Information-Flow Control and Effects

NWPT2021

Carlos Tomé Cortiñas[1] Fabian Ruch

Chalmers University of Technology

November 6, 2021

Information-flow control (IFC) aims to protect *confidentiality* of data by controlling flows of information in a system.

Information-flow control (IFC) aims to protect *confidentiality* of data by controlling flows of information in a system.

IFC ensures that `secret` data cannot influence the `public`ly observable behaviour of programs.

Information-flow control (IFC) aims to protect *confidentiality* of data by controlling flows of information in a system.

IFC ensures that `secret` data cannot influence the `public`ly observable behaviour of programs.

We are interested in IFC for programs with effects.

Consider a security specification

$$\mathsf{Bool}_{\texttt{public}} \times \mathsf{Bool}_{\texttt{secret}} \Rightarrow \mathsf{Bool}_{\texttt{public}}$$

wrt the security policy $\texttt{public} \not\sqsupseteq \texttt{secret}$.

Consider a security specification

$$\mathsf{Bool}_{\texttt{public}} \times \mathsf{Bool}_{\texttt{secret}} \Rightarrow \mathsf{Bool}_{\texttt{public}}$$

wrt the security policy $\texttt{public} \not\sqsupseteq \texttt{secret}$.

Consider the programs

$$f = \lambda x, y.\, \mathsf{not}(x) \qquad\qquad g = \lambda x, y.\, \mathsf{and}(x, y)$$

Consider a security specification

$$\mathsf{Bool}_{\texttt{public}} \times \mathsf{Bool}_{\texttt{secret}} \Rightarrow \mathsf{Bool}_{\texttt{public}}$$

wrt the security policy $\texttt{public} \not\sqsupseteq \texttt{secret}$.

Consider the programs

$$f = \lambda x, y.\, \mathsf{not}(x) \qquad\qquad g = \lambda x, y.\, \mathsf{and}(x, y)$$

Noninterference: the public outputs of a program do not depend on its secret inputs, for all programs.

The Dependency Core Calculus (DCC) [1] is a simply-typed lambda calculus enhanced with a family of type constructors R_l for $l \in \{\mathtt{public}, \mathtt{secret}\}$ (pronounced "redaction").

The Dependency Core Calculus (DCC) [1] is a simply-typed lambda calculus enhanced with a family of type constructors R_l for $l \in \{\texttt{public}, \texttt{secret}\}$ (pronounced "redaction").

R is a graded monad over the lattice $\texttt{public} \sqsubseteq \texttt{secret}$.

The Dependency Core Calculus (DCC) [1] is a simply-typed lambda calculus enhanced with a family of type constructors R_l for $l \in \{\texttt{public}, \texttt{secret}\}$ (pronounced "redaction").

R is a graded monad over the lattice $\texttt{public} \sqsubseteq \texttt{secret}$.

In DCC *all programs are secure.*

Consider Moggi's monadic metalanguage [4] as a prog. language for printing to a `public`ly-observable channel:

Consider Moggi's monadic metalanguage [4] as a prog. language for printing to a `public`ly-observable channel:

> the type of computations is explicit $\mathsf{T}_{\texttt{public}}\, A$.

Consider Moggi's monadic metalanguage [4] as a prog. language for printing to a `public`ly-observable channel:

- the type of computations is explicit $\mathsf{T}_{\texttt{public}}\,A$.
- there is a do-nothing computation and sequencing of computations.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}(t) : \mathsf{T}_{\texttt{public}}\,A} \qquad \frac{\Gamma \vdash t : \mathsf{T}_{\texttt{public}}\,A \qquad \Gamma, x : A \vdash u : \mathsf{T}_{\texttt{public}}\,B}{\Gamma \vdash \mathsf{let}(t, x.\,u)}$$

Consider Moggi's monadic metalanguage [4] as a prog. language for printing to a `public`ly-observable channel:

- the type of computations is explicit $\mathsf{T}_{\texttt{public}}\, A$.
- there is a do-nothing computation and sequencing of computations.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}(t) : \mathsf{T}_{\texttt{public}}\, A} \qquad \frac{\Gamma \vdash t : \mathsf{T}_{\texttt{public}}\, A \qquad \Gamma, x : A \vdash u : \mathsf{T}_{\texttt{public}}\, B}{\Gamma \vdash \mathsf{let}(t, x.\, u)}$$

- there is a primitive that prints. $\dfrac{\Gamma \vdash b : \mathsf{Bool}}{\Gamma \vdash \mathsf{print}(b) : \mathsf{T}_{\texttt{public}}\, \mathsf{Unit}}$

Consider Moggi's monadic metalanguage [4] as a prog. language for printing to a $\texttt{public}$ly-observable channel:

- the type of computations is explicit $\mathsf{T}_{\texttt{public}}\, A$.
- there is a do-nothing computation and sequencing of computations.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}(t) : \mathsf{T}_{\texttt{public}}\, A} \qquad \frac{\Gamma \vdash t : \mathsf{T}_{\texttt{public}}\, A \qquad \Gamma, x : A \vdash u : \mathsf{T}_{\texttt{public}}\, B}{\Gamma \vdash \mathsf{let}(t, x.\, u)}$$

- there is a primitive that prints. $\dfrac{\Gamma \vdash b : \mathsf{Bool}}{\Gamma \vdash \mathsf{print}(b) : \mathsf{T}_{\texttt{public}}\, \mathsf{Unit}}$

A prog. language for IFC with printing to $\texttt{public} = \mathrm{DCC} + \mathrm{Moggi}^{\mathrm{print}}$

Consider the type

$$\mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \times \mathsf{R}_{\texttt{secret}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,(\mathsf{R}_{\texttt{public}}\,\mathsf{Unit})$$

Consider the type

$$\mathsf{R}_{\mathtt{public}}\,\mathsf{Bool} \times \mathsf{R}_{\mathtt{secret}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\mathtt{public}}\,(\mathsf{R}_{\mathtt{public}}\,\mathsf{Unit})$$

where printing to `public` is part of the `public` outputs of a program.

Consider the type

$$\mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \times \mathsf{R}_{\texttt{secret}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,(\mathsf{R}_{\texttt{public}}\,\mathsf{Unit})$$

where printing to `public` is part of the `public` outputs of a program.

We can construct a `public` computation from a `public` Boolean:

$$\mathsf{R}_{\texttt{public}}(\mathsf{print}) : \mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{R}_{\texttt{public}}\,(\mathsf{T}_{\texttt{public}}\,\mathsf{Unit})$$

Printing Effects: $\mathrm{DCC} + \mathrm{Moggi}^{\mathrm{print}}$

Consider the type

$$\mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \times \mathsf{R}_{\texttt{secret}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,(\mathsf{R}_{\texttt{public}}\,\mathsf{Unit})$$

where printing to `public` is part of the `public` outputs of a program.

We can construct a `public` computation from a `public` Boolean:

$$\mathsf{R}_{\texttt{public}}(\mathsf{print}) : \mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{R}_{\texttt{public}}\,(\mathsf{T}_{\texttt{public}}\,\mathsf{Unit})$$

however, we cannot "run" it since the types $\mathsf{R}_{\texttt{public}}$ and $\mathsf{T}_{\texttt{public}}$ *do not* interact.

We could add a new primitive

$$\mathsf{print}' : \mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,\mathsf{Unit}$$

Printing Effects: $\mathrm{DCC} + \mathrm{Moggi}^{\mathrm{print}}$ (C'ed)

We could add a new primitive

$$\mathsf{print}' : \mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,\mathsf{Unit}$$

and prove again that our language satisfies noninterference.

