Algebra for Quantitative Information Flow

AK McIver[1], CC Morgan[2], and T Rabehaja[1] ⋆

[1] Dept. Computing, Macquarie University, Australia
[2] School of Comp. Sci. and Eng., UNSW, and Data61, Australia

Abstract. A core property of program semantics is that local reasoning about program fragments remains sound even when the fragments are executed within a larger system. Mathematically this property corresponds to *monotonicity of refinement*: if A refines B then $\mathcal{C}(A)$ refines $\mathcal{C}(B)$ for any (valid) context defined by $\mathcal{C}(\cdot)$.
In other work we have studied a *refines* order for information flow in programs where the comparison defined by the order preserves both functional *and* confidentiality properties of secrets. However the semantic domain used in that work is only sufficient for scenarios where either the secrets are static (i.e. once initialised they never change), or where contexts $\mathcal{C}(\cdot)$ never introduce *fresh* secrets.
In this paper we show how to extend those ideas to obtain a model of information flow which supports local reasoning about confidentiality. We use our model to explore some algebraic properties of programs which contain secrets that can be updated, and which are valid in arbitrary contexts made up of possibly freshly declared secrets.

Keywords: Refinement; information flow; security; monotonicity; probabilistic semantics; compositional reasoning; Dalenius Desideratum.

1 Introduction

Algebras are powerful tools for describing and reasoning about complex behaviours of programs and algorithms. The effectiveness of algebraic reasoning is founded on the principle that equalities between expressions mean that those expressions are interchangeable: if P and Q are algebraic expressions representing programs that are considered to have "the same" behaviours, then $\mathcal{C}(P)$ and $\mathcal{C}(Q)$ must also exhibit "the same" behaviours for any program context $\mathcal{C}(\cdot)$ represented in the algebra. In theories of non-interference security this principle poses a surprising challenge in models describing properties of programs containing secrets which can both be updated during program execution, *and* which can be partially observed by a passive but curious adversary. Although there are many semantic models for reasoning about information flow, they typically support only a subset of these behaviours. For example [1, 4, 24] assume that the secrets once set never

⋆ We acknowledge the support of the Australian Research Council Grant DP140101119. This work was carried out while visiting the Security Institute at ETH Zürich.

change. Our more recent work [13, 16] does allow updates to secrets, however it also assumes a "closed system model" for program execution, where there is a single global secret type which must be declared at the outset.

In this paper we show how to extend the applicability of algebraic reasoning for all contexts and behaviours, in particular we remove the assumption of a closed system model of operation. On a technical level this requires generalising our earlier model [13, 16] based on Hidden Markov Models (*HMM*'s) to include not only information flow about some declared secret, but also information flow that can potentially have an impact on third-party secrets –undeclared in a given program fragment– but introduced later as part of a context $\mathcal{C}(\cdot)$. In terms of practical verification this theoretical extension is crucial: it means that local reasoning about program fragments remains sound even when those program fragments are executed in contexts which could contain arbitrary secrets.

The surprise here is that our extension of standard *HMM*'s is related to an old problem in privacy in "read-only" statistical databases, first articulated by Dalenius [6] and later developed by Dwork [7]. It says that third-party information flow is possible if a database's contents are known to be correlated with data not in the databases; in this case, information revealed by a query could also lead to information leaked through the correlation.

Our approach rests fundamentally on Goguen and Meseguer's original model for *qualitative* non-interference [9] and on more recent work in *quantitative* information flow of communication channels [24, and its citations]; we combine them into a denotational program-semantics in the style of [21, for qualitative] and [13, for quantitative].

In [9] the program state is divided into high- and low-security portions, and a program is said to be "non-interfering" if the low variables' final values cannot be affected by the the high variables' initial values. Note that this is a qualitative judgement: a program either suffers from interference or it doesn't. Here instead we follow others in *quantifying* the interference in a program [24], since it has been recognised for some time that absolute noninterference is in practice too strong: even a failed password guess leaks what the password is not. Then a more personal choice is that we address leaks wrt. the high variables' *final* values, not their initial ones; we explain below why we believe this view is important for refinement.

We embed both of the above features in a denotational semantics based on *HMM*'s supporting a refinement relation in the style of [3, 12, 20]: our programs here are probabilistic, without demonic choice, and include a special statement `leak` that passes information directly to an adversary. A result is that the apparently "exotic" problems we highlighted above become surprisingly mundane. For example, the program state is entirely secret (all of it is "high"), and leaks are not through "low variables" but rather are explicit via the special `leak` statements. Furthermore, conventional refinement –the tradition on which we draw– compares programs' final states, not their initial ones: and thus so do we here. Finally, the "Dalenius" problem, of our potential effect on third parties

// Secret X is initially uniformly distributed over $\mathcal{X} = \{0..N-1\}$.

Program run in isolation

```
while(X > 1) {
   leak (X > 1)        †
   X:= X-2
}
```

Program run in context

```
Z:= X                  *
while(X > 1) {
   leak (X > 1)        †
   X:= X-2
}
```

* *The program is run in a context where new secret* `Z` *is freshly declared, and then set to the initial value of* `X`

† `leak (...)` *This models the timing leak, by allowing the adversary to count the number of times the loop checks the loop guard as it executes.*

Fig. 1. Timing attacks in isolation and in context

unknown to us, is merely the issue of preserving refinement when extra variables are declared that were not mentioned in our original program.

We make the following contributions.

(a) We note that the Dalenius issue is simply that security of data can be affected by programs that do not refer to it, and we illustrate it by example;
(b) We review Hidden Markov Models and explain how they can be used as the basis for an abstract model of programs that model information flows to secrets that can be updated;
(c) We show, by considering *HMM*'s as transformers of correlations, that they can also model information flows of possibly correlated secrets;
(d) We define a partial order on *abstract HMM*'s based on the information order defined elsewhere [2, 1, 18] and show that it is general in the sense that equality is maintained in arbitrary contexts.

We begin in §2 with an example addressing (a) just above. In §3 we show how to model program fragments as *HMM*'s and, in doing so, we show that to address the issue at (a) it is sufficient to model only the correlation between initial and final states of secrets in local reasoning in order to predict general information flows about arbitrary secrets in arbitrary contexts. In §4 we show how to define a partial order on *HMM*'s as correlation transformers resulting in a general law of equivalence (Thm. 1). Finally in §5 we prove some general algebraic laws valid for abstract *HMM*'s as correlation transformers.

2 Getting real: updating secrets and third-party "collateral" damage in <u>everyday</u> <u>programs</u>

Fig. 1 illustrates the difference between running a program in isolation versus in a system where there are multiple possibly correlated secrets. We adopt the working assumption of a passive, but curious adversary, by which we mean an adversary who is trying to guess secrets. She does so by observing the program as it executes and matching observations to (possible) values of the secret. The adversary is able to do this because we assume that she has a copy of the program code. The adversary is not actively malicious, however: she cannot change data nor affect normal program operation.

In both scenarios in Fig. 1 there is a loop which is subject to a timing attack and, for simplicity, we assume that it can be performed by counting how many times the guard is executed. We use an explicit statement `leak` as a signal that our passive adversary can observe when the loop body is executed, even though she cannot observe the exact value of the secret `X`. When `leak (X>1)` is executed, the adversary learns whether the current value of `X` is strictly greater than 1 or not. She cannot deduce anything else about `X`, but by accumulating all her observations, and her knowledge of the program code, over time she can deduce many facts about the initial and final states of `X`.

First of all, since the adversary knows the program code, without even executing it, she deduces that the final value of `X` will be either 0 or 1. To learn something about the *initial* state, she must observe the program as it runs: if she observes that $(\mathtt{X} > \mathtt{1})$ was true three times in total then the initial value of `X` *must have been* either 6 or 7; if it was true only twice, then the initial value of `X` *must have been* either 4 or 5.

What can the adversary do with this analysis?

Suppose that the adversary also knows that there are no other secrets, i.e. even if the loop is part of a larger piece of code, the only secret variable referred to in that code is `X`. This means that her knowledge about the initial state of `X` is not useful because `X` is no longer equal to its initial state (unless it was 0 or 1). Thus the information leak about the initial value of `X` is not useful to the adversary who tries to guess the current value of `X`.

On the other hand, suppose that in a different scenario there is a second secret `Z` and it is initially correlated with `X`, as in the program at the right in Fig. 1. Now the fact that `X` was initially 6 or 7 is highly significant because it tells the adversary that the *current* value of `Z` is either 6 or 7. And the adversary might actually be more interested in `Z` than in `X`. This is the Dalenius problem referred to above: the impact of some information leaks become manifest only when programs are executed in some context with fresh secrets.

In sequential program semantics, it is usual to focus on the final state because the aim is to establish some goal by updating the variables, and so when we integrate security we still need to consider final values. Indeed it is the concentration on final values that allows small state-modifying programs, whether secure or not, to be sequentially composed to make larger ones [13, 15, 16]. But here we have demonstrated by example that this is not enough if we want a semantics which is compositional when contexts introduce new secrets, because if the semantics only captures the uncertainty about the final states, we have potentially lost the Dalenius effect i.e. how much uncertainty remains about the correlated secrets. For example, if we only consider scenarios where there is a single secret `X`, then we would only need to consider the remaining uncertainty of final states. We could then confidently argue that the loop on the left at Fig. 1 is equivalent to the program $\texttt{X} := \texttt{X} \mod 2$. However, consider context $\mathcal{C}(P)$ defined by $\texttt{Z} := \texttt{X}; P$, where $\texttt{Z}$ is a secret (and P is some program fragment). We must now consider confidentiality properties of *both* $\texttt{Z}$ and $\texttt{X}$, and for monotonicity of refinement since $\mathcal{C}(\texttt{X} := \texttt{X} \mod 2)$ leaks less than $\mathcal{C}(\texttt{leak}(\texttt{X} > 1); \texttt{X} := \texttt{X} \mod 2)$, then our semantics must distinguish between $(\texttt{X} := \texttt{X} \mod 2)$ and $\texttt{leak}\ (\texttt{X} > 1); \texttt{X} := \texttt{X} \mod 2$, even though all their properties concerning the final state of $\texttt{X}$ are the same.

