

# Secure information flow and program logics

Lennart Beringer, Martin Hofmann  
Institut für Informatik, Universität München  
Oettingenstrasse 67, 80538 München, Germany  
{beringer|mhofmann}@tcs.ifi.lmu.de

## Abstract

We present interpretations of type systems for secure information flow in Hoare logic, complementing previous encodings in binary (e.g. relational) program logics. Treating base-line non-interference, multi-level security and flow sensitivity for a while language, we show how typing derivations may be used to automatically generate proofs in the program logic that certify the absence of illicit flows. In addition, we present proof rules for base-line non-interference for object-manipulating instructions. As a consequence, standard verification technology may be used for verifying that a concrete program satisfies the non-interference property. Our development is based on a formalisation of the encodings in Isabelle/HOL.

## 1 Introduction

One of the main impediments for the establishment of collaborative large-scale computing networks is the reluctance to open devices for code originating from unknown or untrusted sources. Topics of concern include resource limitations, exploitation of the computational device as a launch pad for attacks on external networks, and the integrity, privacy, availability or secrecy of data and services at the host system. An important class of policies belonging to the latter category concerns *secure information flow*. Policies of this class assert that the system does not leak properties of data classified as secret to external parties. Recent research has aimed to complement formulations of security phrased in terms of automata [16] by verification technology for programming languages [33]. In this paper, we consider a notion of security known as termination-insensitive non-interference. For a simple imperative language of commands and while-loops [38], this notion may be phrased as follows, where  $\sigma \xrightarrow{C} \tau$  denotes the (big-step) operational semantics over states  $\sigma, \tau \in \Sigma$  and the set  $\mathcal{X}$  of program variables is disjointly partitioned into high security vari-

ables  $\mathcal{X}_{high}$  and low security variables  $\mathcal{X}_{low}$ .

**Definition 1.** Two states  $\sigma, \tau \in \Sigma$  are indistinguishable (w.r.t. low variables), written  $\sigma \sim \tau$ , if  $\sigma(x) = \tau(x)$  for all  $x \in \mathcal{X}_{low}$ . Program  $C$  is secure if whenever  $\sigma \sim \sigma'$  and  $\sigma \xrightarrow{C} \tau$  and  $\sigma' \xrightarrow{C} \tau'$  then  $\tau \sim \tau'$ .

Several approaches have been developed for verifying adherence to non-interference policies, including (conservative) static analyses phrased as type systems [37, 32, 29] or abstract interpretations [15, 20], flow logics [13], special-purpose program logics [3, 2, 6], and encodings in relational or dynamic program logics [9, 18]. *Self-composition* [23, 1, 7, 36] is a recent approach that avoids the consideration of two executions of  $C$  for  $\sim$ -related initial states. Instead, one verifies the equivalence between two suitably modified versions  $f_1(C)$  and  $f_2(C)$  of  $C$  [23] or the verification of a program  $f(C)$  that originates from  $C$  by composing  $C$  with a copy of  $C$  where all variable names have been renamed [1, 7, 36]. The verification task for the resulting program(s) can then be handled by program logics [23, 1] or model checkers [36]. In particular, [36] shows how to exploit the results of static analyses to obtain a variant of the self-composed program that can be handled more efficiently by standard model checkers.

In this paper, we present an approach that relates type-based approaches to ordinary (non-relational) program logics. We exhibit a class of assertions correspond to non-interference for a given program, and show how judgements involving such assertions may be automatically constructed from appropriate type systems. Thus, the analysis task (type inference) is separated from the verification task (proof checking), which makes our approach suitable for deployment in certified code infrastructures such as PCC [30].

The observation that allows us to encode noninterference in a program logic is the same as the idea behind self composition, namely the fact that two executions of  $C$  do not interfere. Roughly speaking, we identify a scheme such that the non-interference property can be factored into two parts, with auxiliary binary formulae  $\phi$  at the glue point.

For  $\sigma \xrightarrow{C} \tau$ ,  $\phi(\tau, \rho)$  is an invariant that relates the post-state  $\tau$  of  $C$  to any state  $\rho$  that is indistinguishable from the pre-state  $\sigma$ , and which ensures that pre-state  $\sigma$  will evolve to a post-state that is similar to  $\tau$ . That is

- for all  $\rho$ ,  $\sigma \sim \rho$  implies  $\phi(\tau, \rho)$ , and
- for all  $\rho$ ,  $\phi(\rho, \sigma)$  implies  $\rho \sim \tau$ .

The first condition concerns the first parameter of  $\phi$  which loosely corresponds to the execution of the first half of the self-composed program. The second condition concerns the second parameter and may thus be compared to the execution of the second half of the self-composed program.

The proof that the formulation involving formulae  $\phi$  is sound replaces the syntactic operation involved in self-composition by two invocations of the soundness property of the program logic. The construction of appropriate  $\phi$  is achieved automatically from derivations in type systems such as the one by Volpano et al. [37]. While the characterisation is also complete (we exhibit a formula  $\phi$  which holds whenever  $C$  is secure), our overall verification framework inherits the incompleteness of the type systems.

The paper is structured as follows. We start by recollecting the operational semantics of **IMP** and presenting a sound and (relative) complete program logic for **IMP** in Section 2. Limiting our attention to termination-insensitive non-interference, the program logic is equipped with a partial-correctness interpretation. Similar to specifications in the VDM formalism [21], assertions are binary predicates over states. In Section 3 we consider non-interference in the style of Volpano et al. [37], restricted to the binary lattice  $low \sqsubset high$ . We introduce a *derived proof system* that interprets typing judgements as judgements in the program logic, and present program-logic counterparts of the typing rules. We then consider two extensions. In Section 4, we generalise our approach to multi-level security and flow sensitivity by giving an interpretation of the type system of Hunt and Sands [19]. In Section 5, we return to binary, flow-insensitive non-interference, but extend the language by object-manipulating instructions. Finally, we outline future and related work and discuss shortcomings of our approach in Section 6.

The material presented in the technical Sections 2 to 5 is based on a formalisation of the entire development in the theorem prover Isabelle/HOL. As the source files of this development are available electronically [11], we omit most proofs from our presentation.

## 2 Syntax and semantics of IMP

**Syntax** The syntax of **IMP** is defined over a set  $\mathcal{X}$  of variables (ranged over by  $x$ ), a set  $\mathcal{V}$  of values (ranged over

by  $v$ ), arithmetic operations  $aop$  (plus, minus, ...) and boolean operations  $bop$  (less, equal, ...):

$$\begin{aligned} e \in Expr & ::= x \mid v \mid e \ aop \ e \\ b \in BExpr & ::= e \ bop \ e \\ C \in Com & ::= \mathbf{Skip} \mid x := e \mid C; C \mid \mathbf{While} \ b \ \mathbf{do} \ C \\ & \quad \mid \mathbf{If} \ b \ \mathbf{then} \ C \ \mathbf{else} \ C \mid \mathbf{Call} \end{aligned}$$

In addition to the language constructs covered in [38], we consider the instruction **Call** which represents the invocation of a fixed, parameterless, but possibly recursive, procedure. We denote the command representing the body of this procedure by **Body**.

**Operational semantics** The operational semantics is given by a big-step relation  $\sigma \xrightarrow{C} \tau$  where  $\sigma$  and  $\tau$  are states, i.e. members of the set  $\Sigma \equiv \mathcal{X} \rightarrow \mathcal{V}$ . The relation is defined by the rules in Fig. 1, where  $\llbracket \cdot \rrbracket_\sigma$  denotes the evaluation of an arithmetic or boolean expression in state  $\sigma$ .

**Axiomatic semantics** The axiomatic semantics is given by a proof system for partial correctness with judgements in precondition-less VDM style. Assertions  $A$  are binary state relations, i.e. are of type  $\mathcal{A} \equiv \Sigma \rightarrow \Sigma \rightarrow \mathcal{B}$  where  $\mathcal{B}$  denotes the set of booleans. Command  $C$  *satisfies* an assertion, notation  $\models C : A$ , if for all  $\sigma$  and  $\tau$ ,  $\sigma \xrightarrow{C} \tau$  implies  $A \ \sigma \ \tau$ . The proof system employs judgements of the form  $G \triangleright C : A$  where context  $G$  represents a finite set of assertions which are assumed to hold for the unnamed procedure. Fig. 2 presents the proof rules. In addition to the syntax-directed rules **SKIP**, ..., **WHILE**, we have an axiom rule that extracts procedure specifications from  $G$ , and a rule of consequence.

Rule **WHILE** is a little unusual, but is better suited for proving the derived proof rule in the following section. It is in fact inter-derivable with the standard rule

$$\frac{G \triangleright C : \lambda \sigma \tau. \forall \rho. \llbracket b \rrbracket_\sigma \rightarrow I \ \sigma \ \rho \rightarrow I \ \tau \ \rho}{G \triangleright \mathbf{While} \ b \ \mathbf{do} \ C : \lambda \sigma \tau. \forall \rho. I \ \sigma \ \rho \rightarrow (I \ \tau \ \rho \wedge \neg \llbracket b \rrbracket_\tau)}.$$

The admissible rule

$$\frac{A \in G \quad \forall B \in G. G \triangleright \mathbf{Body} : B}{\emptyset \triangleright \mathbf{Call} : A}$$

allows one to derive the validity of a judgement in the empty proof context, provided that all assumptions in  $G$  can be validated by a corresponding proof for the method body.

Using standard techniques [24, 31], it is easy to show soundness and relative completeness of the proof system, i.e. the following property.

