Information-Flow Control and Effects

Carlos Tomé Cortiñas[1] and Fabian Ruch

[1] Chalmers University of Technology

Language-based information-flow control (IFC) aims to ensure the confidentiality of data by controlling how information flows within programs. More concretely, IFC ensures that the *secret* inputs of a program will not influence (and thereby leak through) its *public* outputs—a property known as noninterference [5]. What constitutes program inputs and outputs (and hence what is the precise statement of noninterference) varies with what behaviour programs can exhibit and hypothetical attackers can observe. Since computational effects, like nontermination and input/output for instance, are integral to the behaviour of real programs, we are interested in studying IFC for programs that can exhibit them.

Static approaches to IFC (e.g. DCC [1] and SC [14]) usually use a graded monad R (for "redaction") to specify the confidentiality levels of data and control that flows abide by a security policy $\mathcal{L}$. In the simplest scenario, the policy concerns only two levels `secret` and `public`, and declares that all flows are permitted *except* those from `secret` to `public`, which are forbidden. In more complicated scenarios, the policy may be an arbitrary join semilattice $\mathcal{L} = (L, \sqsubseteq, \bot, \vee)$ [4] where L denotes the set of levels, $l \sqsubseteq l'$ the permission of flows from $l \in L$ to $l' \in L$, $\bot \in L$ the level from which all flows are permitted, and $l \vee l' \in L$ the level from which the flows that are permitted are exactly those that are permitted from both l and l'. We denote both the join-semilattice structure and its underlying set of levels by $\mathcal{L}$, in particular we write $l \in \mathcal{L}$ instead of $l \in L$.

Intuitively, given a type A and a level l of a policy $\mathcal{L} = (L, \sqsubseteq, \bot, \vee)$, $\mathsf{R}_l(A)$ denotes the type that forces the values of A to be indistinguishable from one another for any observer whose level is *not* above l with respect to $\sqsubseteq$. In their work on dependency analysis, Abadi et al. [1] introduce the so-called dependency category as a model of the redaction monad, and, more recently, Kavvos [8] formulates a "more high-tech" variation of that model called classified sets. The common feature of the dependency category and classified sets models is that they interpret a type by a cpo and a set of values, respectively, *endowed* with a family of relations R_l indexed by levels l, and a term by a map that preserves them. If we think of $\mathrm{R}_l(a, b)$ as encoding indistinguishability of a pair of values a and $b \in A$ then a map $f : A \to B$ is required to map them to indistinguishable values $f(a)$ and $f(b)$, i.e. $\mathrm{R}_l(f(a), f(b))$.

Starting from the work of Moggi on modelling computational effects using monads [9], there is a plethora of models of computational effects in the programming language literature (e.g. algebraic effects [11] and graded monads [7]). The basic idea is that values and computations that produce values denote different things: in Moggi [10] for instance, the type of computations that produce values of type A is denoted by the type $\mathsf{T}(A)$ where T has a monad structure.

Towards the goal of extending the type-based IFC approach to programs that can exhibit computational effects, we start by taking a look at printing effects. Extending a calculus like DCC or SC with nothing but a monad $\mathsf{T}_{\mathtt{public}}$ and a primitive $\mathsf{print}_{\mathtt{public}}(b) : \mathsf{T}_{\mathtt{public}}(\mathbb{1})$ for printing a Boolean $b : \mathbb{2}$ to a publicly observable channel is clearly insufficient because even the most basic noninterferent programs are not typeable. In particular, the type of $\mathsf{print}_{\mathtt{public}}$ does not allow us to print public Booleans $r : \mathsf{R}_{\mathtt{public}}(\mathbb{2})$. However, we can construct the *redacted* computation $\mathsf{bind}(r, b.\, \mathsf{print}_{\mathtt{public}}(b)) : \mathsf{R}_{\mathtt{public}}(\mathsf{T}_{\mathtt{public}}(\mathbb{1}))$, and the classified sets model suggests that further extending the calculus with a primitive $\mathsf{distr}(t) : \mathsf{T}_{\mathtt{public}}(\mathsf{R}_{\mathtt{public}}(\mathbb{1}))$ for "executing" redacted computations $t : \mathsf{R}_{\mathtt{public}}(\mathsf{T}_{\mathtt{public}}(\mathbb{1}))$ does not break noninterference. More generally, such a map $\mathsf{R}_l(\mathsf{T}_{l'}(A)) \to \mathsf{T}_{l'}(\mathsf{R}_l(A))$ exists for any A in classified sets whenever flows from l to