In the remainder of the paper we describe a semantics which combines information flow and state updates in which refinement between program fragments can be determined by local reasoning alone (i.e. only about $\texttt{X}$ when the program fragments only refer to $\texttt{X}$). Crucially the refinement relation satisfies monotonicity when contexts can include fresh secrets (as in $\mathcal{C}(\cdot)$ above).

3 A denotational model for quantitative information flow

3.1 Review of the probability monad and hyper-distributions

We model secrets as probability distributions reflecting the uncertainty about their values (if indeed they are secret). Our semantics for programs computing with secrets is based on the "probability monad", which we now review; and in §3.2 we explain how it can be used to define a model for information flow.

Given a state space $\mathcal{X}$ we write $\mathbb{D}\mathcal{X}$ for the set of probability distributions over $\mathcal{X}$, which we assume here to be finite so that we consider only discrete distributions that assign a probability to every individual element in $\mathcal{X}$. For some distribution δ in $\mathbb{D}\mathcal{X}$ we write δ_x, between 0 and 1, for the probability that δ assigns to x in particular.[3]

The *probability monad* [8] is based on $\mathbb{D}$ as a "type constructor" that obeys a small collection of laws shared by other, similar constructors like say the powerset operator $\mathbb{P}$. Each monad has two polymorphic functions η for "unit", and μ for "multiply", that interact with each other in elegant ways. For example in $\mathbb{P}$, unit η has type $\mathcal{X} \to \mathbb{P}\mathcal{X}$, generically in $\mathcal{X}$, and $\eta(x)$ is the singleton set $\{x\}$; in $\mathbb{D}$ the unit has type $\mathcal{X} \to \mathbb{D}\mathcal{X}$ and $\eta(x)$ is $[x]$, the point-distribution on x.[4]

For multiply, in $\mathbb{P}$ it is distributed union, taking a set of sets to the one set that is the union of them all, having thus the type $\mathbb{P}^2\mathcal{X} \to \mathbb{P}\mathcal{X}$. In $\mathbb{D}$ we can construct $\mathbb{D}(\mathbb{D}\mathcal{X})$ or $\mathbb{D}^2\mathcal{X}$ which is the set of "distributions of distributions" (just as $\mathbb{P}^2$ was sets of sets), equivalently distributions over $\mathbb{D}\mathcal{X}$. We call these "hyper-distributions", and use them below to model information flow; we normally use capital Greeks like Δ for hyper-distributions. In $\mathbb{D}$, the multiply has type $\mathbb{D}^2\mathcal{X} \to \mathbb{D}\mathcal{X}$ — it "squashes" distributions of distributions back to a single one,

[3] Mostly we use the conventional $f(x)$ for application of function f to argument x. Exceptions include δ_x for δ applied to x and $\mathbb{D}f$ for functor $\mathbb{D}$ applied to f and $f.x.y$ for function $f(x)$, or $f.x$, applied to argument y, and $[\![H]\!].\pi$, when H is an *HMM* inside semantic brackets $[\![\cdot]\!]$.

[4] The point distribution on x assigns probability 1 to x alone, and probability 0 to everything else; we write it $[x]$.

and is defined $\mu(\mathit{\Delta})_x{:=}\ \sum_{\delta:\,\mathbb{D}\mathcal{X}} \mathit{\Delta}_\delta{\times}\delta_x$, giving the probability assigned to x as the sum of the inner probabilites (of x), each scaled by their corresponding outers. We also write avg for the μ of $\mathbb{D}\mathcal{X}$.[5]

Monadic type-constructors like $\mathbb{P}$ and $\mathbb{D}$ are *functors*, meaning they can be applied to functions as well as to objects: thus for f in $\mathcal{X}{\rightarrow}\mathcal{Y}$ the function $\mathbb{P}f$ is of type $\mathbb{P}\mathcal{X}{\rightarrow}\mathbb{P}\mathcal{Y}$ so that for X in $\mathbb{P}\mathcal{X}$ we have $\mathbb{P}f(X) = \{f(x) \mid x{\in}X\}$ in $\mathbb{P}\mathcal{Y}$. In $\mathbb{D}$ instead we get the *push forward* of f, so that for π in $\mathbb{D}\mathcal{X}$ we have $(\mathbb{D}f)(\pi)_y = \sum_{f(x)=y} \pi_x$.

We shall write $\mathcal{X}$ and $\mathcal{Z}$ for secret types and $\mathcal{Y}$ for the type of observations used to enable us to model information flow. Given a joint distribution J in $\mathbb{D}(\mathcal{X}{\times}\mathcal{Y})$ we define hyper-distributions of secrets by abstracting from the values of observations as follows: we retain only the probabilities that an observation occurs, together with the residual uncertainty in $\mathcal{X}$ related to those observations. The probability of observation y is computed from $\overrightarrow{J}$ the marginal on $\mathcal{Y}$ (relative to the joint distribution J in $\mathbb{D}(\mathcal{X}{\times}\mathcal{Y})$), so that $\overrightarrow{J}_y{:=}\ \sum_{x:\,\mathcal{X}} J_{xy}$. Next, for each observation y, we can compute $J^y{:}\,\mathbb{D}\mathcal{X}$, the conditional probability distribution over x given this y. It is $J^y_x{:=}\ J_{xy}/\overrightarrow{J}_y$. This conditional probability distribution represents the *residual uncertainty* of x by taking the observation y into account. We now write $[J]{:}\,\mathbb{D}^2\mathcal{X}$ so that δ is in the support of $[J]$ provided that there is some $y{:}\,\mathcal{Y}$ such that $\delta = J^y$. Finally $[J]_\delta$ is equal to the sum $\sum_{\delta=J^y} \overrightarrow{J}_y$.

3.2 Review of "traditional" vs. more recent quantitative information flow semantics for (non-)interference

Goguen and Meguer's treatment of non-interference separated program variables into high- and low-security, and defined "non-interference" of high inputs with low outputs: that a change in a high input-value should never cause a consequential change in a low output-value [9]. A hostile observer of final low values in that case could never learn anything about initial high values. Subsequent elaborations of this allowed more nuanced measurements, determining "how much" information was revealed by low variables about the initial values of high variables. The measurements can be of many different kinds: Shannon Entropy was until recently the default choice, but that has now been significantly generalised [2].

In the traditional style, the side channel attack of Fig. 1 would be modelled by an explicit assignment to some low-security variable `L` say, actually in the program text; and the program's security would be assessed in terms of how much final observations of `L` could tell you about the original secret value `X`. In particular, the program's action on `X` and `L` together would be described as a

[5] We are aware that in $\mathbb{D}(\mathbb{D}\mathcal{X})$ the outer $\mathbb{D}$ is not acting over a finite type: indeed $\mathbb{D}\mathcal{X}$ is non-denumerable even when $\mathcal{X}$ is finite, so a fully general treatment would use proper measures as we have done elsewhere [14, 16]. Here however we use the fact that, for programs, the only members of $\mathbb{D}^2\mathcal{X}$ we encounter have finite support (i.e. finitely many $\mathbb{D}\mathcal{X}$'s within them), and constructions like $\sum_{\delta:\,\mathbb{D}\mathcal{X}} \mathit{\Delta}_\delta\delta_x$ remain meaningful.

joint distribution, and standard Bayesian reasoning would be used to ask (and answer) questions like "Given this particular final value of `L`, how do we change our *prior* belief of the distribution on `X` to an *a-posteriori* distribution on `X`?"

Our more recent style here is instead to make the whole of the program's state-space hidden, and to model information flow to the hostile observer via an explicit `leak` statement. Execution of a statement `leak` E, for some expression E in the program variables (here just `X`), models the emission of E's value at that moment directly to an observer: from it, she makes deductions about `X`'s possible values at that point. Usually E will not be injective (since otherwise she would learn `X` exactly); but, unless E is constant, she will still learn something. But how much? Assume in the following program that `X` is initially one of $0, 1, 2$ with equal probability:

```
X:= X+1;   leak (X mod 2);   X:= 2*X   .
```

Informal reasoning would say that after the first statement `X` is uniformly distributed over 1–3; after the second statement it would (via Bayesian reasoning) either be uniform over 1,3 (if a 1 was leaked) or it would certainly be 2 (if a zero was leaked) — and the first case would occur with probability $^2/_3$, the second with probability $^1/_3$. After the third statement, then, our observer would $^2/_3$ of the time believe `X` to be uniform over $2, 6$ and $^1/_3$ of the time know that `X` was 4. A key feature of this point of view is that her final "belief state" can be summarised in a hyper-distribution introduced above. In this case we would have the distribution "uniformly either 2 or 6" itself with probability $^2/_3$, and the distribution "certainly 4" itself with probability $^1/_3$.