**Theorem 1.** (*Soundness and completeness of program logic*) *The derivability of the judgement  $\emptyset \triangleright C : A$  is equivalent to  $\models C : A$ , i.e., its semantic validity.*

---


$$\begin{array}{c}
\text{SKIP} \frac{}{\sigma \xrightarrow{\text{Skip}} \sigma} \quad \text{ASSIGN} \frac{}{\sigma \xrightarrow{x:=e} \sigma[x \mapsto \llbracket e \rrbracket_\sigma]} \quad \text{COMP} \frac{\sigma \xrightarrow{C} \rho \quad \rho \xrightarrow{D} \tau}{\sigma \xrightarrow{C;D} \tau} \quad \text{WHILET} \frac{\llbracket b \rrbracket_\sigma \quad \sigma \xrightarrow{C} \rho \quad \rho \xrightarrow{\text{While } b \text{ do } C} \tau}{\sigma \xrightarrow{\text{While } b \text{ do } C} \tau} \\
\text{WHILEF} \frac{\neg \llbracket b \rrbracket_\sigma}{\sigma \xrightarrow{\text{While } b \text{ do } C} \sigma} \quad \text{IFT} \frac{\llbracket b \rrbracket_\sigma \quad \sigma \xrightarrow{C} \tau}{\sigma \xrightarrow{\text{If } b \text{ then } C \text{ else } D} \tau} \quad \text{IFF} \frac{\neg \llbracket b \rrbracket_\sigma \quad \sigma \xrightarrow{D} \tau}{\sigma \xrightarrow{\text{If } b \text{ then } C \text{ else } D} \tau} \quad \text{CALL} \frac{\sigma \xrightarrow{\text{Body}} \tau}{\sigma \xrightarrow{\text{Call}} \tau}
\end{array}$$


---

Figure 1. Operational semantics

---


$$\begin{array}{c}
\text{SKIP} \frac{}{G \triangleright \text{Skip} : \lambda \sigma \tau. \tau = \sigma} \quad \text{ASSIGN} \frac{}{G \triangleright x:=e : \lambda \sigma \tau. \tau = \sigma[x \mapsto \llbracket e \rrbracket_\sigma]} \quad \text{CALL} \frac{\{A\} \cup G \triangleright \text{Body} : A}{G \triangleright \text{Call} : A} \\
\text{COMP} \frac{G \triangleright C : A \quad G \triangleright D : B}{G \triangleright C;D : \lambda \sigma \tau. \exists \rho. A \sigma \rho \wedge B \rho \tau} \quad \text{IF} \frac{G \triangleright C : A \quad G \triangleright D : B}{G \triangleright \text{If } b \text{ then } C \text{ else } D : \lambda \sigma \tau. (\llbracket b \rrbracket_\sigma \rightarrow A \sigma \tau) \wedge (\neg \llbracket b \rrbracket_\sigma \rightarrow B \sigma \tau)} \\
\text{WHILE} \frac{G \triangleright C : B \quad \forall \sigma. \neg \llbracket b \rrbracket_\sigma \rightarrow A \sigma \sigma \quad \forall \sigma \rho \tau. \llbracket b \rrbracket_\sigma \rightarrow B \sigma \rho \rightarrow A \rho \tau \rightarrow A \sigma \tau}{G \triangleright \text{While } b \text{ do } C : \lambda \sigma \tau. A \sigma \tau \wedge \neg \llbracket b \rrbracket_\tau} \quad \text{AX} \frac{A \in G}{G \triangleright \text{Call} : A} \quad \text{CONSEQ} \frac{G \triangleright C : A \quad \forall \sigma \tau. A \sigma \tau \rightarrow B \sigma \tau}{G \triangleright C : B}
\end{array}$$


---

Figure 2. Axiomatic semantics

### 3 Base-line non-interference

In order to convince a code consumer that program  $C$  is non-interferent, a simple solution would be to show that the final values of the low variables only depend upon the initial values of these variables, i.e. that there is some function  $f$  such that  $C$  satisfies  $\lambda \sigma \tau. \tau|_{\mathcal{X}_{low}} = f(\sigma|_{\mathcal{X}_{low}})$ . The downside of this approach is that the code producer is left in the dark as to how such  $f$  could be constructed unless the semantics of  $C$  is known. Instead, our approach draws upon the principle of self-composition which can be described as follows. In order to express security of program  $C$  let  $C_1, C_2$  be copies of  $C$  on mutually disjoint sets of variables. Write states  $\sigma$  as  $\sigma = (\sigma_1, \sigma_2)$  where  $\sigma_i$  is the part of  $\sigma$  that refers to the variables in  $C_i$ . Now the definition of when  $C$  is secure coincides with

$$\models C_1; C_2 : \lambda (\sigma_1, \sigma_2) (\tau_1, \tau_2). \sigma_1 \sim \sigma_2 \Rightarrow \tau_1 \sim \tau_2.$$

In order to show that security is expressible in VDM, we define the following operator  $Sec(\phi)$  for predicates  $\phi$  of type  $\mathcal{T} \equiv (\Sigma \times \Sigma) \rightarrow \mathcal{B}$ .

**Definition 2** (Sec). For  $\phi : \mathcal{T}$  we let  $Sec(\phi)$  denote the assertion

$$Sec(\phi) \equiv \lambda \sigma \tau. (\forall \rho. \sigma \sim \rho \Rightarrow \phi(\tau, \rho)) \wedge (\forall \rho. \phi(\rho, \sigma) \Rightarrow \rho \sim \tau)$$

Formula  $\phi$  factors non-interference into two parts by communicating the information between the two conjuncts in a similar way as an assertion that applies at the state separating the two halves of a self-composed program. Security can now be characterised as follows.

**Proposition 1.** If  $\models C : Sec(\phi)$  then  $C$  is secure. Conversely, if  $C$  is secure then  $\models C : Sec(\phi)$  for  $\phi \equiv \lambda (\rho, \tau). \exists \sigma. \sigma \xrightarrow{C} \rho \wedge \sigma \sim \tau$ .

*Proof.* Suppose  $\models C : Sec(\phi)$  and  $\sigma \xrightarrow{C} \tau$  and  $\sigma' \xrightarrow{C} \tau'$  and  $\sigma \sim \sigma'$ . Applying  $\models C : Sec(\phi)$  to  $\sigma \xrightarrow{C} \tau$  gives  $\phi(\tau, \sigma')$ . Applying  $\models C : Sec(\phi)$  again, but now to  $\sigma' \xrightarrow{C} \tau'$ , gives  $\tau \sim \tau'$  as required. For the opposite direction, suppose  $\sigma \xrightarrow{C} \tau$ . The proof of  $(Sec(\lambda (\rho, \tau). \exists \sigma. \sigma \xrightarrow{C} \rho \wedge \sigma \sim \tau)) \sigma \tau$  consists of two parts. First, we have to show that there is some  $\omega$  with  $\omega \xrightarrow{C} \tau$  and  $\omega \sim \rho$  whenever  $\sigma \sim \rho$  holds. Choose  $\omega := \sigma$ . Second, we have to show that  $\rho \sim \tau$  holds whenever  $\omega \xrightarrow{C} \rho$  for some  $\omega$  with  $\omega \sim \sigma$ . This follows from the definition of  $C$  secure.  $\square$

As a consequence of this proposition one may verify that a concrete program  $C$  is non-interferent by considering a property (namely  $Sec(\phi)$ ) that is phrased as a judgement formally involving only a single execution of  $C$ , despite the occurrence of two operational statements in the the definition of non-interference (Definition 1).

For proving  $C$  secure, it thus suffices to exhibit an arbitrary  $\phi$  with  $G \triangleright C : Sec(\phi)$ . In order to support proof generation, we employ a type system. Although often incomplete, type systems provide a powerful technique to syntactically identify classes of programs with desirable runtime properties. In case of the runtime property  $Sec(\phi)$ , an appropriate type system is that of Volpano, Smith, and Irvine [37], restricted to the binary lattice  $low \sqsubseteq high$ . Using the presentation of this type system given by Sabelfeld

---

Security levels:  $i ::= low \mid high$  Variables:  $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$  Typing judgements:  $\vdash e : i, \vdash b : i$ , and  $[i] \vdash C$

$$\begin{array}{c}
\text{T-EXP-H} \frac{}{\vdash e : high} \quad \text{T-EXP-L} \frac{\mathcal{X}_{high} \cap \text{Vars}(e) = \emptyset}{\vdash e : low} \quad \text{T-BEXP} \frac{\vdash e_1 : i \quad \vdash e_2 : i}{\vdash e_1 \text{ bop } e_2 : i} \\
\text{T-SKIP} \frac{}{[high] \vdash \text{Skip}} \quad \text{T-ASSIGN-H} \frac{}{[i] \vdash h := e} \quad \text{T-ASSIGN-L} \frac{\vdash e : low}{[low] \vdash l := e} \quad \text{T-COMP} \frac{[i] \vdash C_1 \quad [i] \vdash C_2}{[i] \vdash C_1; C_2} \\
\text{T-WHILE} \frac{\vdash b : i \quad [i] \vdash C}{[i] \vdash \text{While } b \text{ do } C} \quad \text{T-IF} \frac{\vdash b : i \quad [i] \vdash C_1 \quad [i] \vdash C_2}{[i] \vdash \text{If } b \text{ then } C_1 \text{ else } C_2} \quad \text{T-SUB} \frac{[high] \vdash C}{[low] \vdash C}
\end{array}$$

**Figure 3. Volpano-Smith-Irvine Type System**

---

and Myers [33], we now show how typing derivations automatically yield the desired relations  $\phi$  along with derivations of  $G \triangleright C : Sec(\phi)$ . The type system is given in Figure 3, and guarantees security in the following sense:

**Proposition 2.** *If  $\vdash e : low$  then  $\sigma \sim \tau \Rightarrow \llbracket e \rrbracket_\sigma = \llbracket e \rrbracket_\tau$ . If  $\vdash b : low$  then  $\sigma \sim \tau \Rightarrow \llbracket b \rrbracket_\sigma = \llbracket b \rrbracket_\tau$ . If  $[low] \vdash C$  then  $C$  is secure. If  $[high] \vdash C$  then  $\sigma \xrightarrow{C} \tau$  implies  $\sigma \sim \tau$ .*

Consequently, we interpret a typing judgement  $[low] \vdash C$  as a judgement  $G \triangleright C : Sec(\phi)$  for some suitable  $\phi$ , and a typing judgement  $[high] \vdash C$  as a judgement  $G \triangleright C : HiSec$ , where *HiSec* abbreviates the assertion  $\lambda \sigma \tau. \sigma \sim \tau$ .

