

Security, probability and nearly fair coins in the Cryptographers' Café

Annabelle McIver¹, Larissa Meinicke¹, and Carroll Morgan^{2*}

¹ Dept. Computer Science, Macquarie University, NSW 2109 Australia

² School of Comp. Sci. and Eng., Univ. New South Wales, NSW 2052 Australia

Abstract. Security and probability are both artefacts that we hope to bring increasingly within the reach of refinement-based Formal Methods; although we have worked on them separately, in the past, the goal has always been to bring them together.

In this report we describe our ongoing work in that direction: we relate it to a well known problem in security, Chaum's *Dining Cryptographers*, where the various criteria of correctness that might apply to it expose precisely the issues we have found to be significant in our efforts to deal with security, probability and abstraction all at once.

Taking our conviction into this unfamiliar and demanding territory, that abstraction and refinement are the key tools of software development, has turned out to be an exciting challenge.

1 Introduction

When I took office, only high-energy physicists had ever heard of what is called the World Wide Web... Now even my cat has its own page.

More than 10 years later, the internet is the playground of far stranger things than White-House pets,³ and some estimates place its size upwards of tens of billions of websites: everybody and every thing seems to be at it. There has been a boom not only in science and business but also in private transactions, so that where once decisions were based on “rules of thumb” and inter-personal relationships, now we increasingly defer to automation — even when it is our security and privacy that is at stake.

The movement towards “transactions at a distance” has spawned a huge number of ingenious algorithms and protocols to make computers do for us what we used to do for ourselves, using “gut feeling” and the full repertoire of body language and other out-of-band channels. The reliance on machines has blurred the distinction between the real world —with all its uncertainties— and software, with its inescapable reliance on an exact unforgiving model — one that its designers hope has some connection to its environment and purpose. As users thus we place an increasingly heavy burden on those designers: once they

* We acknowledge the support of the Australian Research Council Grant DP0879529.

³ The quote is attributed to Bill Clinton in 1996 (but unconfirmed).

could assume absolute regularity; and now they must decide just what exactly *can* be assumed in an uncertain world, and what guarantees can be given with any confidence about the software they create.

The answer –we believe in Formal Methods – is to create rigorous models of the phenomena with which the software must contend, as far as is possible. As we do so, we are misunderstood by the wider Computer Science community when they accuse us of believing that the real world can be captured entirely in mathematics, or that the only way to write programs is to calculate them by hand with a quill pen.

It is as if planning a household budget using decimals for dollars, and a computer for calculation, were somehow evidence of one's belief that all of family life can be reduced to arithmetic. The truth –as everyone knows– is that using arithmetic for what can be quantified leaves more time to enjoy what can't be.

Similarly, formalising and automating what can be captured in semantics, models and calculi for computer applications leaves more time for concentrating on the aspects that require intuition, common sense and good taste. Thus our aim is always to expand our formalisations so that we can capture more.⁴

In this paper we concentrate on capturing more aspects of software systems for which security is an issue and whose operational environment is only partially predictable. Specifically, our goal is a mathematical model of probability, security and modularity: they are –we will argue– closely linked phenomena, and (we do not need to argue) they are all relevant to the design of computer systems and the control and understanding of their behaviour and their applications.

Crucial to getting a grip on slippery concepts is to setting up a framework within which one can pose precise questions and draw rigorous conclusions. If the conclusions agree with our intuitions, or nearly so, then we proceed, elaborating the framework; if not, we backtrack and try a different angle. Constructing a framework requires, in turn, a repertoire of small, compelling examples that can serve as targets for the construction.

Our compelling example is Chaum's well known exemplary problem of the Dining Cryptographers: we stretch it and twist it in order to illustrate the intricacies that arise when we mix the above concepts. From there we discuss a number of models which attempt to tame those intricacies, and finally we pose some further intriguing lines of research.