Hyper-distributions (objects of type $\mathbb{D}^2\mathcal{X}$), or *hypers* for short, explicitly structure the relationship between *a-posteriori* distributions ("2 or 6" and "certainly 4" above) and the probabilities with which those "posteriors" occur ($^2/_3$ and $^1/_3$ resp.) — we call the a-posteriori, or posterior distributions "inners" of the hyper; and we call the distribution over them the "outer". In our model for security programs we use this two-layered feature to provide a clean structure for information flow. It is based on our general conviction that the value of an observation itself is not important; what matters is how much change that observation induces in the probability distribution of a secret value [18]. Therefore the observations' values do not need to participate in the semantics of information flow, and its formalisation becomes much simpler.

That semantic simplification also enables a calculus of information flow, explored in other work [13], and allows the use of monads, a very general semantic tool for rigorous reasoning about computations [19] and even the implementation of analysis tools [23].

Definition 1. *[16] Given a state space $\mathcal{X}$ of hidden (i.e. high-security) variables, a denotational model of quantitative non-interference secure-sensitive programs consists of functions from prior (input) distributions on the state space to hyper (output) distributions on the same space — the domain is $\mathbb{D}\mathcal{X}{\rightarrow}\mathbb{D}^2\mathcal{X}$.*

Given two abstract programs $h^1, h^2 {:} \mathbb{D}\mathcal{X} \to \mathbb{D}^2\mathcal{X}$ we define their composition as $h^1; h^2 {:=} \mathsf{avg} \circ \mathbb{D}h^2 \circ h^1$, which is also of type $\mathbb{D}\mathcal{X} \to \mathbb{D}^2\mathcal{X}$. [6]

In other work we have shown that Def. 1 is an abstraction of *HMM*'s and works well in *closed* systems where there is exactly one secret $\mathcal{X}$ and that the composition defined using the Giry constructors correspond exactly to composition of *HMM*'s. However, as illustrated by Fig. 1, modelling only the residual uncertainty of the final state does not enable us to draw conclusions about behaviour of the program fragment running in the larger context in which fresh secrets participate in some larger computation. It turns out however that we are able to predict the behaviour of a program fragment in such larger contexts by preserving the uncertainty with respect to the correlation between initial *and* final states. We do this by viewing *HMM*'s as *correlation transformers* and we show that this is sufficient to obtain a compositional model suitable for open systems where contexts of execution can contain arbitrary fresh secrets.

3.3 *HMM*'s as correlation transformers

The basic step of an *HMM* consists of a secret type $\mathcal{X}$, and two stochastic matrices[7], one to describe the updates to the secret (called a Markov matrix) and the other to describe the information flowing about the secret (called a Channel matrix). A Markov matrix M (over $\mathcal{X}\times\mathcal{X}$) defines $M_{xx'}$ to be the transition probability for an initial value of the secret x updated to x'. A Channel matrix C (over $\mathcal{X}\times\mathcal{Y}$) defines C_{xy} to be the probability that y is observed given that the secret is currently x, where recall that we use $\mathcal{Y}$ for the type of observations. A Hidden Markov Model step is also a stochastic matrix, and is determined by first a Channel step followed by a Markov step, denoted here by $(C{:}M)$. The execution of the Markov step is independent of the observation, and so $(C{:}M)_{xyx'} {:=} C_{xy} \times M_{xx'}$ (where "$\times$" here means multiplication). In general we write $\mathcal{A} \multimap \mathcal{B}$ for the type of stochastic matrix over $\mathcal{A}\times\mathcal{B}$, so that rows are labelled by type $\mathcal{A}$. Thus $M{:}\, \mathcal{X} \multimap \mathcal{X}$, $C{:}\, \mathcal{X} \multimap \mathcal{Y}$ and $(C{:}M){:}\, \mathcal{X} \multimap \mathcal{Y}\times\mathcal{X}$.

We can compose *HMM* steps to obtain the result of executing several leak-update steps one after another. Let $H^1{:}\, \mathcal{X} \multimap \mathcal{Y}^1\times\mathcal{X}$ and $H^2{:}\, \mathcal{X} \multimap \mathcal{Y}^2\times\mathcal{X}$. We define their composition $H^1; H^2{:}\, \mathcal{X} \multimap (\mathcal{Y}^1\times\mathcal{Y}^2)\times\mathcal{X}$, which now has observation type $\mathcal{Y}^1\times\mathcal{Y}^2$ so that information leaks accumulate.

$$(H^1; H^2)_{x(y_1,y_2)x'} \quad := \quad \sum_{x''{:}\,\mathcal{X}} {H^1}_{xy_1x''} \times {H^2}_{x''y_2x'} \ . \tag{1}$$

We use the term *HMM* to mean both an *HMM* step, and more generally some composition of steps. In the latter case, the observation type $\mathcal{Y}$ will actually consist of a product of observation types arising from the observations of the component steps. Given an *HMM* $H{:}\, \mathcal{X} \multimap \mathcal{Y}\times\mathcal{X}$ and an initial distribution $\pi{:}\, \mathbb{D}\mathcal{X}$ we write $(\pi\rangle H)$ for the joint distribution $\mathbb{D}(\mathcal{X}\times\mathcal{Y}\times\mathcal{X})$ defined by $(\pi\rangle H)_{xyx'} {:=} \pi_x \times H_{xyx'}$.

[6] This is the standard method of composing functions defined by a monad.

[7] A matrix is stochastic if its rows sum to 1.

It is the probability that the initial state was x, that the final state is now x' and that the adversary observed y. Similarly, as special cases, we write $(\pi\rangle C)$ and $(\pi\rangle M)$ for the result of applying a prior π to respectively a pure Channel and a pure Markov.

In the next sections (§3.2 and §4) we describe a modification of Def. 1 based on *HMM*'s; it focusses on tracking correlations between initial and final states. We begin by illustrating how the loop body of Fig. 1 can be represented as an *HMM*-style matrix. Recall its definition as a program fragment:

$$\texttt{leak}\ (\texttt{X} > \texttt{1});\quad \texttt{X} := \texttt{X} - \texttt{2}\ . \tag{2}$$

The first statement of (2) –`leak (X > 1)`– corresponds to a channel matrix in $\mathcal{X}\mathbin{\multimap}\mathcal{Y}$, where the observation type $\mathcal{Y}$ consists of two values, one for when the secret is strictly greater than 1 and one where the secret is no more than 1.

$$C\colon\quad \begin{array}{c|cc} & \circ\texttt{G} & \circ\texttt{L} \\ \hline \texttt{0} & 0 & 1 \\ \texttt{1} & 0 & 1 \\ \texttt{2} & 1 & 0 \\ \texttt{3} & 1 & 0 \end{array}$$

The labels $\circ\texttt{G}, \circ\texttt{L}$ denote the observations that `X` is strictly greater than 1, or no more than 1 respectively. C_{xy} is the chance that y will be observed given that the secret is x. Observe that this is a deterministic channel.

If $\pi\colon\mathbb{D}\mathcal{X}$ is a prior distribution over $\mathcal{X}$ we create a joint distribution in $\mathbb{D}(\mathcal{X}{\times}\mathcal{Y})$ defined by $(\pi\rangle C)$. In our example, we take π to the the uniform distribution over $\{0,1,2,3\}$; for each observation, we learn something about this initial value of `X` — if $\circ\texttt{G}$ is observed, then we can use Bayesian reasoning to compute the residual uncertainty of the secret. It is the conditional distribution over $\mathcal{X}$ of $(\pi\rangle C)$ given the observation $\circ\texttt{G}$; we call this posterior $(\pi\rangle C)^{\circ\texttt{G}}$. If $\circ\texttt{L}$ is observed instead we can similarly define the posterior $(\pi\rangle C)^{\circ\texttt{L}}$ as the conditional distribution over $\mathcal{X}$ of $(\pi\rangle C)$ given the observation $\circ\texttt{L}$. Notice that both posteriors occur with probability 1/2.

The second statement of (2) is only executed in a context when $\texttt{X} > \texttt{1}$ and therefore is equivalent to `if (X>1) then X:= X-2`. It corresponds to a Markov matrix $\mathcal{X}\mathbin{\multimap}\mathcal{X}$:

$$M\colon\quad \begin{array}{c|cccc} & \texttt{0} & \texttt{1} & \texttt{2} & \texttt{3} \\ \hline \texttt{0} & 1 & 0 & 0 & 0 \\ \texttt{1} & 0 & 1 & 0 & 0 \\ \texttt{2} & 1 & 0 & 0 & 0 \\ \texttt{3} & 0 & 1 & 0 & 0 \end{array}$$

Each entry of the Markov matrix $M_{xx'}$ provides the probability that the final state will be x' given that it is x initially. Note that it is impossible for x' to be either 2 or 3.