We translate the typing judgements into VDM derivations of security assertions as shown in Figure 4. In particular, the rules constructively exhibit assertions  $\phi$  for the commands typeable in a low context. Similar to the derived assertions in [12] and [10], the formulae  $\phi$  (and thus the assertions  $Sec(\phi)$ ) are syntax-dependent, i.e. the assertions in the conclusions are composed from the assertions in the assumptions using different combinators for the various instruction forms. Together with the first part of Proposition 2, the side condition  $\vdash b : low$  in the rules *IFL* and *WHILEL* ensures that no “diagonal cases” need to be considered, i.e. situations where (operationally) the branch condition evaluates differently in the two executions.

**Proposition 3.** *The rules in Figure 4 are derivable in the program logic.*

The derived proof system inherits the incompleteness of the type system, i.e. it is only suitable for validating non-interference, not disproving it. However, all derivations in the system of Volpano et al. are covered:

**Theorem 2.** *Let  $[i] \vdash C$ . If  $i = high$  then  $G \triangleright C : HiSec$  holds, and if  $i = low$  then there is some  $\phi$  such that  $G \triangleright C : Sec(\phi)$ .*

The proof proceeds by induction on  $[i] \vdash C$ .

**Recursive procedures** We can extend the type system to cover the (possibly recursive) procedure as follows. One assumes a typing for **Call** and must show that the body has the announced typing where possible recursive calls may be assumed well-typed. For example, extending the type system by the rule

$$\text{T-CALL} \frac{}{[low] \vdash \text{Call}}$$

models the situation where the unnamed procedure is assumed to be of type  $[low]$ . A complete proof then includes a proof of  $[low] \vdash \text{Body}$ , which amounts to a formalised justification of the hypothetical typing of the procedure.

In order to map this situation to the program logic, we start from the context  $G = \{Sec(X)\}$  where  $X$  is a predicate variable of type  $\mathcal{T}$ . From the proof of  $[low] \vdash \text{Body}$  we obtain (using the rules from Figure 4) a proof of  $G \triangleright \text{Body} : Sec(\phi)$  where  $\phi$  is some relation on states and contains  $X$  if the procedure is recursive, i.e. if **Body** contains further occurrences of **Call**. Inspection of the rules shows that (even in the case of recursive procedures)  $X$  occurs positively in  $\phi(X)$  so that there exists a relation  $\phi_0$  of type  $\mathcal{T}$ , e.g. the least fixed point of the operator  $\lambda X. \phi(X)$ , satisfying  $\phi_0 \leftrightarrow \phi(\phi_0)$ . With rule *CONSEQ* we thus obtain  $\{Sec(\phi_0)\} \triangleright \text{Body} : Sec(\phi_0)$ . Applying the context elimination rule mentioned above allows us to conclude  $\emptyset \triangleright \text{Call} : Sec(\phi_0)$  and thus the security of **Call**.

In the formalisation, we define the explicit fixed point operator  $Fix : (\mathcal{T} \rightarrow \mathcal{T}) \rightarrow \mathcal{T}$  by

$$Fix(\Phi)(\sigma, \tau) \equiv \forall \phi. (\forall \sigma' \tau'. \Phi \phi(\sigma', \tau') \Rightarrow \phi(\sigma', \tau')) \Rightarrow \phi(\sigma, \tau).$$

It is easily shown that  $Fix$  satisfies the fixed point property

$$\Phi(Fix(\Phi)) = Fix(\Phi), \quad (1)$$

provided that  $\Phi$  is *monotone*, i.e. all  $\phi$  and  $\psi$  satisfy

$$(\forall \sigma \tau. \phi(\sigma, \tau) \Rightarrow \psi(\sigma, \tau)) \Rightarrow (\forall \sigma \tau. \Phi \phi(\sigma, \tau) \Rightarrow \Phi \psi(\sigma, \tau)).$$

$$\begin{array}{c}
\text{SKIPH} \frac{}{G \triangleright \mathbf{Skip} : \text{HiSec}} \quad \text{SKIPL} \frac{}{G \triangleright \mathbf{Skip} : \text{Sec}(\lambda(\sigma, \tau). \sigma \sim \tau)} \quad \text{SUB} \frac{G \triangleright C : \text{HiSec}}{G \triangleright C : \text{Sec}(\lambda(\sigma, \tau). \sigma \sim \tau)} \\
\\
\text{ASSIGNH} \frac{}{G \triangleright h := e : \text{HiSec}} \quad \text{ASSIGNL} \frac{\forall \sigma \tau. \sigma \sim \tau \Rightarrow \llbracket e \rrbracket_\sigma = \llbracket e \rrbracket_\tau}{G \triangleright l := e : \text{Sec}(\lambda(\sigma, \tau). \sigma \sim \tau \llbracket l \rrbracket \mapsto \llbracket e \rrbracket_\tau)} \\
\\
\text{COMPH} \frac{G \triangleright C_1 : \text{HiSec} \quad G \triangleright C_2 : \text{HiSec}}{G \triangleright C_1; C_2 : \text{HiSec}} \quad \text{COMPL} \frac{G \triangleright C_1 : \text{Sec}(\phi) \quad G \triangleright C_2 : \text{Sec}(\psi)}{G \triangleright C_1; C_2 : \text{Sec}(\lambda(\sigma, \tau). \exists \rho. \phi(\rho, \tau) \wedge (\forall \omega. \rho \sim \omega \Rightarrow \psi(\sigma, \omega)))} \\
\\
\text{IFH} \frac{G \triangleright C_1 : \text{HiSec} \quad G \triangleright C_2 : \text{HiSec}}{G \triangleright \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 : \text{HiSec}} \quad \text{IFL} \frac{\forall \sigma \tau. \sigma \sim \tau \Rightarrow \llbracket b \rrbracket_\sigma = \llbracket b \rrbracket_\tau \quad G \triangleright C_1 : \text{Sec}(\phi) \quad G \triangleright C_2 : \text{Sec}(\psi)}{G \triangleright \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 : \text{Sec}(\lambda(\sigma, \tau). (\llbracket b \rrbracket_\tau \Rightarrow \phi(\sigma, \tau)) \wedge (\neg \llbracket b \rrbracket_\tau \Rightarrow \psi(\sigma, \tau)))} \\
\\
\text{WHILEH} \frac{G \triangleright C : \text{HiSec}}{G \triangleright \mathbf{While } b \mathbf{ do } C : \text{HiSec}} \quad \text{WHILEL} \frac{\forall \sigma \tau. \sigma \sim \tau \Rightarrow \llbracket b \rrbracket_\sigma = \llbracket b \rrbracket_\tau \quad G \triangleright C : \text{Sec}(\phi)}{G \triangleright \mathbf{While } b \mathbf{ do } C : \text{Sec}(\lambda(\sigma, \tau). \text{var}(b, \phi, \sigma, \tau))}
\end{array}$$

Relation  $\text{var}$  is defined inductively by the rules

$$\frac{\neg \llbracket b \rrbracket_\tau \quad \sigma \sim \tau}{\text{var}(b, \phi, \sigma, \tau)} \quad \frac{\llbracket b \rrbracket_\tau \quad \phi(\rho, \tau) \quad \forall \omega. \rho \sim \omega \Rightarrow \text{var}(b, \phi, \sigma, \omega)}{\text{var}(b, \phi, \sigma, \tau)}$$

and is hence monotone in  $\phi$ .

**Figure 4. Derived proof rules for base-line non-interference**

The low Call rule requires the body to satisfy the security assertion for one unfolding of property (1) and includes the monotonicity property as a side condition

$$\text{CALLL} \frac{\text{monotone}(\Phi) \quad \{ \text{Sec}(\text{Fix}(\Phi)) \} \cup G \triangleright \mathbf{Body} : \text{Sec}(\Phi(\text{Fix}(\Phi)))}{G \triangleright \mathbf{Call} : \text{Sec}(\text{Fix}(\Phi))}.$$

The high call rule is the expected

$$\text{CALLH} \frac{\{ \text{HiSec} \} \cup G \triangleright \mathbf{Body} : \text{HiSec}}{G \triangleright \mathbf{Call} : \text{HiSec}}.$$

## 4 Multi-level security and flow sensitivity

We now generalise our approach to more permissive notions of information flow security. In this section, we treat the generic type system proposed by Hunt and Sands [19]. In this system, the two-point lattice  $\text{low} \sqsubset \text{high}$  is replaced by an arbitrary lattice  $\mathcal{L}$ . Furthermore, we relax the fixed association of program variables to security levels by allowing variables to *float*, i.e. to be associated with different security levels at different program points. Typing judgements take the form  $p \vdash \Gamma \{C\} \Delta$  where  $p \in \mathcal{L}$  and  $\Gamma$  and  $\Delta$  are contexts that describe the security levels associated with the program variables prior to, and after, the execution of  $C$ , respectively. Figure 5 presents the typing rules of Hunt and Sands, together with the obvious rules for arithmetic and

boolean expressions<sup>1</sup>

The security property guaranteed by this system may be expressed as an instantiation of a general relational notion of validity that is commonly used in the literature [9, 19]. For state relations  $R$  and  $S$  we say that  $C$  is  $R \Rightarrow S$ -secure, notation  $\models C : R \Rightarrow S$ , if the relation  $R$  before  $C$  ensures the relation  $S$  after  $C$ :

$$\models C : R \Rightarrow S \equiv \forall \sigma \sigma' \tau \tau'. \sigma \xrightarrow{C} \tau \Rightarrow \sigma' \xrightarrow{C} \tau' \Rightarrow R \sigma \sigma' \Rightarrow S \tau \tau'.$$

If  $C$  furthermore satisfies assertion  $A$ , we call  $C$  to be  $A; R \Rightarrow S$ -secure, notation  $\models C : A; R \Rightarrow S$ :

$$\models C : A; R \Rightarrow S \equiv \models C : A \wedge \models C : R \Rightarrow S.$$

Base-line non-interference as considered in the previous section thus amounts to  $(\lambda \sigma \tau. \text{True}); \sim \Rightarrow \sim$ -security.

Continuing in parallel with the development of Section 3, we introduce a generalisation of the operator  $\text{Sec}(\cdot)$  that replaces the negatively occurring  $\sim$  by a relation  $R$ , the positively occurring one by a relation  $S$ , and is parametric in an assertion  $A$  that relates initial and final states.

