Possibilistic Information Flow is safe for Probabilistic Non-Interference

(Extended Abstract)

David Clark[1], Chris Hankin[1], Sebastian Hunt[2], and Rajagopal Nagarajan[1]

1 Introduction

Recently there has been considerable interest in analysing programs for the existence of probabilistic covert channels. Sabelfeld and Sands [6] have given a semantic characterisation of probabilistic non-interference and used it to give elegant correctness proofs for type-based systems proposed by Volpano and Smith [7]. Our thesis in this paper is that one can use a possibilistic information flow analysis to check for probabilistic interference—the absence of information flow implies the absence of interference. We apply our techniques to the analysis of a Probabilistic Idealised Algol. Our analysis is based on a control flow analysis which is used to construct an analogue of the the higher-order flowcharts of Malacaria and Hankin [3]; the information flows can be read fairly directly from the flowchart.

In the next section we introduce the syntax and semantics of the programming language. We then present an appropriate definition of probabilistic non-interference for this language. Section 4 presents the three stages of our analysis and we conclude with the statement of correctness.

2 Probabilistic Algol

In this paper, we consider a Probabilistic Idealised Algol (PA) [1]. Idealized Algol is a language developed by Reynolds [5] which extends the simply-typed λ-calculus with imperative features. The result is a prototype language combining block structured imperative programming and elements of higher-order functional programming.

The basic phrase types of PA are

$$B ::= \texttt{Exp}[X] \ \mid \ \texttt{Var}[X] \ \mid \ \texttt{com}$$

where X is a basic data type `integer` or `boolean`. General types are constructed in the usual way:

$$T ::= B \ \mid \ T \to T$$

The syntax of the language is as follows:

[1]{djc, clh, raja}@doc.ic.ac.uk, Dept of Computing, Imperial College, London.
[2]seb@soi.city.ac.uk, Dept of Computing, City University, London.

$$
\begin{array}{rcl}
 & & x \in \mathit{Var} \qquad c \in \mathit{Const} \qquad e \in \mathit{Exp} \\
e & ::= & x \mid c \mid e_1\, e_2 \mid \lambda x.e \mid \\
 & & \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid e_1 ; e_2 \mid \\
 & & x := e \mid \texttt{deref}\ x \mid \texttt{while}\ e_0\ \texttt{do}\ e_1 \mid \texttt{coin} \mid \\
 & & \texttt{new}\ x.e
\end{array}
$$

where c are constants of basic type (as is `coin`) and the other constants are of type:

$$
\begin{array}{l}
\texttt{if} : \texttt{Exp[boolean]} \to T \to T \to T \\
; : \texttt{com} \to \texttt{com} \to \texttt{com} \\
:= : (\texttt{Var}[X] \to \texttt{com}) \to \texttt{com} \\
\texttt{deref} : \texttt{Var}[X] \to \texttt{Exp}[X] \\
\texttt{while} : \texttt{Exp[boolean]} \to \texttt{com} \to \texttt{com} \\
\texttt{new} : (\texttt{Var}[X] \to \texttt{com}) \to \texttt{com}
\end{array}
$$

For the current paper we disallow lambda-bound variables of type $\texttt{Var}[X]$, though we believe that this restriction can be removed if the initial control flow analysis (see Sect. 4) is adapted to take account of variable aliasing.

We now present the structural operational semantics of PA, adapting the semantics for Fun in [4]. A function definition evaluates to a closure containing the syntax of the function definition together with an environment mapping its free variables to their values.

$$
\begin{array}{rcll}
v & \in & \mathbf{Val} & \text{values} \\
\rho & \in & \mathbf{Env} & \text{environments} \\
d & \in & \mathbf{DVal} & \text{basic data values} \\
\iota & \in & \mathbf{Loc} & \text{locations} \\
\sigma & \in & \mathbf{St} & \text{stores}
\end{array}
$$

defined by:

$$
\begin{array}{rcl}
v & ::= & d \mid \texttt{close}\ e\ \texttt{in}\ \rho \mid \texttt{void} \mid \iota \\
\rho & ::= & [\,] \mid \rho[x \mapsto v] \\
\sigma & ::= & [\,] \mid \sigma[\iota \mapsto d]
\end{array}
$$

A function abstraction $\lambda x.e_0$ will then evaluate to a closure, written $\texttt{close}\ (\lambda x.e_0)\ \texttt{in}\ \rho$. Our definitions do not demand that all terms e occurring in some $\texttt{close}\ e\ \texttt{in}\ \rho$ in the semantics will be of the form $\lambda x.e_0$; however, it will be the case for the semantics presented below.