The combination of the two statements yields an *HMM* to form the composition $(C{:}M)\colon\mathcal{X}\mathbin{\multimap}\mathcal{Y}{\times}\mathcal{X}$, where recall that $(C{:}M)_{xyx'}{:=}C_{xy}{\times}M_{xx'}$ (where "$\times$" here means multiplication). For our example y is one of the observations $\circ\texttt{G}$ or $\circ\texttt{L}$. The combination as an *HMM* matrix becomes:

$$(C{:}M):\quad \begin{array}{c}\\ \\ 0\\1\\2\\3\end{array}\begin{array}{cccccccc} \multicolumn{4}{c}{\overbrace{\hphantom{0\;\;0\;\;0\;\;0}}^{\circ\mathtt{G}}} & \multicolumn{4}{c}{\overbrace{\hphantom{0\;\;0\;\;0\;\;0}}^{\circ\mathtt{L}}}\\ 0&1&2&3&0&1&2&3\\ \end{array}$$

$$\begin{array}{c|cccccccc} & \multicolumn{4}{c}{\circ\mathtt{G}} & \multicolumn{4}{c}{\circ\mathtt{L}}\\ & 0&1&2&3&0&1&2&3\\ \hline 0 & 0&0&0&0&1&0&0&0\\ 1 & 0&0&0&0&0&1&0&0\\ 2 & 1&0&0&0&0&0&0&0\\ 3 & 0&1&0&0&0&0&0&0 \end{array}$$

The labels $\circ\mathtt{G}$ and $\circ\mathtt{L}$ denote the observations corresponding to those from C, and the other column labels come from the column labels in M. Thus each column is labelled by a pair in $\{\circ\mathtt{G}, \circ\mathtt{L}\}\times\mathcal{X}$.

Notice that the rows are *not* identical, because M updates the state in a way dependent on its incoming value.

Consider now the initial (uniform) prior $\pi{:}\,\mathbb{D}\mathcal{X}$ combined with the matrix $(C{:}M)$ above. The combination is a joint distribution $(\pi\rangle(C{:}M))$ of type $\mathbb{D}(\mathcal{X}{\times}\mathcal{Y}{\times}\mathcal{X})$. We can now take the conditional probability with respect to an observation $y{:}\,\mathcal{Y}$ to obtain the corresponding residual uncertainty over the correlation in $\mathbb{D}(\mathcal{X}^2)$ between the initial and final state of `X`. For example, given that $\circ\mathtt{G}$ was observed, the probability that the initial value of the secret was 2 and the final value is 0 is $1/4 \div 1/2 = 1/2$. The probability that the initial value of the secret was 3 and its final value is 0 is 0.

We summarise all the posterior distributions by forming the hyper-distribution $[\pi\rangle(C{:}M)]{:}\,\mathbb{D}^2\mathcal{X}^2$. The outers in our example are both 1/2, because each observation occurs with the same probability; the corresponding inners are distributions of correlations modelling the adversary's residual uncertainty. These correlations retain just enough detail about the relationship between initial and final states to explain the behaviour of (2) in arbitrary contexts. In particular $\delta{:}\,\mathbb{D}\mathcal{X}^2$ is in the support of $[\pi\rangle(C{:}M)]$ means that for some observation, the adversary can deduce the likelihood between the initial and final states of the secret, and from that the likelihood of its initial value, and the likelihood of its current value.

The hyper-distribution over correlations for our example at (2) is as follows.

$$[\pi\rangle(C{:}M)]:\quad \begin{array}{c|cc|cc|} & \multicolumn{2}{c}{\boxed{1/2}} & \multicolumn{2}{c}{\boxed{1/2}}\\ \hline 0 & 0&0 & 1/2&0\\ 1 & 0&0 & 0&1/2\\ 2 & 1/2&0 & 0&0\\ 3 & 0&1/2 & 0&0\\ \hline & 0&1 & 0&1 \end{array}$$

The labels along the outside of the boxes represent the possible initial states (`0,1,2,3` on the left column), and the possible final states (`0,1` twice along the bottom). The large boxes represent the two distinct posterior distributions (one for each observation in $\{\circ\mathtt{G}, \circ\mathtt{L}\}$), and the small boxes (1/2 each) are the marginal probabilities for each observation.

Notice that we no longer need labels of type $\mathcal{Y}$, in fact they have been replaced by outers (the small boxes containing 1/2 each). By comparing with $(C{:}M)$ above, we also see now that only the relevant effect of the observations has been preserved in $[\pi\rangle(C{:}M)]$ – for example there are no columns only containing zeros because they represent events that cannot occur. Only the relevant posteriors are retained, together with the chance that they are observed. For example with probability 1/2 the observer can now deduce when the initial state was in the set $\{\texttt{0},\texttt{0}\}$ or in the set $\{\texttt{2},\texttt{3}\}$.

Next we show that $[\pi\rangle(C{:}M)]$ is all that is required for computing the behaviour when we introduce a correlated fresh secret `Z`, as in this program fragment:

$$\texttt{Z} := \texttt{X};\ \ \texttt{leak}\ (\texttt{X} > \texttt{1});\ \ \texttt{X} := \texttt{X} - \texttt{2}\ . \tag{3}$$

Take the *HMM* $(C{:}M)$ above representing the program fragment at (2), but now consider executing it with extra secret `Z` as at (3). The initialisation of `Z` creates an interesting correlation between `X` and `Z` given by $\mathit{\Pi}^*{:}\,\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$ which is defined to be $\mathit{\Pi}^*_{zx}{:=}\,\pi_x$ if and only if $x = z$, where recall π is the uniform initialisation of `X`. We now compute the final joint distribution $(\mathit{\Pi}^*\rangle(C{:}M))$ of type $\mathbb{D}(\mathcal{Z}{\times}\mathcal{Y}{\times}\mathcal{X})$; in this case because of the definition of $\mathit{\Pi}^*$, it is

$$(\mathit{\Pi}^*\rangle(C{:}M))_{zyx'} \quad := \quad \mathit{\Pi}_{zz} \times (C{:}M)_{zyx'}\ ,$$

with corresponding hyper-distribution in $\mathbb{D}^2(\mathcal{Z}{\times}\mathcal{X})$ given as follows:

$$\begin{array}{rcc}
[\mathit{\Pi}^*\rangle(C{:}M)]: & \boxed{1/2} & \boxed{1/2} \\
 & & \\
\texttt{(0,0)} & 0 & 1/2 \\
\texttt{(1,1)} & 0 & 1/2 \\
\texttt{(2,0)} & 1/2 & 0 \\
\texttt{(3,1)} & 1/2 & 0
\end{array}$$

The labels along the outside of the boxes now represent the final posteriors in $\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$. Notice that the posteriors can be computed directly from $(\pi\rangle(C{:}M))$ as explained above. In fact, as we describe below, it can also be computed directly from the abstraction $[\pi\rangle(C{:}M)]$.

More generally we can study the behaviour of an *HMM* $H{:}\,\mathcal{X} \multimap \mathcal{Y}{\times}\mathcal{X}$ when it is executed in the context of some arbitrary correlation $\mathit{\Pi}{:}\,\mathcal{Z}{\times}\mathcal{X}$. The joint distribution $\mathit{\Pi}\rangle H$ is computed explicitly as

$$(\mathit{\Pi}\rangle H)_{zyx'} \quad := \quad \sum_{x{:}\,\mathcal{X}} \mathit{\Pi}_{zx}{\times}H_{xyx'}\ . \tag{4}$$

Just as in the special case above, we can calculate the associated hyper-distribution from H's posterior correlations on the initial and final value of the secret type $\mathcal{X}$. We first define a matrix $Z{:}\,\mathcal{Z} \multimap \mathcal{X}$ as $Z_{zx}{:=}\,\mathit{\Pi}_{zx}/\overrightarrow{\mathit{\Pi}}_x$, and $\overrightarrow{\mathit{\Pi}}^*{:}\,\mathbb{D}\mathcal{X}^2$ as:

$$\mathit{\Pi}^*{}_{xx'}{:=}\,\overrightarrow{\mathit{\Pi}}_x \qquad \textit{if and only if} \quad x = x'\ .$$

With these in place we have that $\mathit{\Pi} = Z \cdot \mathit{\Pi}^*$, where we are using $(Z\cdot)$ as matrix product, as in $(Z \cdot \mathit{\Upsilon})_{zx'}{:=}\,\sum_{x{:}\,\mathcal{X}} Z_{zx}{\times}\mathit{\Upsilon}_{xx'}$. We can now express (4) equivalently as an equation between hyper-distributions:

$$[\mathit{\Pi}\rangle H] \quad = \quad \mathbb{D}(Z\cdot)[\overrightarrow{\mathit{\Pi}}\rangle H]\ . \tag{5}$$

Notice that we have applied $\mathbb{D}$ to the function $(Z\cdot)$, since it must act on the inners of the hyper-distribution $[\overrightarrow{\Pi}\rangle H]$ to re-install correlations of type $\mathbb{D}(\mathcal{Z}\times\mathcal{X})$. But $[\overrightarrow{\Pi}\rangle H]$ is the special case of H applied to a prior in $\mathbb{D}\mathcal{X}$, which models the hyper-distribution of the correlation between the initial and final value of the secret (type $\mathcal{X}$). This allows us to define a family of liftings of *HMM*'s which, by construction, can be computed from hyper-distributions of the form $[\overrightarrow{\Pi}\rangle H]$.

Definition 2. *Given an HMM $H{:}\,\mathcal{X} \rightarrow\!\!\!\!\triangleright\, \mathcal{Y}\times\mathcal{X}$. We say that $\mathcal{X}$ is its mutable type; for fresh secret type $\mathcal{Z}$ called the correlation type define $[\![H]\!]^{\mathcal{Z}}{:}\,\mathbb{D}(\mathcal{Z}\times\mathcal{X}) \rightarrow \mathbb{D}^2(\mathcal{Z}\times\mathcal{X})$ as*

$$[\![H]\!]^{\mathcal{Z}}.\Pi \quad := \quad \mathbb{D}(Z\cdot)[\overrightarrow{\Pi}\rangle H]\ ,$$

where Z is defined relative to Π as at (5).