We could add a new primitive

$$\mathsf{print}' : \mathsf{R}_{\texttt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\texttt{public}}\,\mathsf{Unit}$$

and prove again that our language satisfies noninterference.

For combinations of $\mathrm{Moggi}^{\mathrm{print}}$ and other calculi for "effect-free" IFC, e.g. Sealing Calculus (SC) [5] or Moggi's metalanguage (again), we would have to redo all the work.

We could add a new primitive

$$\mathsf{print}' : \mathsf{R}_{\mathtt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\mathtt{public}}\,\mathsf{Unit}$$

and prove again that our language satisfies noninterference.

For combinations of $\mathrm{Moggi}^{\mathrm{print}}$ and other calculi for "effect-free" IFC, e.g. Sealing Calculus (SC) [5] or Moggi's metalanguage (again), we would have to redo all the work.

And what about other effects?

We could add a new primitive

$$\mathsf{print}' : \mathsf{R}_{\mathtt{public}}\,\mathsf{Bool} \Rightarrow \mathsf{T}_{\mathtt{public}}\,\mathsf{Unit}$$

and prove again that our language satisfies noninterference.

For combinations of $\mathrm{Moggi}^{\mathrm{print}}$ and other calculi for “effect-free” IFC, e.g. Sealing Calculus (SC) [5] or Moggi's metalanguage (again), we would have to redo all the work.

And what about other effects?

This approach is not modular.

To achieve (some) modularity, we abstract over concrete choices of IFC calculi and study effects in the *classified sets* model of Abadi et al. [1] and Kavvos [3].

To achieve (some) modularity, we abstract over concrete choices of IFC calculi and study effects in the *classified sets* model of Abadi et al. [1] and Kavvos [3].

The model captures dependency (independency) by interpreting types as sets *endowed* with a family of relations indexed by security levels that programs (functions) need to preserve. For instance,

To achieve (some) modularity, we abstract over concrete choices of IFC calculi and study effects in the *classified sets* model of Abadi et al. [1] and Kavvos [3].

The model captures dependency (independency) by interpreting types as sets *endowed* with a family of relations indexed by security levels that programs (functions) need to preserve. For instance,

$$\mathsf{R}_{\texttt{public}}\,\mathsf{Bool} = (\{\mathrm{tt}, \mathrm{ff}\}, \mathrm{R}_{\texttt{public}} = \mathrm{tt} \quad \mathrm{ff}, \mathrm{R}_{\texttt{secret}} = \mathrm{tt} \quad \mathrm{ff})$$

Classified Sets

To achieve (some) modularity, we abstract over concrete choices of IFC calculi and study effects in the *classified sets* model of Abadi et al. [1] and Kavvos [3].

The model captures dependency (independency) by interpreting types as sets *endowed* with a family of relations indexed by security levels that programs (functions) need to preserve. For instance,

$$\mathsf{R}_{\texttt{public}}\ \mathsf{Bool} = (\{\mathrm{tt}, \mathrm{ff}\}, \mathrm{R}_{\texttt{public}} = \mathrm{tt}\ \ \mathrm{ff}, \mathrm{R}_{\texttt{secret}} = \mathrm{tt}\ \ \mathrm{ff})$$

$$\mathsf{R}_{\texttt{secret}}\ \mathsf{Bool} = (\{\mathrm{tt}, \mathrm{ff}\}, \mathrm{R}_{\texttt{public}} = \mathrm{tt}\ \ \mathrm{ff}, \mathrm{R}_{\texttt{secret}} = \mathrm{tt}\ \ \mathrm{ff})$$

To achieve (some) modularity, we abstract over concrete choices of IFC calculi and study effects in the *classified sets* model of Abadi et al. [1] and Kavvos [3].