We shall need intermediate configurations to handle the binding of local variables. We therefore introduce a syntactic category for *intermediate expressions*:

$$
ie \in \mathbf{IExp} \quad \text{intermediate expressions}
$$

that extends the syntax of expressions as follows:

$$
\begin{array}{rcl}
ie & ::= & x \mid c \mid ie_1\, ie_2 \mid \lambda x.e \mid \\
 & & \texttt{if}\ e_0\ \texttt{then}\ ie_1\ \texttt{else}\ e_2 \mid ie_1 ; ie_2 \mid \\
 & & x := ie \mid \texttt{deref}\ x \mid \texttt{while}\ ie_0\ \texttt{do}\ e_1 \mid \texttt{coin} \mid \\
 & & \texttt{new}\ x.e \mid \\
 & & \texttt{bind}\ \rho\ \texttt{in}\ ie \mid \texttt{close}\ e\ \texttt{in}\ \rho
\end{array}
$$

The intermediate expression $\texttt{bind}\ \rho\ \texttt{in}\ ie$ records that the ie has to be evaluated in an environment with the bindings ρ. (The sequence of environments of nested `bind`-constructs may be viewed as an encoding of the frames of a run-time stack.) So while $\texttt{close}\ e\ \texttt{in}\ \rho$ is a fully evaluated value this is not the case for $\texttt{bind}\ \rho\ \texttt{in}\ ie$.

$$\rho \vdash \langle x, \sigma \rangle \rightarrow_1 \langle v, \sigma \rangle \quad \text{if } x \in dom(\rho) \text{ and } v = \rho(x)$$

$$\rho \vdash \langle (\lambda x.e_0), \sigma \rangle \rightarrow_1 \langle (\texttt{close } (\lambda x.e_0) \texttt{ in } \rho_0), \sigma \rangle$$
$$\text{where } \rho_0 = \rho \mid FV(\lambda x.e_0)$$

$$\frac{\rho \vdash \langle ie_1, \sigma \rangle \rightarrow_p \langle ie_1', \sigma' \rangle}{\rho \vdash \langle (ie_1\ ie_2), \sigma \rangle \rightarrow_p \langle (ie_1'\ ie_2), \sigma' \rangle}$$

$$\frac{\rho \vdash \langle ie_2, \sigma \rangle \rightarrow_p \langle ie_2', \sigma' \rangle}{\rho \vdash \langle (v_1\ ie_2), \sigma \rangle \rightarrow_p \langle (v_1\ ie_2'), \sigma' \rangle}$$

$$\rho \vdash \langle ((\texttt{close } (\lambda x.e_1) \texttt{ in } \rho_1)\ v), \sigma \rangle \ \rightarrow_1$$
$$\langle (\texttt{bind } \rho_1[x \mapsto v] \texttt{ in } e_1), \sigma \rangle$$

$$\frac{\rho_1 \vdash \langle ie_1, \sigma \rangle \rightarrow_p \langle ie_1', \sigma' \rangle}{\rho \vdash \langle (\texttt{bind } \rho_1 \texttt{ in } ie_1), \sigma \rangle \rightarrow_p \langle (\texttt{bind } \rho_1 \texttt{ in } ie_1'), \sigma' \rangle}$$

$$\rho \vdash \langle (\texttt{bind } \rho_1 \texttt{ in } v_1), \sigma \rangle \rightarrow_1 \langle v_1, \sigma \rangle$$

Table 1: The Structural Operational Semantics (part 1).

$$\frac{\rho \vdash \langle ie_0, \sigma \rangle \rightarrow_p \langle ie_0', \sigma' \rangle}{\rho \vdash \langle (\texttt{if } ie_0 \texttt{ then } e_1 \texttt{ else } e_2), \sigma \rangle \rightarrow_p \langle (\texttt{if } ie_0' \texttt{ then } e_1 \texttt{ else } e_2), \sigma' \rangle}$$

$$\rho \vdash \langle (\texttt{if true then } t_1 \texttt{ else } t_2), \sigma \rangle \rightarrow_1 \langle t_1, \sigma \rangle$$

$$\rho \vdash \langle (\texttt{if false then } t_1 \texttt{ else } t_2), \sigma \rangle \rightarrow_1 \langle t_2, \sigma \rangle$$