Def. 2 divides the secrets up into a mutable part $\mathcal{X}$ and correlated part $\mathcal{Z}$, where the former can have information leaked about it, and then subsequently be updated by H, whilst the latter cannot be changed (by H), but can have information about its value leaked through a correlation with $\mathcal{X}$. Our next definition, modifies Def. 1 to describe an abstract semantics taking the correlated part into account, consistent with the way a concrete *HMM* leaks information about correlated secrets. Our abstraction enforces the behaviour of the correlated secret to be determined by the behaviour of the mutable state as in Def. 2, via the commuting diagram summarised in Fig. 2.

Definition 3. *Given a state space $\mathcal{X}$ of hidden (i.e. high-security) variables, a context-aware denotational model of quantitative non-interference is a family of functions from prior (input) distribution correlations on $\mathbb{D}(\mathcal{Z}\times\mathcal{X})$ to (output) hyper-distributions on the same space, where $\mathcal{Z}$ is any correlated type, and $\mathcal{X}$ is the mutable type. For a given correlated type $\mathcal{Z}$, function $h^{\mathcal{Z}}$ has type $\mathbb{D}(\mathcal{Z}\times\mathcal{X}){\rightarrow}\mathbb{D}^2(\mathcal{Z}\times\mathcal{X})$. Moreover $h^{\mathcal{X}}$ and $h^{\mathcal{Z}}$ must satisfy the commuting diagram in Fig. 2.*

We define the composition of $h_1^{\mathcal{Z}}, h_2^{\mathcal{Z}}$ to be $h_1^{\mathcal{Z}}; h_2^{\mathcal{Z}} = \mathsf{avg} \circ \mathbb{D}h_2^{\mathcal{Z}} \circ h_1^{\mathcal{Z}}$.

Fig. 2 describes the commuting diagram which captures exactly the effect on collateral secrets as described by the concrete situation of *HMM*'s at (5). The function $h^{\mathcal{X}}$ can preserve the correlation between the initial and final values of the mutable variables, after which the the correlated variable can be reinstalled by applying $\mathbb{D}(Z\cdot)$ to the hyper-distribution $\mathbb{D}^2\mathcal{X}^2$.

The next lemma shows that the standard Giry composition also satisfies the healthiness condition of Fig. 2.

Lemma 1. *Let $h_{1,2}^{\mathcal{Z}}{:}\,\mathbb{D}(\mathcal{Z}\times\mathcal{X}) \rightarrow \mathbb{D}^2(\mathcal{Z}\times\mathcal{X})$ satisfy the commuting diagram in Fig. 2. Then $h_1^{\mathcal{Z}}; h_2^{\mathcal{Z}}$ also satisfies it.*

Proof. Let $\Pi{:}\,\mathbb{D}(\mathcal{Z}\times\mathcal{X})$ and Z, π^* be defined as in Fig. 2 so that $\Pi = Z\cdot\pi^*$. We now reason:

At left: Given a correlation $\mathit{\Pi}{:}\,\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$ define a matrix $Z{:}\,\mathcal{Z}{\times}\mathcal{X}$ and $\mathcal{X}$-prior $\pi{:}\,\mathbb{D}\mathcal{X}$ to be such that $\mathit{\Pi}_{z,x} = Z_{z,x}\pi_x$. This is arranged so that $\mathit{\Pi} = Z{\cdot}\pi^*$ where $\pi^*{:}\,\mathbb{D}\mathcal{X}^2$ with $\pi^*_{x'x} = \pi_x$ iff $x = x'$.

$$\begin{tikzcd}
\mathbb{D}(\mathcal{Z}{\times}\mathcal{X}) \arrow[r, "h^{\mathcal{Z}}"] & \mathbb{D}(\,\mathbb{D}(\mathcal{Z}{\times}\mathcal{X}')\,) & \\
 & & \mathbb{D}(\mathcal{Z}{\times}\mathcal{X}') \\
\mathbb{D}(\mathcal{X}{\times}\mathcal{X}) \arrow[uu, "Z\cdot"'] \arrow[r, "h^{\mathcal{X}}"] & \mathbb{D}(\,\overbrace{\mathbb{D}(\mathcal{X}{\times}\mathcal{X}')}\,) \arrow[uu, "\mathbb{D}(Z\cdot)"] \arrow[ur, "Z\cdot"'] &
\end{tikzcd}$$

At right (in grey): For an initial/final distribution $\delta{:}\,\mathbb{D}(\mathcal{X}{\times}\mathcal{X}')$, the left-multiplication $Z{\cdot}\delta$ produces a distribution in $\mathbb{D}(\mathcal{Z}{\times}\mathcal{X}')$, just as matrix multiplication would (with δ as a matrix of probabilities). *At right (in black):* The $\mathbb{D}$-lifting (push forward) of the multiplication $(Z\cdot)$ thus takes an initial-final hyper in $\mathbb{D}^2(\mathcal{X}{\times}\mathcal{X}')$ to a hyper in $\mathbb{D}^2(\mathcal{Z}{\times}\mathcal{X}')$.

Summary: A collateral `Z` is linked to our state `X` by joint distribution $\mathit{\Pi}{:}\,\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$. This $\mathit{\Pi}$ can be decomposed into its right marginal $\pi{:}\,\mathcal{X}$ on our state space, and a "collateral stochastic channel matrix" $Z{:}\,\mathcal{Z}{\leftarrow}\mathcal{X}$ between it and $\mathcal{Z}$, i.e a right conditional of $\mathit{\Pi}$. For each x the matrix Z gives the conditional distribution over $\mathcal{Z}$, as in "the probability that `Z` is some value, given that `X` is x". The original joint distribution $\mathit{\Pi}$ is restored from π and Z by matrix multiplication. Since π is not presented as a matrix, but Z is, we use the notation $Z\langle\pi$ to reconstruct $\mathit{\Pi}$ from the components. (Note that right-conditionals are not necessarily unique; but the variation on x's where $\pi.x{=}0$ does not affect $\mathbb{D}(Z\cdot)$ at right.)

Fig. 2. Healthiness condition for h: general collateral correlation $\mathit{\Pi}{:}\,\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$ can be computed from the effect of h on initial and final states.

$$\begin{array}{ll@{\qquad}r}
 & (h^{\mathcal{Z}}_1; h^{\mathcal{Z}}_2).\mathit{\Pi} & \\
= & \mathsf{avg}\circ\mathbb{D}h^{\mathcal{Z}}_2\circ h^{\mathcal{Z}}_1.\mathit{\Pi} & \text{“Def. 3”} \\
= & \mathsf{avg}\circ\mathbb{D}h^{\mathcal{Z}}_2\circ\mathbb{D}(Z\cdot)\circ h^{\mathcal{X}}_1.\pi^* & \text{“Fig. 2 for } h_1\text{: } h^{\mathcal{Z}}_1.\mathit{\Pi} = \mathbb{D}(Z\cdot)\circ h^{\mathcal{X}}_1.\pi^*\text{”} \\
= & \mathsf{avg}\circ\mathbb{D}(h^{\mathcal{Z}}_2\circ(Z\cdot))\circ h^{\mathcal{X}}_1.\pi^* & \text{“Function composition: } \mathbb{D}(f\circ g) = \mathbb{D}f\circ\mathbb{D}g\text{”} \\
= & \mathsf{avg}\circ\mathbb{D}(\mathbb{D}(Z\cdot)\circ h^{\mathcal{X}}_2)\circ h^{\mathcal{X}}_1.\pi^* & \text{“Fig. 2 for } h_2\text{: } h^{\mathcal{Z}}_2\circ(Z\cdot) = \mathbb{D}(Z\cdot)\circ h^{\mathcal{X}}_2\text{”} \\
= & \mathsf{avg}\circ\mathbb{D}^2(Z\cdot)\circ\mathbb{D}h^{\mathcal{X}}_2\circ h^{\mathcal{X}}_1.\pi^* & \text{“Function composition”} \\
= & \mathbb{D}(Z\cdot)\circ\mathsf{avg}\circ\mathbb{D}h^{\mathcal{X}}_2\circ h^{\mathcal{X}}_1.\pi^* & \text{“Monad law: } \mathsf{avg}\circ\mathbb{D}^2 f = \mathbb{D}f\circ\mathsf{avg}\text{”} \\
= & \mathbb{D}(Z\cdot)\circ(h^{\mathcal{X}}_2; h^{\mathcal{X}}_1).\pi^*\ . & \text{“Def. 3”}
\end{array}$$

By construction, the action of *HMM*'s defined above at Def. 2 satisfy the commuting diagram of Fig. 2, because for π^* defined in Fig. 2 we have

$$[\![H]\!]^{\mathcal{X}}.\pi^* \quad = \quad [\overrightarrow{\pi^*}\rangle H]\ . \tag{6}$$

Finally we note that *HMM* composition given above at (1) is consistent with the abstract semantics.

Lemma 2. *Let $H^{1,2}{:}\,\mathcal{X}\rightarrowtriangle\mathcal{Y}{\times}\mathcal{X}$ be HMM's, with mutable type $\mathcal{X}$; further let $\mathcal{Z}$ be any collateral type. Then $[\![H^1;H^2]\!]^{\mathcal{Z}} = [\![H^1]\!]^{\mathcal{Z}};[\![H^2]\!]^{\mathcal{Z}}$, where composition of HMM's (inside $[\![\cdot]\!]$) is defined at (1), and composition in the semantics (outside $[\![\cdot]\!]$) is defined at Def. 3.*

Proof. This follows from [16][Thm12] for HMM's generally where we set the secret type explicitly to be $\mathcal{Z}\times\mathcal{X}$ (rather than just $\mathcal{X}$).