⁴ Assailed on both sides, Formal Methods is sometimes thought to be trying to reinvent or trivialise the long-established work of mathematicians. It's not so. Nobody for example worries about the correctness of Bubble Sort *in principle*; but because source-level reasoning about array segments remains difficult for many people, every day new and incorrect sorting programs are written by accident... and propagated: that's bad *practice*. Finding better ways to reason about array segments at the source level would therefore be an improvement for everyone.

2 The Cryptographers' Café

In this section we introduce Chaum's *Dining Cryptographers* protocol, and we outline the point of view that will allow us to use it to explore refinement and nondeterminism as well as the security and probability it already contains. Here is the original abstract, which sketches the problem domain:

Keeping confidential who sends which messages, in a world where any physical transmission can be traced to its origin, seems impossible. The solution presented here is unconditionally or cryptographically secure, depending on whether it is based on one-time-use keys or on public keys, respectively. It can be adapted to address efficiently a wide variety of practical considerations. [8]

Now we look at the solution, and see where it leads.

2.1 Chaum's solution: The Dining Cryptographers' Protocol

Chaum's protocol for anonymity is built around the following story:

Three cryptographers have finished their lunch, and ask the waiter for their bill; the waiter says it's already been paid. But by whom? The cryptographers do the following to find out.

Each pair of cryptographers flips a fair coin privately between the two of them, concealed from the third; then each announces whether she paid, (1) but lies if the two flipped coins she sees show different faces.

If the number of "I paid" utterances is odd, then indeed one of the cryptographers paid; but only she knows which one that was (i.e. herself). If the number is even, then the bill was paid by a fourth party.

Chaum gives a rigorous mathematical proof that, no matter what an individual cryptographer observes, because the coins are fair (50%) the chance she can guess which other cryptographer paid *after* the protocol (50%) is the same as it was before (50%).

His proof does not refer to a specific program text, and so makes no use of anything that could be called a program logic. Our contribution is *not* however to increase the rigour of Chaum's argument. It is rigorous already. Rather we want to find a way of capturing that style of argument in a programming-language context so that we can be more confident when we confront larger, more complex examples of it in actual applications, and so that we have some chance of automating it.

As a first step, we separate concerns by dealing with the hiding and the probability separately. We name the cryptographers A , B and C ; then

Cryptographer A 's utterance does not reveal whether she paid, since neither of the other two cryptographers B, C knows both of the coins A consulted when deciding whether to lie or not: Cryptographer B sees one but not the other; and Cryptographer C sees the other but not the one. The same applies from Cryptographers' B, C points of view. (2)

Yet when the parity of the three utterances is calculated, each coin is tallied twice and so cancels out. Thus the parity of "I paid" *claims* is the same as the parity of "I paid" *acts*. That number of acts is either zero or one (because the bill is paid at most once, no matter by whom) — and so it is determined by the parity.

The attractiveness of this formulation is that it reduces the problem to Boolean algebra, a domain much closer to programming logic. But it raises some intriguing questions, as we now see.

2.2 What is the relevance of the fair coins?

Given that the statement of the protocol (1) refers to "fair coins," and our version of the correctness argument (2) does not, it's reasonable to ask where in our framework the fairness should fit in. Clearly the coins' bias has *some* bearing, since in the extreme case —returning always heads, or always tails— the cryptographer's utterances won't be encrypted at all: with each coin's outcome known in advance, whether or not the coin can be seen, a cryptographer's utterance will reveal her act. In that case the protocol is clearly incorrect.

But what if the coins are biased only a little bit? At what point of bias does the protocol become incorrect? If it's correct only when the coins are exactly fair, and incorrect otherwise, is the protocol therefore unimplementable? Or does it become "increasingly incorrect" as the bias increases? After all, there are no exactly fair coins in nature.

And worse — what about the case where the coins *are* extremely biased, but no-one knows beforehand what that bias is? Furthermore, does it make a difference if the bias is not fixed and is able to change with time?