$$\frac{\rho \vdash \langle ie_1, \sigma \rangle \rightarrow_p \langle ie_1', \sigma' \rangle}{\rho \vdash \langle ie_1; ie_2, \sigma \rangle \rightarrow_p \langle ie_1'; ie_2, \sigma' \rangle}$$

$$\rho \vdash \langle \texttt{void}; ie_2, \sigma \rangle \rightarrow_1 \langle ie_2, \sigma \rangle$$

$$\frac{\rho \vdash \langle ie, \sigma \rangle \rightarrow_p \langle ie', \sigma' \rangle}{\rho \vdash \langle x := ie, \sigma \rangle \rightarrow_p \langle x := ie', \sigma' \rangle}$$

$$\rho \vdash \langle x := d, \sigma \rangle \rightarrow_1 \langle \texttt{void}, \sigma[\rho(x) \mapsto d] \rangle$$

$$\rho \vdash \langle \texttt{deref } x, \sigma \rangle \rightarrow_1 \langle \sigma(\rho(x)), \sigma \rangle$$

$$\vdots$$

$$\rho \vdash \langle \texttt{coin}, \sigma \rangle \rightarrow_{0.5} \langle \texttt{true}, \sigma \rangle$$

$$\rho \vdash \langle \texttt{coin}, \sigma \rangle \rightarrow_{0.5} \langle \texttt{false}, \sigma \rangle$$

$$\rho \vdash \langle \texttt{new } x.e, \sigma \rangle \rightarrow_1 \langle (\texttt{bind } \rho[x \mapsto \iota] \texttt{ in } e), \sigma \rangle$$
$$\text{where } \iota \notin \text{ran}(\rho)$$

Table 2: The Structural Operational Semantics (part 2).

3 Security

We are concerned here with trojan horse confidentiality attacks. Suppose that H and L are two sets of variable names. H is the set of high-security inputs, which we wish to keep secret from the attacker (who supplies the program), and L is the set of low-security outputs, which all users (including the attacker) can read when the program terminates. We need to verify that the final values of variables in L give no information about the initial values of variables in H.

The particular security property we are concerned with is the form of probablistic non-interference defined by Sands and Sabelfeld [6]. Since the analysis we describe is global and non-compositional, it suffices to give a direct specification of the property without invoking the machinery of probabilistic power domains.

We consider a *program* to be an open PA term of type `com` in which all free variables are of type $\mathtt{Var}[X]$. Given such a term e, let ρ be an injective environment such that $\mathrm{dom}(\rho) = \mathrm{FV}(e)$ and let σ be a store such that $\mathrm{ran}(\rho) \subseteq \mathrm{dom}(\sigma)$. We say that ρ, σ are *legal for* e. Then a *computation starting in* σ *and ending in* σ' is a derivation sequence $\rho \vdash \langle e, \sigma\rangle \rightarrow_{p_1} \cdots \rightarrow_{p_n} \langle \mathtt{void}, \sigma'\rangle$. Let $\mathrm{comp}_{\rho,e}(\sigma, \sigma')$ denote the set[3] of all such computations. Given $s \in \mathrm{comp}_{\rho,e}(\sigma,\sigma')$, let πs be the product of the p_i along s (set $\pi(\epsilon) = 1$). Now each program and legal environment determines a map from legal stores to sub-probability measures on stores (equivalently, distributions on $\mathbf{St}_\perp$) thus:

$$[\![e]\!]_\rho \sigma\sigma' = \sum_{s \in \mathrm{comp}_{\rho,e}(\sigma,\sigma')} \pi s$$

Recall that we are assuming that the attacker only has access to the final values of variables in L. As a result, all stores assigning the same values to variables in L will appear identical to the attacker. Given ρ and some set X of variable names, let $=_X$ be the equivalence relation on stores given by $\rho \vdash \sigma_0 =_X \sigma_1$ iff $\sigma_0(\rho(x)) = \sigma_1(\rho(x))$ for all $x \in X$. When defining our non-interference property we have to sum probabilities over the equivalence classes of $=_L$.

Definition 1 (Probabilistic non-interference) *A program e has the non-interference property with respect to H and L if, for all legal ρ, σ_0, σ_1, for all $=_L$ equivalence classes S:*

$$\rho \vdash \sigma_0 =_{\overline{H}} \sigma_1 \Rightarrow \sum_{\sigma' \in S} [\![e]\!]_\rho \sigma_0 \sigma' = \sum_{\sigma' \in S} [\![e]\!]_\rho \sigma_1 \sigma'$$

where $\overline{H}$ is the set complement of H.