In this section we have shown how to describe an abstract semantics for programs based on viewing *HMM*'s as correlation transformers, generalising our previous work [16]. We have identified in Fig. 2 how the behaviour of the correlation transformer $h^{\mathcal{X}}$ determines the behaviour of $h^{\mathcal{Z}}$ when $\mathcal{X}$ is the mutable type and $\mathcal{Z}$ is the correlation type. This provides a general abstract account of how programs modelled as *HMM*'s update and leak information about secrets.

So far we have not defined a refinement order on abstract *HMM*'s which takes information flow into account. We do that next.

4 Generalising entropy: secure refinement

We quantify our ignorance of hidden variables' unknown exact value using *uncertainty measures* over their (known) distribution, a generalisation of (e.g.) Shannon entropy and others [1, 2, 13, 18, 24]. These measures are continuous, concave functions in $\mathbb{D}\mathcal{X}{\rightarrow}\mathbb{R}$ [16]. With them, programs' security behaviours can be compared wrt. the average uncertainty of their final (probabilistic) state when run from the same prior distribution; and for programs that don't update their state (e.g. the information channels of Shannon, intensively studied in current security research), the *amount of information* flowing due to a single program's execution can be measured by looking for a *change* in uncertainty, i.e. by comparing the program's prior uncertainty with its average posterior uncertainty. Such comparisons between uncertainties are used to define secure refinement.

It is assumed that the adversary knows the program text (and for us this usually means some *HMM*), and that he observes the values emitted by (for example) `leak` statements as described above. Given a hyper-distribution produced by some program, each inner is a posterior distribution having some uncertainty; and the (outer) probability of that inner represents the probability with which that uncertainty occurs.

For general $\mathcal{S}$, a distribution $\sigma{:}\,\mathbb{D}\mathcal{S}$ and a real-valued (measurable) function $f{:}\,\mathcal{S}{\rightarrow}\mathbb{R}$, we write $\mathcal{E}_\sigma(f)$ for the expected value of f over σ. Typical cases are when $\mathcal{S}{=}\mathcal{X}$ and $f{:}\,\mathcal{X}{\rightarrow}\mathbb{R}$ is over the initial state, and when $\mathcal{S}{=}\mathbb{D}\mathcal{X}$ and we are taking expected values of some $f{:}\,\mathbb{D}\mathcal{X}{\rightarrow}\mathbb{R}$ over an output hyper in $\mathbb{D}^2\mathcal{X}$: in that case $\mathcal{E}_\Delta(\texttt{se})$ would e.g. be the *conditional* Shannon Entropy of a hyper Δ, where $\texttt{se}.\pi = -\sum_{x{:}\,\mathcal{X}} \pi_x \log(\pi_x)$.

An important class of uncertainty measures, more appropriate for security applications than Shannon entropy alone, are the "loss functions" [16].

Definition 4. *A* loss function *ℓ is of type $I{\rightarrow}\mathcal{X}{\rightarrow}\mathbb{R}$ for some index set I, with the intuitive meaning that $\ell.i.x$ is the cost to the adversary of using "attack strategy" i when the hidden value turns out actually to be x. Her expected cost for* *an attack planned but not yet carried out is then* $\mathcal{E}_\delta(\ell.i)$ *if* δ *is the distribution in* $\mathbb{D}\mathcal{X}$ *she knows to be governing* x *currently.*[8]

From such an ℓ *we define an uncertainty measure* $U_\ell(\rho) := \inf_{i:\,I}\ \mathcal{E}_\rho(\ell.i)$ *. When* I *is finite, the* inf *can be replaced by* min.

The inf represents a rational strategy where the adversary minimises her cost or risk under the ignorance expressed by her knowing only the distribution ρ and not an exact value: she will choose the strategy i whose expected cost $\mathcal{E}_\rho(\ell.i)$ is the least. If ρ is the prior δ, then $U_\ell(\delta)$ is her expected cost if attacking without running the program, i.e. she attacks the input. If she does run the program, producing output hyper $\Delta{=}[\![P]\!].\delta$, then her expected cost is $\mathcal{E}_\Delta(U_\ell)$; here she attacks the program's output, taking advantage of the observations she made as the program ran.

For example, consider the following loss function for which the index set I is the same as $\mathcal{X}$, and the adversary is trying to guess the secret. If she chooses some i which turns out to be the same as the secret, then her losses are 0, otherwise if her guess is wrong, then she loses \$1. Formalised, this becomes:

$$\ell.i.x := (0\ \underline{\text{if}}\ i{=}x\ \underline{\text{else}}\ 1)\ . \tag{7}$$

In Fig. 1 if the prior $\pi{:}\,\mathbb{D}\mathcal{X}$ is uniform over `X` then $U_\ell(\pi) = 3/4$, since whichever i is picked there is only $1/4$ chance that it is equal to the value of the variable. After executing the program however, the hyper-distribution Δ on `X` alone has a single posterior which assigns equal probability to 0 or 1, thus $\mathcal{E}_\Delta(U_\ell) = 1/2$, showing that the adversary is better able to guess the secret after executing the program than she was before.

Loss functions have been studied extensively elsewhere [2], where they have been shown to describe more accurately than Shannon entropy the adversary's intentions and losses versus benefits involved in attacks [24]. The crucial property of the derived uncertainty measures is that they are *concave functions* of distributions — this feature embodies the idea that when information is leaked then the losses to the attacker will be reduced. Thus if C is Channel matrix, and I is the identity Markov matrix then we always obtain the inequality:

$$\mathcal{E}_{[\![(C:I)]\!]^{\mathcal{Z}}.\Pi} U_\ell \quad \leq \quad U_\ell(\Pi) \tag{8}$$

where the expression on the left gives the adversary's losses relative to the release of information through C and loss function ℓ, and on the right are the losses without any release of information.

In other work [13, 18, 16] we have shown that loss functions (equivalently their dual "gain functions") are sufficient to determine hyper-distributions, that is (remarkably) that if we know $\mathcal{E}_\Delta(U_\ell)$ for all ℓ then we know Δ itself.[9] They therefore define a "secure refinement" relation between programs (Def. 5 below), based on their output hyper-distributions. Any program that, for all loss functions, can only cost more for the adversary, never less, is regarded as being more secure:

[8] Here $\ell.i$ is the function $\ell(i)$ of type $\mathcal{X}{\to}\mathbb{R}$ — we are using Currying.

[9] This was called the *Coriaceous Conjecture* in [2].

Definition 5. [13] *Let $H^{1,2}$ be programs represented by HMM's in $\mathcal{X} \twoheadrightarrow \mathcal{Y}{\times}\mathcal{X}$ so that $\mathcal{X}$ is the mutable type. We say that $H^1{\sqsubseteq}H^2$ just when, for any correlated type $\mathcal{Z}$ we have $\mathcal{E}_{[\![H^1]\!]^{\mathcal{Z}}.\Pi}(U_\ell) \leq \mathcal{E}_{[\![H^2]\!]^{\mathcal{Z}}.\Pi}(U_\ell)$ for all priors $\Pi{:}\, \mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$ and loss functions ℓ on $\mathcal{Z}{\times}\mathcal{X}$.*

Notice that Def. 5 captures both functional and information flow properties: when the loss function is derived from a single choice, it behaves like a standard "probabilistic predicate" and the refinement relation for this subset of loss functions reduces to the well known functional refinement of probabilistic programs [12] .

In order to determine when $H_1 \sqsubseteq H_2$ it might seem as though all contexts need to be considered. Fortunately the healthiness condition summarised in Lem. 2 means that general refinement in all correlation contexts can follow from local reasoning relative to $[\![H]\!]^{\mathcal{X}}$. *Context-aware refinement* is defined with respect only to correlations between initial and final states.

Definition 6. *Let $H^{1,2}$ be two HMM's both with mutable type $\mathcal{X}$. We say that $H^1 \tilde{\sqsubseteq} H^2$, whenever $\mathcal{E}_{[\![H^1]\!]^{\mathcal{X}}.\delta}(U_\ell) \leq \mathcal{E}_{[\![H^2]\!]^{\mathcal{X}}.\delta}(U_\ell)$ for all $\delta{:}\, \mathbb{D}\mathcal{X}^2$ and $\ell{:}\, I{\rightarrow}\mathcal{X}^2{\rightarrow}\mathbb{R}$.*

Our principal monotonicity result concerns state extension: it shows that context-aware refinement is preserved within any context — even if fresh variables have been declared.

Theorem 1. *Let $H^{1,2}$ be HMM's with mutable type $\mathcal{X}$. Then*

$$H^1 \tilde{\sqsubseteq} H^2 \quad \textit{iff} \quad H^1 \sqsubseteq H^2 \ .$$

Proof. If $H^1 \sqsubseteq H^2$ holds then it is clear that $H^1 \tilde{\sqsubseteq} H^2$.