2.3 The Café

The key to the questions of Sec. 2.2 —and our point of departure— is *time* in the sense that it implies repeated trials. We elaborate the example of Sec. 2.1 by imagining that the cryptographers meet in the same café every day, and that the protocol (1) is carried out on each occasion. Suppose that the waitress, observing this, notes that Cryptographer A *always* says "I paid."⁵ What does she conclude?

⁵ More precisely, she serves a statistically significant number of lunches, say a whole year's worth, and Cryptographer A says "I paid" on all of them. This is discussed further in Sec. 8.10.

She concludes quite a lot. Given that the coins are independent of each other (irrespective of whether they are fair), and that Cryptographer A follows the protocol properly (whether she believes she paid is not influenced *post hoc* by the coin-flips), the waitress concludes that Cryptographer A either pays every time, or never pays. Moreover, she knows also that both of the coins Cryptographer A sees are completely unfair. For this is the only way, given independence (and in the absence of magic) that the utterance of A can be the same every time. Note that the waitress does *not* learn whether A is a philanthropist or a miser; she learns only that A is resolutely one or the other. Thus –in spite of (2)– without an explicit appeal to fairness the protocol seems insecure.

Let's push this example further, and examine its dependence on the coins' bias when that bias is not extreme. For simplicity, we consider just one hidden coin c and a secret h : the coin is flipped, and $c=h$ is revealed.⁶ What do we learn about c and h if this is done repeatedly?

From now on, for uniformity, our coins show **true** and **false** (sometimes abbreviated **T** and **F**). Over many runs, there will be some proportion p_c of c 's being **T**; similarly there will have been some proportion p_h of h 's being **T**. The resulting proportion of $c=h$'s being **true** will then be

$$p_t := p_c p_h + (1-p_c)(1-p_h) , \quad (3)$$

and arithmetic shows that p_t is $1/2$ just when one or both of $p_{\{c,h\}}$ is $1/2$.

If p_t is *not* $1/2$, then it will be some distance away so that $2p_t$ is $1+\varepsilon_t$, say, and for $\varepsilon \geq 0$ we could say that an ε -biased coin has a probability range of ε , the interval $[(1-\varepsilon)/2, (1+\varepsilon)/2]$, an interval which we abbreviate as just $[\varepsilon]$. Defining $\varepsilon_c, \varepsilon_h$ similarly, we can bring (3) into a much neater form, that

$$\varepsilon_t = \varepsilon_c \varepsilon_h . \quad (4)$$

Then, since $|\varepsilon_{\{t,c,h\}}| \leq 1$, we can see immediately that $|\varepsilon_t| \leq |\varepsilon_{\{c,h\}}|$. That is, the observed bias $|\varepsilon_t|$ is a lower bound for the hidden biases $|\varepsilon_c|$ and $|\varepsilon_h|$. Note particularly that whatever $|\varepsilon_t|$ we observe we can infer a *no lesser* bias $|\varepsilon_h|$ in the hidden h without knowing anything about the bias (or not) of the coin c .

That last sentence gives us a glimpse of the continuum we were looking for: if there is no observed bias, i.e. ε_t is zero, then one of $\varepsilon_{\{c,h\}}$ must be zero also and we learn nothing about the other one. On the other hand, if $c=h$ always or never, so that ε_t is one, then *both* $\varepsilon_{\{c,h\}}$ must be one as well.

Between the extremes of exactly fair ($\varepsilon=0$) and wholly nondeterministic ($\varepsilon=1$), however, we have coins of varying quality. What can we conclude about those?

2.4 The challenge

The series of examples and questions above are intended to reveal what we consider to be important and significant challenges for any formal method operating

⁶ This is an example of what elsewhere we call the Encryption Lemma.

in the domain of probability, nondeterminism and security. In Sec. 8 to come we will return to the Café and say more about what outcomes we expect from a formal analysis built on a model that is practical enough to use in the construction of actual software. Between now and then, we (re-)visit the underlying themes that have led us to this point.