l' are permitted, i.e. $l \sqsubseteq l'$. Thus we are led to study extensions of calculi like DCC or SC with a *family* of monads T_l for $l \in \mathcal{L}$ that supports both a primitive $\mathsf{print}_l(b) : \mathsf{T}_l(\mathbb{1})$ for $b : \mathbb{2}$ and a distributive law $\mathsf{distr}_{l,l'} : \mathsf{R}_l(\mathsf{T}_{l'}(A)) \to \mathsf{T}_{l'}(\mathsf{R}_l(A))$ for $l \sqsubseteq l'$. In fact, in order to support printing to a set $\mathcal{C}$ of output channels $c \in \mathcal{C}$ with assigned security levels $\mathrm{label}(c) \in \mathcal{L}$, we study extensions with a monad T graded by the powerset lattice $\mathcal{P}(\mathcal{C})$. In this case the type of the distributive law becomes $\mathsf{R}_l(\mathsf{T}_C(A)) \to \mathsf{T}_C(\mathsf{R}_l(A))$ for a subset $C \in \mathcal{P}(\mathcal{C})$ such that $\forall c \in C.\, l \sqsubseteq \mathrm{label}(c)$. Without committing to a particular calculus yet, we study IFC and computational effects using graded monads and distributive laws in the context of classified sets.

Graded Redaction

Let us recall the redaction monad on classified sets [8] over a join semilattice $(\mathcal{L}, \sqsubseteq, \bot, \vee)$:

Definition 1 (Category of classified sets)**.** A classified set A over $\mathcal{L}$ consists of a set $\mathrm{U}(A)$ and a family of binary relations $\mathrm{R}_l(A)$ on $\mathrm{U}(A)$ indexed by labels $l \in \mathcal{L}$. Call two elements $a_1, a_2 \in \mathrm{U}(A)$ *indistinguishable* at label $l \in \mathcal{L}$ if $\mathrm{R}_l(A)(a_1, a_2)$. A map $f : A \to B$ between classified sets A and B consists of a function $\mathrm{U}(f) : \mathrm{U}(A) \to \mathrm{U}(B)$ that preserves indistinguishability, i.e. $\mathrm{R}_l(B)(f(a_1), f(a_2))$ whenever $\mathrm{R}_l(A)(a_1, a_2)$. Denote the category of classified sets over $\mathcal{L}$ and maps between them by $\mathrm{CSet}(\mathcal{L})$.

Definition 2 (Pointwise redaction monad)**.** Given $l \in \mathcal{L}$, the monad $(\mathsf{R}_l, \eta_l, \mu_l)$ on $\mathrm{CSet}(\mathcal{L})$ is defined by:

$$\mathrm{U}(\mathsf{R}_l(A)) \coloneqq \mathrm{U}(A) \qquad \mathrm{U}(\mathsf{R}_l(f)) \coloneqq \mathrm{U}(f)$$
$$\mathrm{R}_{l'}(\mathsf{R}_l(A))(a,b) :\Leftrightarrow \begin{cases} \mathrm{R}_{l'}(A)(a,b) & l \sqsubseteq l' \\ \top & \text{otherwise} \end{cases} \qquad \begin{aligned} \mathrm{U}(\eta_{l,A}) &\coloneqq \mathrm{id}_{\mathrm{U}(A)} \\ \mathrm{U}(\mu_{l,A}) &\coloneqq \mathrm{id}_{\mathrm{U}(A)} \end{aligned}$$

In other words, the monad R_l at some label l forces all elements to be pairwise indistinguishable at labels that are *not* above l. Note that this includes but is not restricted to those labels that are strictly below l since the ordering of labels is not required to be total.

Proposition 1.

1. R_l *is idempotent:* $\mu_l : \mathsf{R}_l \circ \mathsf{R}_l \dot{\to} \mathsf{R}_l$ *is invertible*
2. R_l *preserves limits, in particular* R_l *preserves binary products*
3. R_l *preserves exponentials, and hence* R_l *is a Cartesian closed functor*

The family of monads R_l indexed by $l \in \mathcal{L}$ is in fact graded [7] with respect to $\mathcal{L}$ seen as a (strict) monoidal category, i.e. we have coherent natural transformations $up_{l,l'} : \mathsf{R}_l \dot{\to} \mathsf{R}_{l'}$ for $l \sqsubseteq l'$ given by $\mathrm{id}_{\mathrm{U}(A)}$ at A and a graded multiplication $\mu_{l,l'} : \mathsf{R}_l \circ \mathsf{R}_{l'} \dot{\to} \mathsf{R}_{l \vee l'}$ that generalizes the pointwise multiplication $\mu_l : \mathsf{R}_l \circ \mathsf{R}_l \dot{\to} \mathsf{R}_l$. Further, the grading is strongly monoidal:

Proposition 2.