Alternatively, we observe that from Fig. 2 we deduce that $\mathbb{D}(Z\cdot) \circ [\![H]\!]^{\mathcal{X}}.\pi^* = [\![H]\!]^{\mathcal{Z}}.\Pi$, where Z, π^* are determined by Π. This means that for any $\delta{:}\, \mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$ in the support of $[\![H]\!]^{\mathcal{Z}}.\Pi$, there is a corresponding $\delta^*{:}\, \mathbb{D}\mathcal{X}^2$ in the support of $\mathbb{D}(Z\cdot) \circ [\![H]\!]^{\mathcal{X}}.\pi^*$ such that $\delta = Z \cdot \delta^*$.

Now given $\ell{:}\, I \rightarrow \mathcal{Z}{\times}\mathcal{X} \rightarrow \mathbb{R}$ we calculate:

$$\begin{array}{lll}
 & \sum_{x',z} \ell.i.z.x' \times \delta_{zx'} & \\
= & \sum_{x',z} \ell.i.z.x' \times \sum_x Z_{zx} \times \delta^*_{xx'} & \text{“}\delta = Z \cdot \delta^*\text{”} \\
= & \sum_{x',x,z} \ell.i.z.x' \times Z_{zx} \times \delta^*_{xx'} & \textit{“Rearrange”} \\
= & \sum_{x',x} \sum_z (\ell.i.z.x' \times Z_{zx}) \times \delta^*_{xx'} & \textit{“Rearrange”} \\
= & \sum_{x',x} \ell^*.i.x.x' \times \delta^*_{xx'} \ . & \textit{“Define } \ell^*.i.x.x' {:=} \sum_z \ell.i.z.x' {\times} Z_{zx}\text{”}
\end{array}$$

We see now that $\mathcal{E}_{[\![H]\!]^{\mathcal{X}}.\pi^*}(U_{\ell^*}) = \mathcal{E}_{[\![H]\!]^{\mathcal{Z}}.\Pi}(U_\ell)$, where on the left we have an expression that only involves the mutable type. Thus if $H^1 \not\sqsubseteq H^2$ we can find some ℓ and some $\mathcal{Z}$ such that $\mathcal{E}_{[\![H^1]\!]^{\mathcal{Z}}.\Pi}(U_\ell) > \mathcal{E}_{[\![H^2]\!]^{\mathcal{Z}}.\Pi}(U_\ell)$ which, by the above, means that $\mathcal{E}_{[\![H^1]\!]^{\mathcal{X}}.\pi^*}(U_{\ell^*}) > \mathcal{E}_{[\![H^2]\!]^{\mathcal{X}}.\pi^*}(U_{\ell^*})$ implying that $H^1 \tilde{\not\sqsubseteq} H^2$.

Thm. 1 now restores the crucial monotonicity result for reasoning about information flow for sequential programs modelled as *HMM*'s. In particular it removes the quantification over all priors in $\mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$, replacing it with a quantification over all priors in $\mathbb{D}\mathcal{X}^2$. We comment on how this applies to practical program analysis in §6.

5 Some algebraic inequalities

In this section we illustrate some useful algebraic laws for security-aware programs modelled as abstract *HMM*'s denoted by Def. 2, and equality determined by the partial order at Def. 5.

Given $\delta^1, \delta^2 {:}\, \mathbb{D}\mathcal{S}$ we define convex summation for distributions by $\delta^1 {}_p{+}\, \delta^2$, also in $\mathbb{D}\mathcal{S}$, as $(\delta^1 {}_p{+}\, \delta^2)_s {:=} \delta^1{}_s{\times}p + \delta^2{}_s{\times}(1{-}p)$, where $0 \le p \le 1$. Similarly, given $\varDelta^1, \varDelta^2 {:}\, \mathbb{D}^2\mathcal{S}$ we define convex summation between hyper-distributions as $\varDelta^1 {}_p{\oplus}\, \varDelta^2$, also in $\mathbb{D}^2\mathcal{S}$, as $(\varDelta^1 {}_p{\oplus}\, \varDelta^2)_\delta {:=} \varDelta^1{}_\delta{\times}p + \varDelta^2{}_\delta{\times}(1{-}p)$.

5.1 Basic laws for information flow

We write $h {:}\, \mathbb{D}(\mathcal{Z}{\times}\mathcal{X}) \to \mathbb{D}^2(\mathcal{Z}{\times}\mathcal{X})$, where $\mathcal{X}$ is the mutable type and $\mathcal{Z}$ is some correlated type. The laws in Thm. 2 describe some basic monotonicity relationships between *HMM*'s. (1) says that if there is more information available in the prior, then there will be more information flow. Similarly (2) says that if all the observations are suppressed, then less information flows: recall that avg applied to a hyper-distribution averages the inners (in our case the posteriors) and therefore summarises the state updates only. (3–4) say that refinement is preserved by sequential composition. Finally (5) says that if both h^1, h^2 simply release information but don't update the state, then the order in which that information is released is irrelevant.

Theorem 2. *Let h, h^1, h^2 be instances of HMM's respectively $[\![H]\!]^{\mathcal{Z}}, [\![H_1]\!]^{\mathcal{Z}}, [\![H_2]\!]^{\mathcal{Z}}$ with mutable type $\mathcal{X}$ and correlated type $\mathcal{Z}$. Further, let $\varPi {:}\, \mathbb{D}(\mathcal{Z}{\times}\mathcal{X})$, and $0 \le p \le 1$. The following refinements hold.*

1. $h.\varPi \;{}_p{\oplus}\; h.\varPi' \quad \sqsubseteq \quad h.(\varPi \;{}_p{+}\; \varPi')$
2. $h \sqsubseteq \eta \circ \mathsf{avg} \circ h$
3. $h^1 \sqsubseteq h^2$ *implies* $h; h^1 \sqsubseteq h; h^2$
4. $h^1 \sqsubseteq h^2$ *implies* $h^1; h \sqsubseteq h^2; h$
5. *If* h^1, h^2 *correspond to channels, then* $h^1; h^2 = h^2; h^1$.

Proof. (1–4) have appeared elsewhere for an HMM model without collateral variables (see [16, 13] for example), and the proof here is similar for each possible correlated state, and relies on the concavity of loss functions. (5) also follows directly from the definition of channels.

5.2 Information flows concerning the collateral state

In some circumstances we can summarise simply the behaviour of a complex *HMM* matrix formed by sequentially composing some number of leak-update steps. We look at two cases here, and both result in summarising the overall effect as a single step of an *HMM*, i.e. as a leak of information concerning the mutable type, followed by a Markov update.

Let H be an *HMM* matrix with mutable type $\mathcal{X}$, and recall $\mathsf{dup}\colon \mathcal{X} \to \mathcal{X}^2$ is defined by $\mathsf{dup}.x = (x, x)$. Now define $\mathsf{chn}.[\![H]\!]^{\mathcal{X}}\colon \mathbb{D}(\mathcal{X}^2) \to \mathbb{D}^2\mathcal{X}^2$

$$\mathsf{chn}.[\![H]\!]^{\mathcal{X}} \quad := \quad \mathbb{D}^2(\mathsf{dup}) \circ \mathbb{D}(\overleftarrow{\cdot}) \circ [\![H]\!]^{\mathcal{X}} \ , \tag{9}$$

which ignores the update of the final state, and records the information flow concerning the initial state only [17].

Similarly we can define the overall Markov state change $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}\colon \mathbb{D}(\mathcal{X}^2) \to \mathbb{D}^2\mathcal{X}^2$, which simply ignores the information flow. Its output is therefore a point distribution in $\mathbb{D}^2\mathcal{X}$:

$$\mathsf{mkv}.[\![H]\!]^{\mathcal{X}} \quad := \quad \eta \circ \mathsf{avg} \circ [\![H]\!]^{\mathcal{X}} \ . \tag{10}$$

If the effect of an *HMM* can be summarised as its associated channel followed by its associated Markov update then it can be written in the form $\mathsf{chn}.[\![H]\!]^{\mathcal{X}}; \mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$. Next we illustrate two circumstances when this can (almost) happen.

We say that $\mathsf{chn}.[\![H]\!]^{\mathcal{X}}$ is *standard* if it does not leak information probabilistically. For example the leak statement in Fig. 1 is standard — informally this means any information it does leak is not "noisy" and corresponds to the adversary deducing exactly some predicate. We can express standard leaks equationally by saying that if we leak the information (about the initial state) first, and then run the program, we learn nothing more — this is not true if the information released is noisy because each time a noisy channel is executed, a little more information is released. Thus non-probabilistic leaks have associated channel satisfying the following:

$$\mathsf{chn}.[\![H]\!]^{\mathcal{X}}; [\![H]\!]^{\mathcal{X}} \quad = \quad [\![H]\!]^{\mathcal{X}} \ . \tag{11}$$

Similarly we say that $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$ is *standard* if the relation between the initial and final values for all inners in $[\![H]\!]^{\mathcal{X}}$ is functional, which can be expressed equationally as:

$$\mathsf{mkv}.[\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \circ (\overleftarrow{\cdot}).\delta \quad = \quad \eta(\delta) \ , \tag{12}$$

for any δ in the support of $[\![H]\!]^{\mathcal{X}} \circ \mathbb{D}\mathsf{dup}.\pi$.

Thm. 3 says that if either $\mathsf{chn}.[\![H]\!]^{\mathcal{X}}$ or $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$ is standard then H is refined by a leak step followed by a Markov update. In the latter case where $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$ is standard the refinement goes both ways.