3 Underlying themes: a trio of features and their history

Program models and logics applicable to problems like those above will include demonic choice, probabilistic choice and hidden variables. Rather than attempting to integrate these features at once we have approached the problem in smaller, overlapping steps. A benefit is a thorough and incremental exploration of the different modelling issues; another benefit is that some of the less feature-rich models are adequate for modelling certain classes of problems, and so it has been worthwhile to explore these models in their own right.

One of the first steps taken towards the integration of demonic choice, probabilistic choice and hidden variables was to build a framework [41, 59] combining the first two of these three features. Another has been the development of the *Shadow Model* [37, 44, 49, 54–56] for programs with demonic choices and hidden state (but not probability).

A third development, which we report in this paper, may be seen as a hybrid of the above two models in which the value of the hidden variables may be unknown, but must vary according to a known distribution. This model, the *Quantum Model*, does contain nondeterminism; but this nondeterminism is “visible,” whereas uncertainty about the value of the hidden state can only be probabilistic and not nondeterministic.

In the following sections we start by giving an overview of all of those frameworks: the basic model, the Shadow Model and the Quantum Model. We conclude with a discussion of the challenges posed by integrating the three and, on that basis, we suggest another incremental way to move forward.

4 The basics: Probability and Demonic Choice⁷

Demonic choice in sequential programming was conspicuously popularised by Dijkstra [14]: it was perhaps the first elementary appearance of nondeterminism at the source-code level, and was intended to express abstraction in the design process.⁸ As its importance and relevance increased, it acquired an operator of its own, so that nowadays we simply write *this* \square *that* for pure demonic choice, whereas Dijkstra would have written

```

if true  $\rightarrow$  this
  | true  $\rightarrow$  that
fi .

```

⁷ This section is very brief, given the availability of other literature [6, 7, 17, 21, 23–27, 29, 39–42, 21, 45–48, 52, 53, 58, 59, 64, 65].

⁸ And it was often misunderstood to be advocating nondeterminism at run-time.

A natural refinement of the qualitative unpredictable choice \sqcap is the *quantitatively* unpredictable choice $_p\oplus$, so that *this* $_p\oplus$ *that* chooses *this* with probability p and (therefore) *that* with probability $1-p$. This *replacement* of \sqcap by $_p\oplus$ was given a Dijkstra-like semantics by Kozen [34, 35] and, later, put in a more general setting by Jones [32, 31].

Our own contribution was to treat both forms of nondeterminism –demonic and probabilistic– within the same framework, thus simultaneously generalising the two approaches above [41, 59]. With this in place, we can describe a “nearly fair choice” by using both forms of nondeterminism together. For example, the behaviour of a program *this* $_{[\varepsilon]}\oplus$ *that* which describes a coin “within ε of being fair” may be represented by

$$(this \frac{1-\varepsilon}{2} \oplus that) \sqcap (this \frac{1+\varepsilon}{2} \oplus that) .$$

This begins to address the issue of nearly fair coins that was first raised in Sec. 2.2 and to which we return in Sec. 8.

5 The Shadow: Hidden Nondeterministic State⁹

5.1 The Shadow in essence

The Shadow Model arose in order to allow development of non-interference - style protocols by refinement. A long-standing problem in the area was that nondeterminism in a specification usually is an invitation to the developer (or the run-time system) to pick one alternative out of many presented: the intention of offering the choice is to indicate that any of the alternatives is acceptable.

In security however nondeterminism indicates instead a hidden, or high-security value known only to be in a set; and an implementation’s picking just one of those values is exactly what the specification is not allowing. Rather it is specifying exactly the opposite, that the “ignorance” of the unknown value should *not* be decreased.