**Definition 3 (Sec).** For  $\phi$  of type  $\mathcal{T}$ , relations  $R, S$  and assertion  $A$  we let  $\text{Sec}_{A, R \Rightarrow S}(\phi)$  denote the assertion

$$\text{Sec}_{A, R \Rightarrow S}(\phi) \equiv \lambda \sigma \tau. A \sigma \tau \wedge (\forall \rho. R \sigma \rho \Rightarrow \phi(\tau, \rho)) \wedge (\forall \rho. \phi(\rho, \sigma) \Rightarrow S \rho \tau).$$

<sup>1</sup>Our rule for conditionals is interderivable with the rule given by Hunt and Sands,

$$\frac{\Gamma \vdash b : q \quad p \sqcup q \vdash \Gamma \{C_i\} \Delta}{p \vdash \Gamma \{ \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 \} \Delta}.$$

$$\begin{array}{c}
\frac{}{\Gamma \vdash x : \Gamma(x)} \quad \frac{}{\Gamma \vdash v : \perp} \quad \frac{\Gamma \vdash e_i : p_i}{\Gamma \vdash e_1 \text{ aop } e_2 : p_1 \sqcup p_2} \quad \frac{\Gamma \vdash e : p \quad p \sqsubseteq q}{\Gamma \vdash e : q} \quad \frac{\Gamma \vdash e_i : p_i}{\Gamma \vdash e_1 \text{ bop } e_2 : p_1 \sqcup p_2} \quad \frac{\Gamma \vdash b : p \quad p \sqsubseteq q}{\Gamma \vdash b : q} \\
\frac{}{p \vdash \Gamma \{\text{Skip}\} \Gamma} \quad \frac{\Gamma \vdash e : q}{p \vdash \Gamma \{x := e\} \Gamma[x \mapsto p \sqcup q]} \quad \frac{p \vdash \Gamma \{C_1\} \Theta \quad p \vdash \Theta \{C_2\} \Delta}{p \vdash \Gamma \{C_1; C_2\} \Delta} \quad \frac{\Gamma \vdash b : p \quad p \vdash \Gamma \{C_i\} \Delta}{p \vdash \Gamma \{\text{If } b \text{ then } C_1 \text{ else } C_2\} \Delta} \\
\frac{\Gamma \vdash b : q \quad p \sqcup q \vdash \Gamma \{C\} \Gamma}{p \vdash \Gamma \{\text{While } b \text{ do } C\} \Gamma} \quad \frac{p' \vdash \Gamma' \{C\} \Delta' \quad p \sqsubseteq p' \quad \Gamma \sqsubseteq \Gamma' \quad \Delta' \sqsubseteq \Delta}{p \vdash \Gamma \{C\} \Delta}
\end{array}$$

Figure 5. The type system of Hunt and Sands

In analogy to Proposition 1 we have the following result.

**Proposition 4.** *For any  $C$ ,  $A$ ,  $R$ , and  $S$ , the following holds. If  $\models C : Sec_{A,R \Rightarrow S}(\phi)$  for some  $\phi$  then  $\models C : A; R \Rightarrow S$ . Conversely, if  $\models C : A; R \Rightarrow S$  then  $\models C : Sec_{A,R \Rightarrow S}(\phi)$  holds for  $\phi = \lambda(\rho, \tau). \exists \sigma. \sigma \xrightarrow{C} \rho \wedge R \sigma \tau$ .*

The interpretation of the Hunt-Sands type system generalises relation  $\sim$  to a family  $\times_{\Gamma}^p$  of indistinguishability relations over states, indexed by a security level and a context<sup>2</sup>. Each  $\times_{\Gamma}^p$  (also written in infix position) is defined by

$$\sigma \times_{\Gamma}^p \tau \equiv \forall x. \Gamma(x) \sqsubseteq p \Rightarrow \sigma(x) = \tau(x)$$

and expresses that  $\sigma$  and  $\tau$  agree on all variables whose type in  $\Gamma$  is dominated by  $p$ . Also indexed by a security level and a context is the assertion  $\mathcal{Q}_{p,\Delta}$ , defined by

$$\mathcal{Q}_{p,\Delta} \equiv \lambda \sigma \tau. \forall x. p \not\sqsubseteq \Delta(x) \Rightarrow \tau(x) = \sigma(x).$$

This assertion is satisfied by programs that leave those variables unchanged whose type in  $\Delta$  is not greater than  $p$ .

The security property considered by Hunt and Sands is now as follows.

**Definition 4.**  *$C$  is called secure for  $p$ ,  $\Gamma$ , and  $\Delta$ , notation  $p \models \Gamma \{C\} \Delta$ , if for all  $q$ ,  $\models C : \mathcal{Q}_{p,\Delta}; \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q$  holds.*

The unary property  $p \models C : \mathcal{Q}_{p,\Delta}$  expresses that together with  $p$ , the terminal context  $\Delta$  contains information regarding the assignments in  $C$ : at most variables  $x$  with  $p \sqsubseteq \Delta(x)$  are assigned to. The relational property  $\models C : \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q$  for all  $q$  is the information flow guarantee: final states  $\tau$  and  $\tau'$  are indistinguishable (with respect to  $\Delta$ ) at level  $q$  provided that the initial states  $\sigma$  and  $\sigma'$  were indistinguishable at  $q$  with respect to  $\Gamma$ . This latter property is independent of  $p$ . Hunt and Sands' soundness result is now as follows.

**Theorem 3. (Hunt & Sands)** *If  $p \vdash \Gamma \{C\} \Delta$  then  $p \models \Gamma \{C\} \Delta$ .*

<sup>2</sup>Our notation  $\times_{\Gamma}^p$  is inspired by the work of Amtoft et al. [2].

By phrasing Hunt and Sands' security condition in a single judgement, both properties (the unary and the relational one) are available for the hypothetical judgements in the proofs of the derived proof rules introduced below.

Specialising Proposition 4 to the relations and assertions suitable for Hunt and Sands' system yields that  $C$  is secure for  $p$ ,  $\Gamma$ , and  $\Delta$  precisely if

$$\models C : Sec_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^p \Rightarrow \times_{\Delta}^p}(\lambda(\rho, \tau). \exists \sigma. \sigma \xrightarrow{C} \rho \wedge \sigma \times_{\Gamma}^p \tau),$$

but for establishing the security of  $C$ , it suffices to present an arbitrary  $\phi$  with  $\models C : Sec_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^p \Rightarrow \times_{\Delta}^p}(\phi)$ . Appropriate proof rules are collected in Figure 6 and are of a similar structure as the rules presented in Section 3: formulae  $\phi$  in the conclusions are obtained from the  $\phi$ s occurring in the hypotheses. In contrast to the proof system for baseline non-interference, there are two rules for conditionals and while-loops each. The reason for this is that the “diagonal” cases (where the branch/loop condition evaluates differently for the two executions of the phrase) do not vanish. In the case of conditionals, rule IF-P covers the case where identical branches are taken and delivers a postcondition depending on this branch (i.e. involves the  $\phi_i$ ). Rule IF-D applies when different branches are taken and only guarantees indistinguishability with respect to the common terminal context  $\Delta$ . Similarly, rule WHILE-P covers cases where the executions agree on the outcome of the guard  $b$ , while WHILE-D covers the diagonal cases. Nevertheless, the proof system is (apart from rule SUB) algorithmic due to the positive and negative occurrences of the guards  $\exists x. p \sqsubseteq \Delta(x) \wedge \Delta(x) \sqsubseteq q$  and  $p \sqsubseteq q$ .

**Proposition 5.** *The rules in Figure 6 are derivable in the program logic.*

In the the proof of rule WHILE-D, the assertion called  $A$  in the VDM rule WHILE is instantiated to the assertion  $\lambda \sigma \tau. \text{varD}(b, \mathcal{Q}_{p,\Gamma}, \sigma, \tau)$ , where the inductive relation  $\text{varD}$  is defined by the rules

$$\frac{\neg \llbracket b \rrbracket_{\sigma} \quad B \sigma \tau}{\text{varD}(b, B, \sigma, \tau)} \quad \frac{\llbracket b \rrbracket_{\sigma} \quad B \sigma \rho \quad \text{varD}(b, B, \rho, \tau)}{\text{varD}(b, B, \sigma, \tau)}.$$