Theorem 3. *Let H be an HMM with mutable type $\mathcal{X}$.*

1. *If $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$ is standard, then*
   $[\![H]\!]^{\mathcal{X}} \circ \mathbb{D}\mathsf{dup} = (\mathsf{chn}.[\![H]\!]^{\mathcal{X}}; \mathsf{mkv}.[\![H]\!]^{\mathcal{X}}) \circ \mathbb{D}\mathsf{dup}$

2. *If $\mathsf{chn}.[\![H]\!]^{\mathcal{X}}$ is standard, then*
   $[\![H]\!]^{\mathcal{X}} \circ \mathbb{D}\mathsf{dup} \mathrel{\widetilde{\sqsubseteq}} (\mathsf{chn}.[\![H]\!]^{\mathcal{X}}; \mathsf{mkv}.[\![H]\!]^{\mathcal{X}}) \circ \mathbb{D}\mathsf{dup}$ [10]

[10] We overload $\widetilde{\sqsubseteq}$ defined on *HMM*'s directly to be defined similarly for the abstract semantics: $h^1 \widetilde{\sqsubseteq} h^2$ of type $\mathbb{D}\mathcal{X}^2 \to \mathbb{D}^2\mathcal{X}^2$ if $\mathcal{E}_{h^1(\delta)}(U_\ell) \leq \mathcal{E}_{h^2(\delta)}(U_\ell)$ for all ℓ.

Proof. *Suppose that* $\mathsf{mkv}.[\![H]\!]^{\mathcal{X}}$ *is standard. We reason as follows:*

$$\begin{array}{ll}
 & (\mathit{chn}.[\![H]\!]^{\mathcal{X}}; \mathit{mkv}.[\![H]\!]^{\mathcal{X}}) \circ \mathbb{D}(\mathsf{dup}) \\
= & \mathit{avg} \circ \mathbb{D}(\mathit{mkv}.[\![H]\!]^{\mathcal{X}}) \circ \mathbb{D}^2(\mathsf{dup}) \circ \mathbb{D}(\overleftarrow{\cdot}) \circ [\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \qquad \text{“Def. 3 and (9)”} \\
= & \qquad \text{“Function composition: } \mathbb{D}(f \circ g) = \mathbb{D}f \circ \mathbb{D}g\text{”} \\
 & \mathit{avg} \circ \mathbb{D}(\mathit{mkv}.[\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \circ (\overleftarrow{\cdot})) \circ [\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \\
= & \mathit{avg} \circ \mathbb{D}(\eta) \circ [\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \qquad \text{“(12)”} \\
= & [\![H]\!]^{\mathcal{X}} \circ \mathbb{D}(\mathsf{dup}) \ . \qquad \text{“Monad law: } \mathit{avg} \circ \mathbb{D}(\eta) \text{ is the identity”}
\end{array}$$

Now suppose that $\mathit{chn}.[\![H]\!]^{\mathcal{X}}$ *is standard. We reason as follows*

$$\begin{array}{ll}
 & [\![H]\!]^{\mathcal{X}} \\
= & \mathit{chn}.[\![H]\!]^{\mathcal{X}}; [\![H]\!]^{\mathcal{X}} \qquad \text{“(11)”} \\
\widetilde{\sqsubseteq} & \mathit{chn}.[\![H]\!]^{\mathcal{X}}; \mathit{mkv}.[\![H]\!]^{\mathcal{X}} \ . \qquad \text{“Thm. 2(2), (10) and (3)”}
\end{array}$$

Recall our program in Fig. 2 — since the change to X is functional, it means that overall the *HMM* model for the loop is standard in its Markov component. Thus by Thm. 3 (2) we can summarise its behaviour as a single *HMM*-style step, which we can also write as

$$\mathtt{leak}(\mathtt{X} \div 2); \mathtt{X} := \mathtt{X} - (\mathtt{X} \bmod 2) \ . \qquad (13)$$

The inclusion of the `leak` statement now ensures that the possible impact on third-parties is now accurately recorded.

6 Related work and discussion

In this paper we have studied an abstract semantic model suitable for reasoning about information flow in a general sequential programming framework. A particular innovation is to use hyper-distributions over correlations of initial and final states. Hyper-distributions summarise the basic idea in quantitative information flow that the *value* of the observation is not important, but only the effect it induces on change of uncertainty wrt. the secret. An important aspect is that our context-aware refinement order means that local reasoning is now sufficient to deduce that the behaviours of `leak (X) ; X:=0` are not the same as those of `X:=0`: even though all confidentiality properties concerning *only* the final value of `X` are the same in both program fragments. This is because they leak differing kinds of information about he initial state, and this could become significant when the program fragments are executed within contexts containing fresh secrets correlated with `X`.

We have illustrated the model by proving some algebraic properties; further work is required to develop the equational theory, and to apply it to a semantics for a general programming language.

Classical analyses of quantitative information flow assume that the secret does not change, and early approaches to measuring insecurities in programs are based on determining a "change in uncertainty" of some "prior" value of the secret — although how to measure the uncertainty differs in each approach. For example Clark et al [4] use Shannon entropy to estimate the number of bits being leaked; and Clarkson et al [5] model a change in belief. Smith [24] demonstrated the importance of using measures that have some operational significance, and the idea was developed further [2] by introducing the notion of g-leakage to express such significance in a very general way. The partial order used here on programs is the same as the g-leakage order introduced by Alvim et al [2], but it appeared also in even earlier work [13]. Its properties have been studied extensively [1].

Others have investigated information flow for dynamic secrets, for example Marzdiel et al [11] use probabilistic automata. Our recent work similarly explored dynamic secrets, but allows only a single secret type $\mathcal{X}$ [13, 16].

The abstract treatment of probabilistic systems with the introduction of a "refinement order" was originally due to the probabilistic powerdomain of Jones and Plotkin [10]; and those ideas were extended to include demonic nondeterminism (as well as probability) by us [22]. In both cases the order (on programs) corresponds to an order determined by averaging over "probabilistic predicates" which are random variables over the state space. The compositional refinement order for information flow appeared in [13] for security programs expressed in a simple programming language and in [1] for a channel model.

Our work here is essentially the Dalenius scenario presented in a programming-language context where `X` is the statistical database and the correlation with `Z` is "auxiliary information" [7] except that, unlike in the traditional presentation, ours allows the "database" (the password) to be updated. This model can be thought of as a basis for developing a full semantics for context-aware refinement for a programming language with the aim of reasoning about and developing information flow analysis which is valid generally for all operating scenarios.

References

1. Mário S. Alvim, Konstantinos Chatzikokolakis, Annabelle McIver, Carroll Morgan, Catuscia Palamidessi, and Geoffrey Smith. Additive and multiplicative notions of leakage, and their capacities. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 308–322. IEEE, 2014.
2. Mário S. Alvim, Kostas Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF 2012)*, pages 265–279, June 2012.
3. R.-J.R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer, 1998.
4. David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3):238–251, 2001.
5. Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 31–45, 2005.

6. T. Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15:429–44, 1977.
7. Cynthia Dwork. Differential privacy. In *Proc. 33rd International Colloquium on Automata, Languages, and Programming (ICALP 2006)*, pages 1–12, 2006.
8. M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85. Springer, 1981.
9. J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp on Security and Privacy*, pages 75–86. IEEE Computer Society, 1984.
10. C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the IEEE 4th Annual Symposium on Logic in Computer Science*, pages 186–95, Los Alamitos, Calif., 1989. Computer Society Press.
11. Piotr Mardziel, Mário S. Alvim, Michael W. Hicks, and Michael R. Clarkson. Quantifying information flow for dynamic secrets. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 540–555, 2014.
12. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Tech Mono Comp Sci. Springer, New York, 2005.
13. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Compositional closure for Bayes Risk in probabilistic noninterference. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming: Part II*, volume 6199 of *ICALP'10*, pages 223–235, Berlin, Heidelberg, 2010. Springer.
14. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In *Proc. LiCS 2012*, 2012.
15. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Hidden-Markov program algebra with iteration. *Mathematical Structures in Computer Science*, 2014.
16. Annabelle McIver, Carroll Morgan, and Tahiry Rabehaja. Abstract Hidden Markov Models: a monadic account of quantitative information flow. In *Proc. LiCS 2015*, 2015.
17. Annabelle McIver, Carroll Morgan, Tahiry Rabehaja, and Nico Bordenabe. Reasoning about distributed secrets. Submitted to FORTE 2017.
18. Annabelle McIver, Carroll Morgan, Geoffrey Smith, Barbara Espinoza, and Larissa Meinicke. Abstract channels and their robust information-leakage ordering. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2014.
19. E. Moggi. Computational lambda-calculus and monads. In *Proc. 4th Symp. LiCS*, pages 14–23, 1989.
20. C.C. Morgan. *Programming from Specifications.* Prentice-Hall, second edition, 1994. `web.comlab.ox.ac.uk/oucl/publications/books/PfS/`.
21. C.C. Morgan. *The Shadow Knows:* Refinement of ignorance in sequential programs. In T. Uustalu, editor, *Math Prog Construction*, volume 4014 of *Springer*, pages 359–78. Springer, 2006. Treats *Dining Cryptographers.*
22. C.C. Morgan, A.K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Trans Prog Lang Sys*, 18(3):325–53, May 1996. `doi.acm.org/10.1145/229542.229547`.

23. Tom Schrijvers and Carroll Morgan. `Hypers.hs` Haskell code implementing quantitative non-interference monadic security semantics.
`http://www.cse.unsw.edu.au/`$\sim$`carrollm/Hypers.pdf`, 2015.
24. Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *Proc. 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS '09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302, 2009.