In the Shadow Model these two conflicting uses of nondeterminism are dealt with by partitioning the state space into *visible* variables $v: \mathcal{V}$, say, and *hidden* variables $h: \mathcal{H}$ and storing the hidden part as a set of the possible values it could be, rather than as a single definitive value — that is, the state space is $\mathcal{V} \times \mathbb{P}\mathcal{H}$ rather than the conventional $\mathcal{V} \times \mathcal{H}$.¹⁰ Then (informally) conventional refinement allows whole sets of hidden variables’ values to be discarded; but it does not allow the sets themselves to be reduced.

For example, for the program

$$v: \in \{0, 1\}; h: \in \{v+1, v+2\} \tag{5}$$

the denotation in the style $\mathcal{V} \times \mathbb{P}\mathcal{H}$ as above of the final states would be the set of pairs $\{(0, \{0, 1\}), (1, \{1, 2\})\}$; and a possible refinement then would be to throw

⁹ Also this section is brief: there are two other detailed papers at this venue on The Shadow [44, 55].

¹⁰ We write \mathbb{P} for powerset.

the whole first pair away, leaving just the $\{(1, \{1, 2\})\}$ which is the denotation of the program $v:=1; h:\in\{v+1, v+2\}$. On the other hand, it is not a refinement simply to remove values from the h -sets: the denotation $\{(0, \{1\}), (1, \{2\})\}$ of the program $v:\in\{0, 1\}; h:=v+1$ is not a refinement of (5).

With this Shadow Model one can prove that the Dining Cryptographers Protocol is secure for a single run, in the sense that it is a secure refinement of the specification “reveal whether a cryptographer paid, and nothing else.” (See Sec. 8.1 for a more detailed discussion of this and, in particular, for the significance of “single run.”)

5.2 The Shadow’s abstraction of probability

The Shadow is a *qualitative* semantics for security and hiding, indicating only what is hidden and what its possible values are. In practice, hidden values are chosen *quantitatively* according to distributions that are intended to minimise the probability that an attacker can determine what the actual hidden variables are.

In the cases where a quantitative analysis is necessary, an obvious step is to replace the encapsulated set $\mathbb{P}\mathcal{H}$ of hidden values by instead an encapsulated distribution $\mathbb{D}\mathcal{H}$ over the same base type, so that the state space becomes $\mathcal{V}\times\mathbb{D}\mathcal{H}$. That leads conceptually to what we call The Quantum Model, explained in the next section.

6 The Quantum Model:¹¹ Hidden probabilistic state and Ignorant nondeterminism

6.1 Motivation

Our work on probabilistic/demonic semantics, summarised above in Sec. 4, achieved a natural generalisation of its antecedents: its operational (relational) model was $S\rightarrow\mathbb{P}\mathbb{D}S$, generalising both the purely demonic model $S\rightarrow\mathbb{P}S$ and the purely probabilistic model $S\rightarrow\mathbb{D}S$;¹² and its corresponding “expectation transformer” model $(S\rightarrow\mathbb{R})\rightarrow(S\rightarrow\mathbb{R})$ generalised the predicate-transformer model $(S\rightarrow\{0, 1\})\rightarrow(S\rightarrow\{0, 1\})$,¹³ including the discovery of the quantitative form of conjunctivity, called sublinearity [41, p.146].

¹¹ Initial experiments with this are described in unpublished notes [57, 38].

We said “conceptually” above because in fact the Quantum Model was developed ten years before The Shadow, in response to issues to do with data refinement as explained in Sec. 6.2 below. Because of the model’s complexity, however, it was decided to “put it on ice” and seek an intermediate, simpler step in the hope that it would make the issues clearer. We alter the fossil record by placing the Shadow before the Quantum in this presentation.

¹² We write \mathbb{D} for “distributions over.”

¹³ The predicate-transformer model is usually described as $\mathbb{P}S\rightarrow\mathbb{P}S$. We used the above equivalent formulation to make the connection clear with the more general model.