---


$$\begin{array}{c}
\text{SKIP} \frac{}{G \triangleright \mathbf{Skip} : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Gamma}^q}(\lambda(\sigma, \tau). \sigma \times_{\Gamma}^q \tau)} \\
\text{ASSIGN} \frac{\Delta = \Gamma[x \mapsto p \sqcup t] \quad \forall \sigma \rho. \sigma \times_{\Gamma}^t \rho \Rightarrow \llbracket e \rrbracket_{\sigma} = \llbracket e \rrbracket_{\rho}}{G \triangleright x := e : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\lambda(\sigma, \tau). \exists \rho. \sigma = \rho[x \mapsto \llbracket e \rrbracket_{\rho}] \wedge \rho \times_{\Gamma}^q \tau)} \\
\text{COMP} \frac{G \triangleright C_1 : \text{Sec}_{\mathcal{Q}_{p,\Theta}, \times_{\Gamma}^q \Rightarrow \times_{\Theta}^q}(\phi) \quad \forall x. \Gamma(x) \sqsubseteq \Theta(x) \vee p \sqsubseteq \Theta(x) \\ G \triangleright C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Theta}^q \Rightarrow \times_{\Delta}^q}(\psi) \quad \forall x. \Theta(x) \sqsubseteq \Delta(x) \vee p \sqsubseteq \Delta(x)}{G \triangleright C_1; C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\lambda(\sigma, \tau). \exists \rho. \phi(\rho, \tau) \wedge (\forall \omega. \rho \times_{\Theta}^q \omega \rightarrow \psi(\sigma, \omega)))} \\
\text{IF-P} \frac{G \triangleright C_1 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\phi) \quad \exists x. p \sqsubseteq \Delta(x) \wedge \Delta(x) \sqsubseteq q \\ G \triangleright C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\psi) \quad \forall \sigma \tau. \sigma \times_{\Gamma}^p \tau \rightarrow \llbracket b \rrbracket_{\sigma} = \llbracket b \rrbracket_{\tau} \\ \forall x. \Gamma(x) \sqsubseteq \Delta(x) \vee p \sqsubseteq \Delta(x)}{G \triangleright \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\phi_{If})} \\
\text{IF-D} \frac{G \triangleright C_1 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\phi) \quad \forall x. \Gamma(x) \sqsubseteq \Delta(x) \vee p \sqsubseteq \Delta(x) \\ G \triangleright C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\psi) \quad \neg(\exists x. p \sqsubseteq \Delta(x) \wedge \Delta(x) \sqsubseteq q)}{G \triangleright \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\lambda(\sigma, \tau). \sigma \times_{\Delta}^q \tau)} \\
\text{WHILE-P} \frac{G \triangleright C : \text{Sec}_{\mathcal{Q}_{p,\Gamma}, \times_{\Gamma}^q \Rightarrow \times_{\Gamma}^q}(\phi) \quad \forall \sigma \tau. \sigma \times_{\Gamma}^p \tau \rightarrow \llbracket b \rrbracket_{\sigma} = \llbracket b \rrbracket_{\tau} \quad p \sqsubseteq q}{G \triangleright \mathbf{While } b \mathbf{ do } C : \text{Sec}_{\mathcal{Q}_{p,\Gamma}, \times_{\Gamma}^q \Rightarrow \times_{\Gamma}^q}(\phi_{While})} \\
\text{WHILE-D} \frac{G \triangleright C : \text{Sec}_{\mathcal{Q}_{p,\Gamma}, \times_{\Gamma}^q \Rightarrow \times_{\Gamma}^q}(\phi) \quad p \not\sqsubseteq q}{G \triangleright \mathbf{While } b \mathbf{ do } C : \text{Sec}_{\mathcal{Q}_{p,\Gamma}, \times_{\Gamma}^q \Rightarrow \times_{\Gamma}^q}(\lambda(\sigma, \tau). \sigma \times_{\Gamma}^q \tau)} \\
\text{SUB} \frac{G \triangleright C : \text{Sec}_{\mathcal{Q}_{p',\Delta'}, \times_{\Gamma'}^q \Rightarrow \times_{\Delta'}^q}(\phi) \quad p \sqsubseteq p' \quad \Gamma \sqsubseteq \Gamma' \quad \Delta' \sqsubseteq \Delta}{G \triangleright C : \text{Sec}_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\phi)}
\end{array}$$

The assertions  $\phi_{If}$  and  $\phi_{While}$  are defined by

$$\begin{aligned}
\phi_{If} &\equiv \lambda(\sigma, \tau). (\llbracket b \rrbracket_{\tau} \rightarrow \phi(\sigma, \tau)) \wedge (\neg \llbracket b \rrbracket_{\tau} \rightarrow \psi(\sigma, \tau)) \\
\phi_{While} &\equiv \lambda(\sigma, \tau). \text{var}(b, \times_{\Gamma}^q, \phi, \sigma, \tau),
\end{aligned}$$

and the variant  $\text{var}$  in rule WHILE-P is defined inductively by the rules

$$\frac{\neg \llbracket b \rrbracket_{\tau} \quad A \sigma \tau}{\text{var}(b, A, \phi, \sigma, \tau)} \quad \frac{\llbracket b \rrbracket_{\tau} \quad \phi(\rho, \tau) \quad \forall \omega. A \rho \omega \Rightarrow \text{var}(b, A, \phi, \sigma, \omega)}{\text{var}(b, A, \phi, \sigma, \tau)},$$

and is monotone in  $\phi$ .

**Figure 6. Derived proof rules for Hunt and Sands' type system**

---

Satisfaction of those side conditions of the rules that are not guards follows from the hypotheses of the typing rules. In particular, the following properties are easily shown.

**Lemma 1.** • If  $p \vdash \Gamma \{C\} \Delta$  then for all  $x$ ,  $\Gamma(x) \sqsubseteq \Delta(x) \vee p \sqsubseteq \Delta(x)$ .

- If  $\Gamma \vdash e : p$  then for all  $\sigma, \tau$ , and  $q$ ,  $\sigma \times_{\Gamma}^q \tau$  and  $p \sqsubseteq q$  implies  $\llbracket e \rrbracket_{\sigma} = \llbracket e \rrbracket_{\tau}$  (Similarly for boolean expressions).

With the help of this lemma, one may show that typability of  $C$  entails the derivability of a VDM judgement of the restricted form:

**Theorem 4.** If  $p \vdash \Gamma \{C\} \Delta$  then  $G \triangleright C : Sec_{\mathcal{Q}_{p,\Delta}, \times_{\Gamma}^q \Rightarrow \times_{\Delta}^q}(\phi)$  for some  $\phi$ .

The proof proceeds by an induction on the typing rules.

Combining this result with Proposition 4 yields a formalised proof of Theorem 3 that is factored into a generic part – the soundness proof of the program logic – and a type-system specific part – the justification of the typing rules by their interpretation in the program logic.

**Recursive procedures** A general proof rule for the procedure invocation instruction **Call** is

$$\text{CALL} \frac{\text{monotone}(\Phi) \quad \{B\} \cup G \triangleright \mathbf{Body} : Sec_{A,R \Rightarrow S}(\Phi(\text{Fix}(\Phi)))}{G \triangleright \mathbf{Call} : B}$$

where  $B$  abbreviates  $Sec_{A,R \Rightarrow S}(\text{Fix}(\Phi))$ . All rules given in Figure 6 are monotone in  $\phi$  (and  $\psi$  where applicable).

## 5 Extension to objects

We now return to base-line non-interference but extend the language by object-related instructions.

**Syntax and semantics** Let  $c$  range over a set  $\mathcal{C}$  of class names and  $f$  over a set  $\mathcal{F}$  of field names. We extend the syntax of **IMP** by commands for allocating a fresh object and for accessing fields.

$$C \in \text{Com} ::= \dots \mid x := \mathbf{New} \ c \mid x := y.f \mid x.f := e$$

The dynamic semantics of the extended language is formulated using a category  $\mathcal{L}$  of locations, ranged over by  $\ell$ . References, values, objects, heaps, stores, and states are now as follows

$$\begin{aligned} r \in \mathcal{R} & ::= \text{Null} \mid \text{Loc} \ \ell & h \in \mathcal{H} & = \mathcal{L} \rightarrow_{\text{fin}} \mathcal{O} \\ v \in \mathcal{V} & ::= r \mid \text{IVal} \ i & s \in \mathcal{S} & = \mathcal{X} \rightarrow \mathcal{V} \\ o \in \mathcal{O} & = \mathcal{C} \times (\mathcal{F} \rightarrow_{\text{fin}} \mathcal{V}) & \sigma \in \Sigma & = \mathcal{S} \times \mathcal{H} \end{aligned}$$

Operational judgements remain of the form  $\sigma \xrightarrow{C} \tau$ , but  $C$ ,  $\sigma$  and  $\tau$  now refer to the extended notions. The semantics is given by the obvious extension of the rules from Figure 1, plus the three rules

$$\begin{aligned} \text{NEW} & \frac{\ell \notin \text{dom} \ h}{(s, h) \xrightarrow{x := \mathbf{New} \ c} (s[x \mapsto \text{Loc} \ \ell], h[\ell \mapsto (c, [])])} \\ \text{GET} & \frac{s(y) = \text{Loc} \ \ell \quad h(\ell) = (c, F) \quad F(f) = v}{(s, h) \xrightarrow{x := y.f} (s[x \mapsto v], h)} \\ \text{PUT} & \frac{s(x) = \text{Loc} \ \ell \quad h(\ell) = (c, F)}{(s, h) \xrightarrow{x.f := e} (s, h[\ell \mapsto (c, F[f \mapsto \llbracket e \rrbracket_s])])} \end{aligned}$$

Likewise, the axiomatic semantics of the purely imperative instructions is obtained from the earlier rules the obvious way, while the rules for the new instructions are shown in Figure 7. Soundness and completeness follow as before.

**Theorem 5.** (Soundness and completeness of program logic) The derivability of the judgement  $\emptyset \triangleright C : A$  is equivalent to  $\models C : A$ , i.e., its semantic validity.

A rule for context elimination is again easily seen to be derivable.

**Indistinguishability and security** Following Barthe and Rezk [8], the indistinguishability of states is defined parametrically in a partial bijection between location sets and is built up from similar relations on heaps and stores. We formalise partial bijections as sets  $\beta \subseteq \mathcal{L} \times \mathcal{L}$  such that  $(\ell, \ell_1) \in \beta$  and  $(\ell', \ell'_1) \in \beta$  implies  $(\ell = \ell') \Leftrightarrow (\ell_1 = \ell'_1)$ . We define  $\text{dom} \ \beta = \{\ell \mid \exists \ell'. (\ell, \ell') \in \beta\}$  and  $\text{cod} \ \beta = \{\ell \mid \exists \ell'. (\ell', \ell) \in \beta\}$ .

In order to define the indistinguishability relations for values, stores, objects, and heaps, we assume the existence of some context  $\Gamma$  that assigns security levels to variables (notation  $\Gamma(x)$ ) and to fields (notation  $\Gamma(f)$ ).

On values, indistinguishability requires locations to be related by  $\beta$  while other values are required to be identical.

On stores, we define  $s \sim_{\beta} t \equiv \forall x. \Gamma(x) = \text{low} \Rightarrow s(x) \sim_{\beta} t(x)$ .