The model captures dependency (independency) by interpreting types as sets *endowed* with a family of relations indexed by security levels that programs (functions) need to preserve. For instance,

$$\mathsf{R}_{\texttt{public}}\,\mathsf{Bool} = (\{\mathrm{tt}, \mathrm{ff}\}, \mathrm{R}_{\texttt{public}} = \mathrm{tt}\quad \mathrm{ff}, \mathrm{R}_{\texttt{secret}} = \mathrm{tt}\quad \mathrm{ff})$$

$$\mathsf{R}_{\texttt{secret}}\,\mathsf{Bool} = (\{\mathrm{tt}, \mathrm{ff}\}, \mathrm{R}_{\texttt{public}} = \mathrm{tt}\quad \mathrm{ff}, \mathrm{R}_{\texttt{secret}} = \mathrm{tt}\quad \mathrm{ff})$$

A relation at level l captures what observers at l can distinguish, and programs preserving relations means they are forbidden to map indistinguishable inputs to distinguishable outputs.

Let the security policy be a join semilattice $(\mathcal{L}, \sqsubseteq, \bot, \vee)$.

Let the security policy be a join semilattice $(\mathcal{L}, \sqsubseteq, \bot, \vee)$.

The redaction monad is given by:

$$\mathrm{U}(\mathsf{R}_l(A)) := \mathrm{U}(A)$$

$$\mathrm{R}_{l'}(\mathsf{R}_l(A))(a, b) :\Leftrightarrow \begin{cases} \top & l \not\sqsubseteq l' \\ \mathrm{R}_{l'}(A)(a, b) & l \sqsubseteq l' \end{cases}$$

Let the security policy be a join semilattice $(\mathcal{L}, \sqsubseteq, \bot, \vee)$.

The redaction monad is given by:

$$\mathrm{U}(\mathsf{R}_l(A)) \coloneqq \mathrm{U}(A)$$

$$\mathrm{R}_{l'}(\mathsf{R}_l(A))(a,b) :\Leftrightarrow \begin{cases} \top & l \not\sqsubseteq l' \\ \mathrm{R}_{l'}(A)(a,b) & l \sqsubseteq l' \end{cases}$$

In words: it forces the relation of a classified set A to be the everywhere true relation at any level l' such that $l \not\sqsubseteq l'$.

We generalize the previous example to *printing on multiple channels* $c \in \mathcal{C}$ where a function $\mathrm{label} : \mathcal{L} \to \mathcal{C}$ specifies the security level of channels.

We generalize the previous example to *printing on multiple channels* $c \in \mathcal{C}$ where a function $\mathrm{label} : \mathcal{L} \to \mathcal{C}$ specifies the security level of channels.

For a subset $C \subseteq \mathcal{C}$, the monoid $(\mathsf{Out}_C, \epsilon_C, \cdot_C)$ is defined by:

$$\mathrm{U}(\mathsf{Out}_C) \coloneqq C \to \mathrm{List}(2)$$
$$\mathrm{R}_l(\mathsf{Out}_C)(o_1, o_2) :\Leftrightarrow \forall c \in C.\, \mathrm{label}(c) \sqsubseteq l \Rightarrow o_1(c) = o_2(c)$$

We generalize the previous example to *printing on multiple channels* $c \in \mathcal{C}$ where a function $\mathrm{label} : \mathcal{L} \to \mathcal{C}$ specifies the security level of channels.

For a subset $C \subseteq \mathcal{C}$, the monoid $(\mathsf{Out}_C, \epsilon_C, \cdot_C)$ is defined by:

$$\mathrm{U}(\mathsf{Out}_C) \coloneqq C \to \mathrm{List}(2)$$
$$\mathrm{R}_l(\mathsf{Out}_C)(o_1, o_2) :\Leftrightarrow \forall c \in C.\, \mathrm{label}(c) \sqsubseteq l \Rightarrow o_1(c) = o_2(c)$$

The graded monad $(\mathsf{W}, \eta, \mu, \mathrm{up})$ is given by $\mathsf{W}_C\, A \coloneqq A \times \mathsf{Out}_C$.