<pre> module <i>Fork</i> { // No data. public <i>Flip</i> { $g := T \sqcap F$ } } </pre> <p style="text-align: center;">(a) The Demonic Fork</p>	<pre> module <i>Spade</i> { private $h := T \sqcap F$; public <i>Flip</i> { $g := h; h := T \sqcap F$ } } </pre> <p style="text-align: center;">(b) The Demonic Spade</p>
---	--

Fig. 1. *The Fork* has no data. *The Spade* has a local variable h initialised demonically.

Much of that work was achieved by “playing out” the initial insights of Kozen and Jones [34, 35, 32, 31] along the the familiar trajectory that Dijkstra [14] and others had established for purely probabilistic and purely demonic programs respectively. An unexpected obstacle arose however when we attempted to continue that trajectory along the path of data refinement [33, 4, 19]. We found that the modular probabilistic programs require the same program features –such as the ability to hide information– as security applications do.

We explain the data-refinement problem in the following section; it leads directly to the Quantum Model.

6.2 The Fork and the Spade¹⁴

Consider the two modules shown in Fig. 1. It is well known that the left-hand module *Fork* can be transformed via data-refinement into the right-hand module *Spade*; we just use the representation transformer $h := T \sqcap F$.¹⁵

The algebraic inequality (actually identity) that justifies this is

$$g := T \sqcap F; h := T \sqcap F \sqsubseteq h := T \sqcap F; g := h; h := T \sqcap F,$$

that is that $Op_F; rep \sqsubseteq rep; Op_S$ where Op_F is the Fork’s operation and Op_S is the Spade’s operation and rep is the representation transformer [4, 19, 5]. Now we can carry out the same transformation (structurally) if we replace demonic choice \sqcap by probabilistic choice ${}_p\oplus$ throughout, as in Fig. 2. This time the justifying inequality is

$$g := T {}_p\oplus F; h := T {}_p\oplus F \sqsubseteq h := T {}_p\oplus F; g := h; h := T {}_p\oplus F. \quad (6)$$

Unfortunately, in the probabilistic case the (6)-like inequality fails in general for statements in the *rest* of the program, whatever they might be. For example, consider a statement $g' := T \sqcap F$ in the surrounding program, for some second global variable g' distinct from g . The justifying (6)-like inequation here is the requirement

$$g' := T \sqcap F; h := T {}_p\oplus F \quad (7)$$

$$\sqsubseteq h := T {}_p\oplus F; g' := T \sqcap F. \quad (8)$$

¹⁴ These two examples were prominent players in the Oxford-Manchester “football matches” on data-refinement in 1986 [2].

¹⁵ We are using the formulation of data-refinement in which the “coupling invariant” is given by a program rather than a predicate [4, 19]: here we call it the *representation transformer*.

<pre> module <i>Fork</i> { // No data. public <i>Flip</i> { $g := T_{p \oplus F}$ } } </pre> <p style="text-align: center;">(a) The Probabilistic Fork</p>	<pre> module <i>Spade</i> { private $h := T_{p \oplus F}$; public <i>Flip</i> { $g := h; h := T_{p \oplus F}$ } } </pre> <p style="text-align: center;">(b) The Probabilistic Spade</p>
---	--

Fig. 2. This time *The Spade* has a local variable h initialised probabilistically.

And this putative refinement fails, for for the following reasons.

In our basic semantics of Sec. 4, on the right-hand side (8) we can have g' and h finally equal on every run, since the demonic assignment to g' can be resolved to $g := h$. But leaving g', h equal every time is *not* a possible behaviour of the left-hand side (7), since the demon's assignment to g' occurs before h 's final value has been chosen.

This failure is not acceptable. It would mean that altering the interior of a module could not be done without checking every statement in the surrounding program, however large that program might be. The point of having modules is to avoid precisely that.