On objects, we explicitly require the low fields declared by the two objects to be identical (no static typing is assumed). For field map  $F$ , we first define  $\text{lowdom} \ F = \text{dom} \ F \cap \{f \mid \Gamma(f) = \text{low}\}$ . Then object indistinguishability is given by

$$(c, F) \sim_{\beta} (d, G) \equiv c = d \wedge \text{lowdom} \ F = \text{lowdom} \ G \wedge \forall f \ v \ w. F(f) = v \Rightarrow G(f) = w \Rightarrow \Gamma(f) = \text{low} \Rightarrow v \sim_{\beta} w.$$

$$\begin{array}{c}
\text{NEW} \frac{}{G \triangleright x := \text{New } c : \lambda (s, h) \tau. \exists \ell \notin \text{dom } h. \tau = (s[x \mapsto \text{Loc } \ell], h[\ell \mapsto (c, [])])} \\
\text{GET} \frac{}{G \triangleright x := y.f : \lambda (s, h) \tau. \exists \ell c F v. s(y) = \text{Loc } \ell \wedge h(\ell) = (c, F) \wedge F(f) = v \wedge \tau = (s[x \mapsto v], h)} \\
\text{PUT} \frac{}{G \triangleright x.f := e : \lambda (s, h) \tau. \exists \ell c F. s(x) = \text{Loc } \ell \wedge h(\ell) = (c, F) \wedge \tau = (s, h[\ell \mapsto (c, F[f \mapsto \llbracket e \rrbracket_s])])}
\end{array}$$

**Figure 7. Axiomatic semantics of object-related instructions**

Finally, on heaps, we define

$$\begin{aligned}
h \sim_{\beta} k &\equiv \text{dom } \beta \subseteq \text{dom } h \wedge \text{cod } \beta \subseteq \text{dom } k \wedge \\
&\forall \ell \ell' v w \quad h(\ell) = v \Rightarrow k(\ell') = w \Rightarrow \\
&\beta(\ell) = \ell' \Rightarrow v \sim_{\beta} w.
\end{aligned}$$

and on states,  $(s, h) \sim_{\beta} (t, k) \equiv s \sim_{\beta} t \wedge h \sim_{\beta} k$ .

The definition of security requires for each  $\beta$  relating the initial states the existence of a partial bijection  $\gamma$  that relates the final states and extends  $\beta$ .

**Definition 5.** *Program  $C$  is secure, notation secure  $C$ , if whenever  $\sigma \xrightarrow{C} \tau$ ,  $\sigma' \xrightarrow{C} \tau'$ , and  $\sigma \sim_{\beta} \sigma'$  hold, then there exists some  $\gamma \supseteq \beta$  such that  $\tau \sim_{\gamma} \tau'$ .*

In the presence of objects, the intermediate formulae  $\phi$  now depend not only on the two states, but also on a partial bijection. Consequently, we let  $\mathcal{T}$  now denote the type  $(\Sigma \times \Sigma \times \text{PBij}) \rightarrow \mathcal{B}$  and define a variation of the operator  $\text{Sec}(\cdot)$  for relations  $\phi$  of this type as follows.

$$\begin{aligned}
\text{Sec}(\phi) &\equiv \\
&\lambda \sigma \tau. (\forall \rho \beta. \sigma \sim_{\beta} \rho \Rightarrow \phi(\sigma, \rho, \beta)) \wedge \\
&(\forall \rho \beta. \phi(\rho, \sigma, \beta) \Rightarrow (\exists \gamma. \rho \sim_{\gamma} \tau \wedge \beta \subseteq \gamma))
\end{aligned}$$

**Proposition 6.** *For any  $\phi$ ,  $\triangleright C : \text{Sec}(\phi)$  implies secure  $C$ . Conversely, secure  $C$  implies  $\triangleright C : \text{Sec}(\phi)$  for the property  $\phi \equiv \lambda (\rho, \tau, \beta). \exists \sigma. \sigma \xrightarrow{C} \rho \wedge \sigma \sim_{\beta} \tau$ .*

The proof is similar to the one of Proposition 1.

**Derived proof rules** Figure 8 shows the low proof rules for the language with object-related instructions. Regarding the high proof rules, an interpretation using the expected assertion  $\lambda \sigma \tau. \exists \beta. \sigma \sim_{\beta} \tau$  fails. The reason for this is that in the presence of dangling pointers, we are unable to prove indistinguishability of the values  $s(x) \sim_{\beta} t(x)$ , in particular in the case where  $s(x) = \text{Loc } \ell$  (even if  $s = t$ ). We therefore define the predicate  $\vdash \sigma$  which mandates the absence of dangling pointers in state  $\sigma$ .

$$\begin{aligned}
\vdash (s, h) &\equiv \\
&(\forall x \ell. s(x) = \text{Loc } \ell \Rightarrow \ell \in \text{dom } h) \wedge \\
&\forall \ell' c F f \ell. (h(\ell') = (c, F) \wedge F(f) = \text{Loc } \ell) \Rightarrow \\
&\ell \in \text{dom } h
\end{aligned}$$

In addition, we call expression  $e$  *good* if its evaluation in a state without dangling pointers does not result in a dangling pointer<sup>3</sup>:

$$\models e \equiv \forall s h. \vdash (s, h) \Rightarrow \forall \ell. \llbracket e \rrbracket_s = \text{Loc } \ell \Rightarrow \ell \in \text{dom } h$$

Given these definitions, the high proof rules shown in Figure 9 may be derived, i.e. the assertion satisfied by high commands is  $\lambda \sigma \tau. \vdash \sigma \Rightarrow (\vdash \tau \wedge \exists \beta. \sigma \sim_{\beta} \tau)$ . In all cases, the partial bijection whose existence is asserted is in fact given by the identity bijection on  $\text{dom}(\text{snd}(\sigma))$ .

Finally, the rule for procedure invocation is

$$\frac{\text{monotone}(\Phi) \quad \{\text{Sec}(\text{Fix}(\Phi))\} \cup G \triangleright \mathbf{Body} : \text{Sec}(\Phi(\text{Fix}(\Phi)))}{G \triangleright \mathbf{Call} : \text{Sec}(\text{Fix}(\Phi))}$$

where an operator  $\Phi : \mathcal{T} \Rightarrow \mathcal{T}$  over the (new) type  $\mathcal{T}$  is monotone if all  $\phi$  and  $\psi$  satisfy the implication

$$\begin{aligned}
&\text{If } (\forall \sigma \tau \beta. \phi(\sigma, \tau, \beta) \Rightarrow \psi(\sigma, \tau, \beta)) \\
&\text{then } (\forall \sigma \tau \beta. (\Phi\phi)(\sigma, \tau, \beta) \Rightarrow (\Phi\psi)(\sigma, \tau, \beta)).
\end{aligned}$$

**Justification of the notion of security** The definition of secure  $C$  (Definition 5) is only backed up by Proposition 6 and our ability to derive proof rules. As a more conceptual piece of evidence, we show that an attacker is unable to observe the content of low variables if he embeds  $C$  in an arbitrary program context  $P$ , as long as  $P$  *obviously*, (i.e. essentially syntactically) does not depend upon high-security variables.

We first define the syntax of contextual programs  $P$

$$\begin{aligned}
P ::= &[\cdot] \mid \mathbf{CSkip} \mid \mathbf{CAssign}(x, e) \mid \mathbf{CWhile } b \text{ do } P \\
&\mid \mathbf{CGet}(x, y, f) \mid \mathbf{CPut}(x, f, e) \mid \mathbf{CComp}(P, P) \\
&\mid \mathbf{CIf } b \text{ then } P \text{ else } P \mid \mathbf{CNew}(x, c) \mid \mathbf{CCall}.
\end{aligned}$$

**CBody** represents the contextual body of the unnamed procedure.

The substitution operation  $P[C]$  defined in Figure 10 (left) inserts a single command  $C$  into all holes in  $P$ . The

<sup>3</sup>We expect one could eliminate this condition if non-Null references (and operators performing pointer arithmetic) are eliminated from the syntax of expressions. However, our formalisation models operations as (arbitrary) meta-logical functions, hence the need for the condition  $\models e$ .

---


$$\begin{array}{c}
\text{SKIP} \frac{}{G \triangleright \mathbf{Skip} : \text{Sec}(\lambda(\sigma, \tau, \beta). \sigma \sim_{\beta} \tau)} \quad \text{ASSIGN} \frac{\forall \beta s t. s \sim_{\beta} t \Rightarrow \llbracket e \rrbracket_s \sim_{\beta} \llbracket e \rrbracket_t}{G \triangleright x := e : \text{Sec}(\lambda(\sigma, \tau, \beta). \exists s h. \sigma = (s[x \mapsto \llbracket e \rrbracket_s], h) \wedge (s, h) \sim_{\beta} \tau)} \\
\text{COMP} \frac{G \triangleright C_1 : \text{Sec}(\phi) \quad G \triangleright C_2 : \text{Sec}(\phi)}{G \triangleright C_1; C_2 : \text{Sec}(\lambda(\sigma, \tau, \beta). \exists \rho. \phi(\rho, \tau, \beta) \wedge (\forall \omega \delta. \rho \sim_{\delta} \omega \Rightarrow \psi(\sigma, \omega, \delta)))} \\
\text{IFF} \frac{\forall \beta s t. s \sim_{\beta} t \Rightarrow \llbracket b \rrbracket_s = \llbracket b \rrbracket_t \quad G \triangleright C_1 : \text{Sec}(\phi) \quad G \triangleright C_2 : \text{Sec}(\phi)}{G \triangleright \mathbf{If } b \mathbf{ then } C_1 \mathbf{ else } C_2 : \text{Sec}(\lambda(\sigma, (s, h), \beta). (\llbracket b \rrbracket_s \Rightarrow \phi(\sigma, (s, h), \beta)) \wedge ((\neg \llbracket b \rrbracket_s) \Rightarrow \psi(\sigma, (s, h), \beta)))} \\
\text{WHILE} \frac{\forall \beta s t. s \sim_{\beta} t \Rightarrow \llbracket b \rrbracket_s = \llbracket b \rrbracket_t \quad G \triangleright C : \text{Sec}(\phi)}{G \triangleright \mathbf{While } b \mathbf{ do } C : \text{Sec}(\lambda(\sigma, \tau, \beta). \text{var}(b, \phi, \beta, \sigma, \tau))} \\
\text{NEW} \frac{\Gamma(x) = \text{low}}{G \triangleright x := \mathbf{New } c : \text{Sec}(\lambda(\sigma, \tau, \beta). \exists \ell s h. \ell \notin \text{dom } h \wedge (s, h) \sim_{\beta} \tau \wedge \sigma = (s[x \mapsto \text{Loc } \ell], h[\ell \mapsto (c, [])]))} \\
\text{GET} \frac{\Gamma(y) = \text{low} \quad \Gamma(f) = \text{low}}{G \triangleright x := y.f : \text{Sec}(\lambda(\sigma, \tau, \beta). \exists \ell c F v s h. s(y) = \text{Loc } \ell \wedge h(\ell) = (c, F) \wedge F(f) = v \wedge \sigma = (s[x \mapsto v], h) \wedge (s, h) \sim_{\beta} \tau)} \\
\text{PUT} \frac{\Gamma(x) = \text{low} \quad \Gamma(f) = \text{low} \quad \forall \beta s t. s \sim_{\beta} t \Rightarrow \llbracket e \rrbracket_s \sim_{\beta} \llbracket e \rrbracket_t}{G \triangleright x.f := e : \text{Sec}(\lambda(\sigma, \tau, \beta). \exists \ell c F s h. s(x) = \text{Loc } \ell \wedge h(\ell) = (c, F) \wedge \sigma = (s, h[\ell \mapsto (c, F[f \mapsto \llbracket e \rrbracket_s])]) \wedge (s, h) \sim_{\beta} \tau)}
\end{array}$$