4 An Information Flow Analysis

Following Denning [2] we consider two classes of information flow:

Explicit There are two kinds of explicit flow. *Direct flows* arise from variables involved in a simple assignment statement such as $x := y + z$. Here there is an information flow from y and z to x. *Indirect flows* are transitive flows arising from sequences of assignments. For example $y := x; \quad \cdots \quad z := y$. Here there is an indirect explicit flow from x to z via y.

Implicit There are also two types of implicit flow (though Denning only considered the first type). *Local flows* arise from guards in conditionals:

$$\mathtt{if}\ \ x\ \ \mathtt{then}\ \ y := z\ \ \mathtt{else}\ \ y := w.$$

[3] Note that the SOS rules have the property that no derivation sequence can be derived in more than one way.

Here there is an information flow from x to y, in addition to the explicit flows from z and w. *Global flows* arise from guards in while loops

$$x := y; (\texttt{while} \quad w \quad \texttt{do} \quad x := z); \cdots .$$

Here there is a global implicit flow from w to all variables in subsequent program points.

In this section we work with a labelled variant of the language in which all expressions are labelled with a label from some label set **Lab**.

We consider a three phase analysis. The first phase consists of a control flow analysis which associates a set of closures with each label—the set of possible functions which the expression might evaluate to. One suitable analysis for this first phase would be a slight adaptation of the 0-CFA in Nielson, Nielson and Hankin [4]. We then use the results of the CFA to construct a set of flows. In the third phase we read off the information flows from the set of flows constructed in the previous phase.

The result of the 0-CFA analysis is a pair $(\widehat{\mathsf{C}}, \widehat{\rho})$ where:

- $\widehat{\mathsf{C}}$ is the abstract cache associating abstract values with each labelled program point.
- $\widehat{\rho}$ is the abstract environment associating abstract values with each variable.

This is made precise by:

$$\begin{array}{lllll}
\widehat{v} & \in & \widehat{\mathbf{Val}} & = & \mathcal{P}(\mathbf{Term}) & \text{abstract values} \\
\widehat{\rho} & \in & \widehat{\mathbf{Env}} & = & \mathbf{Var} \to \widehat{\mathbf{Val}} & \text{abstract environments} \\
\widehat{\mathsf{C}} & \in & \widehat{\mathbf{Cache}} & = & \mathbf{Lab} \to \widehat{\mathbf{Val}} & \text{abstract caches}
\end{array}$$

For the formulation of the abstract 0-CFA analysis they write

$$(\widehat{\mathsf{C}}, \widehat{\rho}) \models e$$

for when $(\widehat{\mathsf{C}}, \widehat{\rho})$ is an acceptable Control Flow Analysis of the expression e.

We next construct a flow function:

$$\widehat{\mathsf{F}} : \mathbf{Exp} \times \widehat{\mathbf{Cache}} \to \mathcal{P}(\mathbf{Lab}^*)$$

This is shown in Table 3; $\oplus$ is pointwise concatenation, ϵ is the empty sequence.

Essentially $\widehat{\mathsf{F}}$ collects labels from assignments, conditionals and the predicates of while loops. The result is a collection of "paths" which correspond to the 1-valid paths of Malacaria and Hankin [3]. To find the information flows for a particular assignment, we consider all prefixes of the associated label and apply a procedure described below. We assume that `new` bound variables are pairwise distinct and distinct from all global variables.

The result of the flow analysis $\widehat{\mathsf{F}}(\mathit{prog})$ is a set of label sequences of the form $l_1 \ldots l_i \ldots l_n$. Let us say we are interested in the information flows through the program to the value taken by some variable x at some program point l_i. We examine each label, going backwards, in the prefix of the string ending in l_i. This traversal of prefixes accounts for indirect flows of information within the program. For each label we look at the corresponding statement of the program marked by that label. We then build a "dependency" graph using a limited form of dynamic transitive closure for each variable in the body of the statement. This procedure is not a full transitive closure as we need not close up over multiple assignments to the same variable; we can stop at the first such. When we build this graph, the labels could take us through a variety of statements. If the statement is an assignment, the links are quite straightforward, with each link connecting the variables in the assignment expression to the assigned variable. These correspond to direct flows. In case of a conditional expression we need to connect the guard also to the variables in the body of the commands. For while loops, we connect the guards to all variables in subsequent program points. These take care of the case of local and global implicit flows.