Consider programs of type $\mathsf{R}_l(\mathsf{W}_C\, A)$

Printing Effects in Classified Sets (C'ed)

Consider programs of type $\mathsf{R}_l(\mathsf{W}_C\ A)$, when are these secure to "run"?

Consider programs of type $\mathsf{R}_l(\mathsf{W}_C\, A)$, when are these secure to "run"?

Intuitively, when the information at level l is allowed to flow to every channel $c \in C$.

Consider programs of type $\mathsf{R}_l(\mathsf{W}_C\ A)$, when are these secure to "run"?

Intuitively, when the information at level l is allowed to flow to every channel $c \in C$.

Indeed, in classified sets there is a map

$$\delta_{l,C,A} : \mathsf{R}_l(\mathsf{W}_C\ A) \to \mathsf{W}_C(\mathsf{R}_l\ A)$$

exactly when $l \sqsubseteq \mathrm{label}(c)$ for all $c \in C$.

In the example of $\mathrm{DCC} + \mathrm{Moggi}^{\mathrm{print}}$, we add the primitive

$$\frac{\Gamma \vdash t : \mathsf{R}_{\mathtt{public}}(\mathsf{T}_{\mathtt{public}}\, A)}{\Gamma \vdash \mathsf{distr}(t) : \mathsf{T}_{\mathtt{public}}(\mathsf{R}_{\mathtt{public}}\, A)}$$

to the language.

In the example of $\mathrm{DCC} + \mathrm{Moggi}^{\mathrm{print}}$, we add the primitive

$$\frac{\Gamma \vdash t : \mathsf{R}_{\texttt{public}}(\mathsf{T}_{\texttt{public}}\, A)}{\Gamma \vdash \mathsf{distr}(t) : \mathsf{T}_{\texttt{public}}(\mathsf{R}_{\texttt{public}}\, A)}$$

to the language.

More generally, we obtain a prog. language for IFC with printing to multiple channels by *simply* combining DCC (or SC or $\ldots$) with EFE (Katsumata [2]) via a primitive

$$\frac{\Gamma \vdash t : \mathsf{R}_l(\mathsf{W}_C\, A)}{\Gamma \vdash \mathsf{distr}(t) : \mathsf{W}_C(\mathsf{R}_l\, A)}\ \forall c \in C.\, l \sqsubseteq \mathrm{label}(c)$$

Noninterference proofs à la Kavvos [3]

Noninterference proofs à la Kavvos [3]

"Explain" previous approaches to IFC with effects

Further Work

Noninterference proofs à la Kavvos [3]

"Explain" previous approaches to IFC with effects

Study other effects: e.g. exceptions or global store

Thank you for your attention!

[1] Martín Abadi et al. "A Core Calculus of Dependency". In: *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*. Ed. by Andrew W. Appel and Alex Aiken. ACM, 1999, pp. 147–160. DOI: 10.1145/292540.292555. URL: https://doi.org/10.1145/292540.292555.

[2] Shin-ya Katsumata. "Parametric effect monads and semantics of effect systems". In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. Ed. by Suresh Jagannathan and Peter Sewell. ACM, 2014, pp. 633–646. DOI: 10.1145/2535838.2535846. URL: https://doi.org/10.1145/2535838.2535846.

[3] G. A. Kavvos. "Modalities, cohesion, and information flow". In: *Proc. ACM Program. Lang.* 3.POPL (2019), 20:1–20:29. DOI: 10.1145/3290333. URL: https://doi.org/10.1145/3290333.

[4] Eugenio Moggi. "Notions of Computation and Monads". In: *Inf. Comput.* 93.1 (1991), pp. 55–92. DOI: 10.1016/0890-5401(91)90052-4. URL: https://doi.org/10.1016/0890-5401(91)90052-4.

[5] Naokata Shikuma and Atsushi Igarashi. "Proving Noninterference by a Fully