Relation  $\text{var}(b, \phi, \beta, \sigma, \tau)$  is defined by the rules

$$\frac{\neg \llbracket b \rrbracket_s \quad \sigma \sim_{\beta} (s, h)}{\text{var}(b, \phi, \beta, \sigma, (s, h))} \quad \frac{\llbracket b \rrbracket_s \quad \phi(\rho, (s, h), \beta) \quad \forall \omega \gamma. \rho \sim_{\gamma} \omega \Rightarrow \text{var}(b, \phi, \gamma, \sigma, \omega)}{\text{var}(b, \phi, \beta, \sigma, (s, h))}$$

and is monotone in  $\phi$ .

**Figure 8. Low proof rules for language with object-related instructions**

$$\begin{array}{c}
\text{H-ASS} \frac{\Gamma(x) = \text{high} \quad \models e}{G \triangleright x := e : \lambda \sigma \tau. \vdash \sigma \Rightarrow (\vdash \tau \wedge \exists \beta. \sigma \sim_{\beta} \tau)} \quad \text{H-NEW} \frac{\Gamma(x) = \text{high}}{G \triangleright x := \mathbf{New } c : \lambda \sigma \tau. \vdash \sigma \Rightarrow (\vdash \tau \wedge \exists \beta. \sigma \sim_{\beta} \tau)} \\
\text{H-GET} \frac{\Gamma(x) = \text{high}}{G \triangleright x := y.f : \lambda \sigma \tau. \vdash \sigma \Rightarrow (\vdash \tau \wedge \exists \beta. \sigma \sim_{\beta} \tau)} \quad \text{H-PUT} \frac{\Gamma(f) = \text{high} \quad \models e}{G \triangleright x.f := e : \lambda \sigma \tau. \vdash \sigma \Rightarrow (\vdash \tau \wedge \exists \beta. \sigma \sim_{\beta} \tau)}
\end{array}$$

**Figure 9. High proof rules for object-related instructions**

$$\begin{array}{l}
[\cdot][C] = C \\
\mathbf{CSkip}[C] = \mathbf{Skip} \\
\mathbf{CAssign}(x, e)[C] = x := e \\
\mathbf{CNew}(x, c)[C] = x := \mathbf{New } c \\
\mathbf{CGet}(x, y, f)[C] = x := y.f \\
\mathbf{CPut}(x, f, e)[C] = x.f := e \\
\mathbf{CComp}(P, Q)[C] = P[C]; Q[C] \\
\mathbf{CIf } b \mathbf{ then } P \mathbf{ else } Q[C] = \mathbf{If } b \mathbf{ then } P[C] \mathbf{ else } Q[C] \\
\mathbf{CWhile } b \mathbf{ do } P[C] = \mathbf{While } b \mathbf{ do } P[C] \\
\mathbf{CCall}[C] = \mathbf{Call}
\end{array}
\quad
\begin{array}{l}
\text{Vars}(\mathbf{Skip}) = \emptyset \\
\text{Vars}(x := e) = \text{fv}(e) \\
\text{Vars}(x := \mathbf{New } c) = \emptyset \\
\text{Vars}(x := y.f) = \{y\} \\
\text{Vars}(x.f := e) = \text{fv}(e) \\
\text{Vars}(C; D) = \text{Vars}(C) \cup \text{Vars}(D) \\
\text{Vars}(\mathbf{While } b \mathbf{ do } C) = \text{fv}(b) \cup \text{Vars}(C) \\
\text{Vars}(\mathbf{If } b \mathbf{ then } C \mathbf{ else } D) = \text{fv}(b) \cup \text{Vars}(C) \cup \text{Vars}(D) \\
\text{Vars}(\mathbf{Call}) = \emptyset
\end{array}$$

**Figure 10. Substitution of commands into contextual programs, and accessed variables**

$$\begin{array}{c}
\frac{\forall st \beta. s \sim_{\beta} t \Rightarrow \llbracket e \rrbracket_s \sim_{\beta} \llbracket e \rrbracket_t}{\vdash e} \quad \frac{\vdash e_1 \quad \vdash e_2 \quad \forall \beta v_1 w_1 v_2 w_2. v_1 \sim_{\beta} v_2 \Rightarrow w_1 \sim_{\beta} w_2 \Rightarrow v_1 \text{ bop } w_1 = v_2 \text{ bop } w_2}{\vdash e_1 \text{ bop } e_2} \\
\frac{}{X \vdash \mathbf{CSkip}} \quad \frac{fv(e) \subseteq X \quad \vdash e}{X \vdash \mathbf{CAssign}(x, e)} \quad \frac{}{X \vdash \mathbf{CNew}(x, e)} \quad \frac{y \in X \quad \Gamma(f) = \text{low}}{X \vdash \mathbf{CGet}(x, y, f)} \quad \frac{fv(e) \subseteq X \quad \Gamma(x) = \text{low} \quad \vdash e}{X \vdash \mathbf{CPut}(x, f, e)} \\
\frac{}{X \vdash [\cdot]} \quad \frac{X \vdash P \quad X \vdash Q}{X \vdash \mathbf{CComp}(P, Q)} \quad \frac{fv(b) \subseteq X \quad X \vdash P \quad X \vdash Q \quad \vdash b}{X \vdash \mathbf{CIf } b \text{ then } P \text{ else } Q} \quad \frac{fv(b) \subseteq X \quad X \vdash P \quad \vdash b}{X \vdash \mathbf{CWhile } b \text{ do } P} \quad \frac{}{X \vdash \mathbf{CCall}}
\end{array}$$

**Figure 11. Expressions and contexts that only access low variables**

variables accessed by expressions and commands are defined in Figure 10 (right). In Figure 11, we define typing relations characterising expressions and contexts that depend solely on low variables. Writing  $Low_X(P)$  if  $X \vdash P$  and all  $x \in X$  satisfy  $\Gamma(x) = \text{low}$ , one may then prove by induction on the derivation height of the operational semantics the following result.

**Lemma 2.** *Let secure  $C$ ,  $Low_X(P)$ ,  $Low_X(\mathbf{CBody})$  and  $\mathbf{Body} = \mathbf{CBody}[C]$ . Then secure  $P[C]$ .*

In this sense, property *secure  $C$*  is contextually closed. Thus, a variable *res* representing the readable output result of an attacking program satisfies the following.

**Theorem 6.** *Let  $C$  and  $P$  be such that secure  $C$ ,  $Low_X(P)$ , and  $Low_X(\mathbf{CBody})$  hold, and let  $\sigma \sim_{\beta} \sigma'$ ,  $\sigma \xrightarrow{P[C]} (t, h)$  and  $\sigma' \xrightarrow{P[C]} (t', h')$ . Then, for  $\mathbf{Body} = \mathbf{CBody}[C]$  and  $\Gamma(\text{res}) = \text{low}$ , there is a  $\gamma \supseteq \beta$  such that  $t(\text{res}) \sim_{\gamma} t'(\text{res})$ .*

In particular, if *res* is an integer variable, the observed values  $t(\text{res})$  and  $t'(\text{res})$  will be identical.

## 6 Discussion

We presented a novel approach to the verification of language-based information flow security, the embedding into program logics using the factoring by operator  $Sec(\phi)$ . This approach complements self-composition, the development of special-purpose logics, and the encoding in relational program logics. In addition to proving the semantic equivalence of the factored formulae and the original notions of security, we showed that the factored formats admit the interpretation of appropriate type systems, in a rule-by-rule fashion. This feature makes our approach well-suited to proof-carrying code scenarios [30], as it provides a systematic way to generate proofs from program analyses. Under this perspective, program logics emerge as a *lingua franca* in which different static analyses may be interpreted, compared, and combined.

As code transmission in PCC scenarios typically targets low-level code, we intend to transfer our findings to

program logics for (virtual) machine code. Previous work concerned with information flow security for low-level languages includes [27, 8, 14]. We also intend to generalise our approach to languages with more general method invocation mechanisms.

It is evident that the suitability of our approach for further notions of information flow security depends on the choice of operational semantics and program logic. In the present paper, we have considered a base case, namely a standard big-step operational semantics and a partial-correctness program logic. As this logic is invariant under program transformations that are denotationally semantics-preserving, the logic is not suited for the encoding of security notions that distinguish extensionally identical programs. This requirement strictly subsumes the *semantic consistency* condition in Sabelfeld and Sands' "classification of declassification" survey [35]. Their condition merely requires invariance under transformations on declassification-free programs. In particular, our approach does not support the interpretation of the type systems by Li and Zdancewic for *relaxed non-interference* [25] and Sabelfeld and Myers for *delimited information release* [34]. On the other hand, Banerjee and Naumann [6] demonstrate how a variety of extensional declassification policies can be directly represented as pre- and post-conditions in a Hoare logic with a relational interpretation. Considering an object oriented language, their soundness condition is more involved than ours, as it includes disjointness assertions akin to those in Separation Logic. Structurally, however, the soundness condition appears to be similar to what we called  $A; R \Rightarrow S$ -security: it has a relational and a sequential component, namely the non-interference property and a frame condition. It thus seems likely that the proof system presented in [6], as well as the similar system by Amtoft et al. [2] which Banerjee and Naumann's work extends, would be derivable in our program logic for heap structures. In fact, Hunt and Sands have already shown that derivability in the restriction of Amtoft et al.'s system to **IMP** (reported in [3]) is equivalent to the derivability of a certain judgement in the type system of [19] for the lattice given by subsets of program variables.