$$
\begin{array}{lcl}
\widehat{\mathsf{F}}(x^\ell, \widehat{\mathsf{C}}) & = & \{\epsilon\} \\
\widehat{\mathsf{F}}(c^\ell, \widehat{\mathsf{C}}) & = & \{\epsilon\} \\
\widehat{\mathsf{F}}((\lambda x.e)^\ell, \widehat{\mathsf{C}}) & = & \{\epsilon\} \\
\widehat{\mathsf{F}}((\texttt{if } t_0^{\ell_0} \texttt{ then } e_1 \texttt{ else } e_2)^\ell, \widehat{\mathsf{C}}) & = & \widehat{\mathsf{F}}(t_0^{\ell_0}, \widehat{\mathsf{C}}) \oplus \{\ell_0\} \oplus (\widehat{\mathsf{F}}(e_1, \widehat{\mathsf{C}}) \cup \widehat{\mathsf{F}}(e_2, \widehat{\mathsf{C}})) \oplus \{\ell\} \\
\widehat{\mathsf{F}}((e_1; e_2)^\ell, \widehat{\mathsf{C}}) & = & \widehat{\mathsf{F}}(e_1, \widehat{\mathsf{C}}) \oplus \widehat{\mathsf{F}}(e_2, \widehat{\mathsf{C}}) \\
\widehat{\mathsf{F}}((x := e)^\ell, \widehat{\mathsf{C}}) & = & \widehat{\mathsf{F}}(e, \widehat{\mathsf{C}}) \oplus \{\ell\} \\
\widehat{\mathsf{F}}((\texttt{deref } x)^\ell, \widehat{\mathsf{C}}) & = & \{\epsilon\} \\
\widehat{\mathsf{F}}((\texttt{while } t_0^{\ell_0} \texttt{ do } e_1)^\ell, \widehat{\mathsf{C}}) & = & \widehat{\mathsf{F}}(t_0^{\ell_0}, \widehat{\mathsf{C}}) \oplus \{\ell_0\} \oplus \widehat{\mathsf{F}}(e_1, \widehat{\mathsf{C}}) \oplus \{\ell_0\} \\
\widehat{\mathsf{F}}((\texttt{coin})^\ell, \widehat{\mathsf{C}}) & = & \{\epsilon\} \\
\widehat{\mathsf{F}}((\texttt{new } x.e)^\ell, \widehat{\mathsf{C}}) & = & \widehat{\mathsf{F}}(e, \widehat{\mathsf{C}}) \\
\widehat{\mathsf{F}}((t_1^{\ell_1}\ t_2^{\ell_2})^\ell) & = & \widehat{\mathsf{F}}(t_1^{\ell_1}, \widehat{\mathsf{C}}) \oplus \widehat{\mathsf{F}}(t_2^{\ell_2}, \widehat{\mathsf{C}}) \oplus \bigcup_{(\lambda x.t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1)} \widehat{\mathsf{F}}(t_0^{\ell_0}, \widehat{\mathsf{C}})
\end{array}
$$

Table 3: The flow analysis.

5 Correctness

Let $\mathrm{IF}_\ell(x)$ be the information flows (a set of variable names) computed by the analysis for x at ℓ. Let $\mathrm{exits}(prog)$ be the set of all labels which appear at the end of some sequence $l_1 \dots l_i \dots l_n$ in $\widehat{\mathsf{F}}(prog)$.

Theorem 1 *If $\mathrm{IF}_\ell(x) \cap H = \emptyset$ for all $\ell \in \mathrm{exits}(prog)$ and all $x \in L$, then prog has the non-interference property with respect to H and L.*

References

[1] V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. the Thirteenth Annual IEEE Symposium on on Logic in Computer Science*. IEEE Computer Society Press, 2000. To appear.

[2] D. Denning. A lattice model of secure information flow. In *Communications of the ACM*, pages 236–243. ACM, 1976.

[3] P. Malacaria and C. L. Hankin. Non-deterministic games and program analysis: An application to security. In *Proc. the Twelfth Annual IEEE Symposium on on Logic in Computer Science*. IEEE Computer Society Press, 1999.

[4] F. Nielson, H. R. Neilson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999.

[5] J. C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981.

[6] A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. In *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99*, volume 1576 of *Lecture Notes in Computer Science*, pages 40–58. Springer-Verlag, 1999.

[7] D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *Proc. the Tenth IEEE Computer Security Foundations Workshop*, pages 156–168, Rockport, MA, 1997. IEEE Computer Society Press.