1. *At label* $\bot$*, the unit* $\eta_\bot : \mathrm{Id} \dot{\to} \mathsf{R}_\bot$ *is invertible*
2. *For any two labels* l*,* l'*, the graded multiplication* $\mu_{l,l'} : \mathsf{R}_l \circ \mathsf{R}_{l'} \dot{\to} \mathsf{R}_{l \vee l'}$ *is invertible, and hence the redaction monad is* commutative *and* subsumptive *in the sense that* $\mu_{l',l}^{-1} \cdot \mu_{l,l'} : \mathsf{R}_l \circ \mathsf{R}_{l'} \dot{\to} \mathsf{R}_{l'} \circ \mathsf{R}_l$ *is invertible and if* $l \sqsupseteq l'$ *then* $\mu_{l,l'} : \mathsf{R}_l \circ \mathsf{R}_{l'} \dot{\to} \mathsf{R}_l$ *is invertible, respectively*

Printing Effects

We generalize the printing example from earlier to a *set* $\mathcal{C}$ of channels. We assume that the security policy specifies a security level for each channel $c \in \mathcal{C}$ via a function $\mathrm{label} : \mathcal{C} \to \mathcal{L}$. The idea is that only observers whose security level l is above $\mathrm{label}(c)$, i.e. $\mathrm{label}(c) \sqsubseteq l$, may observe outputs on a given channel c.

We model programs with printing effects using a graded monad:

Definition 3 (Output monoid)**.** For a subset $C \subseteq \mathcal{C}$, the monoid $(\mathsf{Out}_C, \epsilon_C, \cdot_C)$ is defined by:

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

Note that for $C' \supseteq C$ we have a monoid homomorphism $up_{C,C'} : \mathsf{Out}_C \to \mathsf{Out}_{C'}$ that assigns the empty list $[]$ to channels $c' \in C' \setminus C$.

Definition 4 (Output monad)**.** The graded monad $(\mathsf{W}, \eta, \mu, up)$ is given by $\mathsf{W}_C(A) \coloneqq A \times \mathsf{Out}_C$ and the structure maps as induced by the monoid maps ϵ_C, $\cdot_C$, and $up_{C,C'}$, respectively.

Proposition 3. *The output monad in the classified sets model validates the intuition that a computation of security level l, i.e. an element of $\mathsf{R}_l(\mathsf{W}_C(A))$, that only prints to channels above l, i.e. $C \subseteq \mathcal{C}$ is such that $l \sqsubseteq \mathrm{label}(c)$ for all $c \in C$, is secure to run in the sense that we have a map $\delta_{l,C,A} : \mathsf{R}_l(\mathsf{W}_C(A)) \to \mathsf{W}_C(\mathsf{R}_l(A))$. In fact, the family of maps $\delta_{l,C,A}$ indexed by A constitutes a distributive law*[1]*.*

Proof. R_l preserves products (see Proposition 1.2) and we have that $\eta_{l,\mathsf{Out}_C} : \mathsf{Out}_C \to \mathsf{R}_l(\mathsf{Out}_C)$ has an inverse $i_{l,C}$ if $l \sqsubseteq \mathrm{label}(C)$. The required diagrams for the composites $\delta_{l,C,A} \coloneqq (\mathsf{R}_l(\mathrm{fst}), \mathsf{R}_l(\mathrm{snd}))\ ;\ A \times i_{l,C}$ to constitute a distributive law commute because the underlying functions of all maps involved are identities. □

As a result of these investigations of the classified sets model, a programming language for IFC with printing effects is given by simply combining SC or DCC with EFᴇ [7, Section 5.1] via a primitive distr that "reifies" the map $\delta_{l,C,A}$ in the syntax:

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

Noninterference for the resulting language states that for any program $prog : \mathsf{R}_{\mathtt{secret}}(2) \to \mathsf{W}_C(\mathbb{1})$ and any two values $s_1, s_2 : \mathsf{R}_{\mathtt{secret}}(2)$ it is the case that $prog\, s_1$ and $prog\, s_2 : \mathsf{W}_C(\mathbb{1})$ produce the same output on those channels $c \in C$ that are observable by $\mathtt{public} \not\sqsupseteq \mathtt{secret}$, i.e. $\mathrm{label}(c) \sqsubseteq \mathtt{public}$.