In [4], it is shown how a logic that guarantees termination can be built “on top of” a partial-correctness logic, by including hypotheses from the latter logic in appropriate rules. Future work will seek to establish whether such a termination logic would be suitable for the interpretation of termination-sensitive notions of information flow, i.e. if the termination precondition guarantees co-termination.

Further notions of secure information flow may be encodeable in program logics that include strong invariants (assertions that are satisfied by all intermediate states, be the computation terminating or not) [17, 10] or are based on operational semantics that expose intensional behaviour [5].

In all of these cases, the main challenge will consist of finding factorisations of the appropriate security notions that admit the encoding of type systems. Compared to soundness statements that refer directly to operational judgements, the use of a program logic allows one to treat issues such as the verification of recursive program structures uniformly for several program analyses.

As an alternative to Hoare logic, dynamic logics may be used to express interpretations of type systems. Indeed, Hähnle et al. [18] recently presented an encoding of Hunt and Sands’ type system in a formalism called *dynamic logic with updates*. The authors demonstrate how the expressiveness of logic complements the efficiency of type systems, and outline how the approach may be used to analyse exceptional behaviour. On the other hand, heap structures and procedures are not covered.

Like the work of Hähnle et al., our encoding decouples the generation of proofs (type inference) from proof checking. Future work will seek to develop concrete certificate formats, analyse the growth of type-interpreting formulae  $Sec(\phi)$ , and aim to combine the precision of program logics with the effectiveness of type systems.

Finally, we would like to point out that our findings do not contradict McLean’s result regarding the inexpressibility of non-interference as a 1-safety property [26]. In fact, our language based approach is conceptually different from the Alpern-Schneider framework considered by McLean: programs in our language represent concrete deterministic systems, and do not support an interpretation of refinement as trace set inclusion. In particular, the “totally unrefined” system that admits arbitrary behaviour would in our setting be a program with trivial operational behaviour  $\forall \sigma \tau. \sigma \xrightarrow{C} \tau$ . Such a program does *not* satisfy non-interference, in contrast to the critical usage of the totally unrefined system McLean makes in his argument. In particular, McLean’s result refers to a single property that has to be satisfied uniformly by all secure systems, while our approach aims to verify (or disprove) non-interference of a single concrete program, and the formula  $\phi$  witnessing the security of program  $C$  may even depend upon  $C$ . Nevertheless, a future extension of our work to non-deterministic

(under-specified) systems would be desirable in order to support the refinement-based development of secure systems.

**Acknowledgements** This work was supported in part by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project, and the DFG-funded project InfoZert, grant numbers Be 3712/2-1 and Ho 1911/7-1. We are grateful to the members of both projects for discussions on information flow and program logics, and on earlier versions. The feedback from the reviewers is also gratefully acknowledged. Andrei Sabelfeld pointed us to the paper by Hunt and Sands.

## References

- [1] m Darvas, R. Hhnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In D. Hutter and M. Ullmann, editors, *Proc. 2nd International Conference on Security in Pervasive Computing (SPC’05)*, volume 3450 of *LNCS*, pages 193–209. Springer, 2005.
- [2] T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In Morrisett and Peyton Jones [28], pages 91–102.
- [3] T. Amtoft and A. Banerjee. Information flow analysis in logical form. In R. Giacobazzi, editor, *Proceedings of the 11th International Symposium on Static Analysis (SAS ’04)*, volume 3148 of *LNCS*, pages 100–115. Springer, 2004.
- [4] D. Aspinall, L. Beringer, M. Hofmann, H.-W. Loidl, and A. Momigliano. A program logic for resources. *Theoretical Computer Science*, 2006. To appear.
- [5] D. Aspinall, L. Beringer, and A. Momigliano. Optimisation validation. In J. Knoop, G. C. Necula, and W. Zimmermann, editors, *Proceedings of the 5th International Workshop on Compiler Optimization meets Compiler Verification (COCV 2006)*, ENTCS. Elsevier, 2006. To appear.
- [6] A. Banerjee and D. Naumann. A logical account of secure declassification. Submitted. Available from D. Naumann’s web page, Feb. 2006.
- [7] G. Barthe, P. R. D’Argenio, and T. Rezk. Secure information flow by self-composition. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW’04)*, pages 100–114. IEEE Computer Society Press, 2004.
- [8] G. Barthe and T. Rezk. Non-interference for a JVM-like language. In J. G. Morrisett and M. Fahndrich, editors, *Proceedings of TLDI’05: 2005 ACM International Workshop on Types in Languages Design and Implementation*, pages 103–112. ACM Press, 2005.
- [9] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In Jones and Leroy [22], pages 14–25.
- [10] L. Beringer and M. Hofmann. A bytecode logic for JML and types. In N. Kobayashi, editor, *Proceedings of the Fourth*

- ASIAN Symposium on Programming Languages and Systems (APLAS 2006)*, volume 4279 of *LNCS*, pages 389 – 405. Springer, 2006.
- [11] L. Beringer and M. Hofmann. Secure information flow and program logics – Isabelle sources. `beringer/CSF2007Sources.tar.gz`, 2007.
- [12] L. Beringer, M. Hofmann, A. Momigliano, and O. Shkaravska. Automatic certification of heap consumption. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Proceedings*, volume 3452 of *LNCS*, pages 347–362. Springer, 2004.
- [13] D. Clark, C. Hankin, and S. Hunt. Information flow for Algol-like languages. *Computer Languages*, 28(1), 2002.
- [14] S. Genaim and F. Spoto. Information Flow Analysis for Java Bytecode. In R. Cousot, editor, *Proceedings of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *LNCS*, pages 346–362. Springer, Jan. 2005.
- [15] R. Giacobazzi and I. Mastroeni. Abstract non-interference: parameterizing non-interference by abstract interpretation. In Jones and Leroy [22], pages 186–197.
- [16] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.
- [17] R. Hähnle and W. Mostowski. Verification of safety properties in the presence of transactions. In G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, and T. Muntean, editors, *Proceedings, Construction and Analysis of Safe, Secure and Interoperable Smart devices (CASSIS'04) Workshop*, volume 3362 of *LNCS*, pages 151–171. Springer, 2005.
- [18] R. Hähnle, J. Pan, P. Rümmer, and D. Walter. Integration of a security type system into a program logic. In *Proceedings of the 2nd Symposium on Trustworthy Global Computing*, *LNCS*. Springer, 2006. To appear.
- [19] S. Hunt and D. Sands. On flow-sensitive security types. In Morrisett and Peyton Jones [28], pages 79–90.
- [20] B. Jacobs, W. Pieters, and M. Warnier. Statically checking confidentiality via dynamic labels. In *Proceedings of the 2005 Workshop on Issues in the theory of security (WITS'05)*, pages 50–56. ACM Press, 2005.
- [21] C. Jones. *Systematic Software Development using VDM*. Prentice Hall, 1990.
- [22] N. D. Jones and X. Leroy, editors. *Proceedings of the 31st ACM Symposium on Principles of Programming Languages (POPL 2004)*. ACM Press, 2004.
- [23] R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113 – 138, 2000.
- [24] T. Kleymann. *Hoare Logic and VDM: Machine-Checked Soundness and Completeness Proofs*. PhD thesis, LFCS, University of Edinburgh, Sept. 1998. Technical Report ECS-LFCS-98-392.
- [25] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proceedings of the 32nd ACM Symposium on Principles of Programming Languages (POPL 2005)*, pages 158–170, New York, NY, USA, 2005. ACM Press.
- [26] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *SP '94: Proceedings of the 1994 IEEE Symposium on Security and Privacy*, pages 79–93. IEEE Computer Society, 1994.
- [27] R. Medel, A. B. Compagnoni, and E. Bonelli. A typed assembly language for non-interference. In M. Coppo, E. Lodi, and G. M. Pinna, editors, *Proceedings of the 9th Italian Conference on Theoretical Computer Science (ICTCS 2005)*, volume 3701 of *LNCS*, pages 360–374. Springer, 2005.
- [28] J. G. Morrisett and S. L. Peyton Jones, editors. *Proceedings of the 33rd ACM Symposium on Principles of Programming Languages (POPL 2006)*. ACM Press, 2006.
- [29] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004)*, pages 172–186. IEEE Computer Society, 2004.
- [30] G. C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL 1997)*, pages 106–119. ACM Press, 1997.
- [31] T. Nipkow. Hoare logics for recursive procedures and unbounded nondeterminism. In J. Bradfield, editor, *Proceedings of Computer Science Logic (CSL 2002)*, volume 2471 of *LNCS*, pages 103–119. Springer, 2002.
- [32] F. Pottier and V. Simonet. Information flow inference for ML. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL 2002)*, pages 319–330. ACM Press, 2002.
- [33] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications – special issue on Formal Methods for Security*, 21(1):5 – 19, Jan. 2003.
- [34] A. Sabelfeld and A. C. Myers. A model for delimited information release. In K. Futatsugi, F. Mizoguchi, and N. Yonezaki, editors, *Software Security - Theories and Systems, Second Mext-NSF-JSPS International Symposium (ISSS 2003)*, volume 3233 of *LNCS*, pages 174–191. Springer, 2004.
- [35] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 255–269. IEEE Computer Society, 2005.
- [36] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In C. Hankin and I. Siveroni, editors, *Proceedings of the 12th International Symposium on Static Analysis (SAS '05)*, volume 3672 of *LNCS*, pages 352–367. Springer, 2005.
- [37] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *J. Computer Security*, 4(3):167–187, 1996.
- [38] G. Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, 1993.