94. J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.

MS92. M. Mizuno and D. Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 4(6A):727–754, 1992.

Nie90. F. Nielson. Two-level semantics and abstract interpretation — fundamental studies. *Theoretical Computer Science*, (69):117–242, 90.

ØP97. Peter Ørbæk and Jens Palsberg. Trust in the λ-calculus. *Journal of Functional Programming*, 7(4), 1997.

Ørb95. Peter Ørbæk. Can you Trust your Data? In M. I. Schwartzbach P. D. Mosses and M. Nielsen, editors, *Proceedings of the TAPSOFT/FASE'95 Conference*, LNCS 915, pages 575–590, Aarhus, Denmark, May 1995. Springer-Verlag.

Plo76. G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

Rey83. John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Proceedings 9th IFIP World Computer Congress, Information Processing '83, Paris, France, 19–23 Sept 1983*, pages 513–523. North-Holland, Amsterdam, 1983.

Smy78. Michael B. Smyth. Powerdomains. *Journal of Computer and Systems Sciences*, 16(1):23–36, February 1978.

SS99. Andrei Sabelfeld and David Sands. A per model of secure information flow in sequential programs. Technical report, Department of Computer Science, Chalmers University of Technology, 1999. http://www.cs.chalmers.se/~csreport/.

SV98. Geoffrey Smith and Dennis Volpano. 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*, pages 355–364, 1998.

TK97. P. Thiemann and H. Klaeren. Binding-time analysis by security analysis. Universitt Tübingen, November 1997.

VS98. Dennis Volpano and Geoffrey Smith. Probabilistic noninterference in a concurrent language. In *11th IEEE Computer Security Foundations Workshop*, pages 34–43, 1998.

VSI96. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *J. Computer Security*, 4(3):1–21, 1996.

Wad89. Philip Wadler. Theorems for free. In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM, 1989.

WH87. P. Wadler and R. J. M. Hughes. Projections for strictness analysis. In *1987 Conference on Functional Programming and Computer Architecture*, pages 385–407, Portland, Oregon, September 1987.