Further Work

The next step is to prove noninterference theorems along the lines of Kavvos [8] using the framework of the redaction monad and semantic structures for effects in classified sets. This should then apply to practical approaches to IFC (e.g. MAC [13] and SLIO [3, 12]) by, for instance, modelling the (idealized) monad MAC_l (cf. [15]) for printing to a fixed set of channels $\mathcal{C}$ as the composite monad $\mathsf{W}_C \circ \mathsf{R}_l$ where $C = \{c \in \mathcal{C} \mid l \sqsubseteq \mathrm{label}(c)\}$. We also leave it to further work to investigate other effects in the context of classified sets and compare them to their treatment in the IFC literature, e.g. global store [16] or exceptions [6].

[1] A distributive law [2] for a monad S over a monad P is given by a natural transformation $\delta : P \circ S \dot{\to} S \circ P$ that commutes with the unit and multiplication maps of P and S.

References

[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] Jon Beck. "Distributive laws". In: *Sem. on Triples and Categorical Homology Theory (ETH, Zürich, 1966/67)*. Springer, Berlin, 1969, pp. 119–140.

[3] Pablo Buiras, Dimitrios Vytiniotis, and Alejandro Russo. "HLIO: mixing static and dynamic typing for information-flow control in Haskell". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*. Ed. by Kathleen Fisher and John H. Reppy. ACM, 2015, pp. 289–301. DOI: 10.1145/2784731.2784758. URL: https://doi.org/10.1145/2784731.2784758.

[4] Dorothy E. Denning. "A Lattice Model of Secure Information Flow". In: *Commun. ACM* 19.5 (1976), pp. 236–243. DOI: 10.1145/360051.360056. URL: https://doi.org/10.1145/360051.360056.

[5] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models". In: *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*. IEEE Computer Society, 1982, pp. 11–20. DOI: 10.1109/SP.1982.10014. URL: https://doi.org/10.1109/SP.1982.10014.

[6] Catalin Hritcu et al. "All Your IFCException Are Belong to Us". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 3–17. DOI: 10.1109/SP.2013.10. URL: https://doi.org/10.1109/SP.2013.10.

[7] 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.

[8] 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.

[9] Eugenio Moggi. "Computational Lambda-Calculus and Monads". In: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 1989, pp. 14–23. DOI: 10.1109/LICS.1989.39155. URL: https://doi.org/10.1109/LICS.1989.39155.

[10] 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.

[11] Gordon D. Plotkin and Matija Pretnar. "Handlers of Algebraic Effects". In: *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*. Ed. by Giuseppe Castagna. Vol. 5502. Lecture Notes in Computer Science. Springer, 2009, pp. 80–94. DOI: 10.1007/978-3-642-00590-9_7. URL: https://doi.org/10.1007/978-3-642-00590-9%5C_7.

[12] Vineet Rajani and Deepak Garg. "On the expressiveness and semantics of information flow types". In: *J. Comput. Secur.* 28.1 (2020), pp. 129–156. DOI: 10.3233/JCS-191382. URL: https://doi.org/10.3233/JCS-191382.

[13] Alejandro Russo. "Functional pearl: two can keep a secret, if one of them uses Haskell". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*. Ed. by Kathleen Fisher and John H. Reppy. ACM, 2015, pp. 280–288. DOI: 10.1145/2784731.2784756. URL: https://doi.org/10.1145/2784731.2784756.

[14] Naokata Shikuma and Atsushi Igarashi. "Proving Noninterference by a Fully Complete Translation to the Simply Typed Lambda-Calculus". In: *Log. Methods Comput. Sci.* 4.3 (2008). DOI: 10.2168/LMCS-4(3:10)2008. URL: https://doi.org/10.2168/LMCS-4(3:10)2008.

[15] Marco Vassena et al. "MAC A verified static information-flow control library". In: *Journal of Logical and Algebraic Methods in Programming* 95 (2018), pp. 148–180. ISSN: 2352-2208. DOI: https://doi.org/10.1016/j.jlamp.2017.12.003. URL: https://www.sciencedirect.com/science/article/pii/S235222081730069X.

[16] Stephan Arthur Zdancewic. *Programming languages for information security*. Cornell University, 2002.