6.3 Encapsulated distributions, and the Quantum Model

The key idea for solving the data-transformation problems exemplified by the failed refinement (7) $\not\sqsubseteq$ (8) is to “encapsulate” the local variable h in a distribution, one to which nondeterminism related to the visible variable v does not have access. Thus where the state-space normally would be $\mathcal{V} \times \mathcal{H}$, the encapsulated state-space would be $S := \mathcal{V} \times \mathbb{D}\mathcal{H}$. On top of this we introduce the basic construction for probability and nondeterminism Sec. 4, which then gives $S \rightarrow \mathbb{P}\mathbb{D}S$ as our model: putting them together therefore gives $(\mathcal{V} \times \mathbb{D}\mathcal{H}) \rightarrow \mathbb{P}\mathbb{D}(\mathcal{V} \times \mathbb{D}\mathcal{H})$ overall as the semantic type of a program with visible- and local variables.

From now on local variables will be called *hidden* variables. Here is an example of how the above could work.

In the classical model $(\mathcal{V} \times \mathcal{H}) \rightarrow \mathbb{P}\mathbb{D}(\mathcal{V} \times \mathcal{H})$ the program $h := T_{p \oplus} h := F$ produces from initial state (v_0, h_0) the singleton-set outcome $\{(v_0, T)_{p \oplus} (v_0, F)\}$, i.e. a set containing exactly one distribution; it's a singleton set because there's no demonic choice. A subsequent program $v := T \sqcap v := F$ then acts first on the distribution's two points (v_0, T) and (v_0, F) separately, producing for each a further pair of point distributions: we get $\{(\overline{T}, \overline{T}), (\overline{F}, \overline{T})\}$ in the first case and $\{(\overline{T}, \overline{F}), (\overline{F}, \overline{F})\}$ in the second.¹⁶ Those two sets are then interpolated Cartesian-wise, after the fact, using the same $p \oplus$ that acted originally between the two initial points. That gives finally the four-element set

$$\{(T, T)_{p \oplus} (T, F), (T, T)_{p \oplus} (F, F), (F, T)_{p \oplus} (T, F), (F, T)_{p \oplus} (F, F)\},$$

¹⁶ We use \bar{x} for the *point distribution* assigning probability one to x and (therefore) zero to everything else.

that is a four-way demonic choice between particular distributions.¹⁷ The third of those four demonic choices is the distribution $(F, T)_p \oplus (T, F)$ in which the final values of v, h are guaranteed to be different; the second on the other hand is $(T, T)_p \oplus (F, F)$ where they are guaranteed to be the same. The demon has access to both choices (as well as the other two remaining), and in this way can treat a testing-inspired postcondition $v=h$ with complete contempt, forcing the equality to yield **true** or **false** exactly as it pleases. The operational explanation we give (have given) for this is that the demon, acting second, “can see” the value of h produced first and then can target, or avoid it at will.

Now we contrast this with the encapsulated “quantum” view. This time the model is $(\mathcal{V} \times \mathbb{D}\mathcal{H}) \rightarrow \mathbb{P}\mathbb{D}(\mathcal{V} \times \mathbb{D}\mathcal{H})$ and the program $h := T_p \oplus h := F$ produces from initial state $(v_0, \overline{h_0})$ the singleton-set outcome $\{(v_0, \overline{T_p \oplus F})\}$ again containing just one distribution: but crucially this time it is a *point* distribution: the proper (non-point) probabilistic choice over h is still “suspended” further inside. Given a point distribution, the subsequent program $v := T \sqcap v := F$ has only one point to act on, giving the set $\{(T, \overline{T_p \oplus F}), (F, \overline{T_p \oplus F})\}$ and with no Cartesian-wise interpolation. Now if the postcondition is again $v=h$, then the demon’s attempt to minimise its probability of being true is limited to just two choices (no longer four), that is in deciding which of p and $1-p$ is the lesser and acting appropriately. For example if p were $3/4$, i.e. it is more likely that h is T , then the demon would choose the second pair (corresponding to having executed $v := F$), and the probability of $v=h$ would be only $1/4$. The demon can do this because, although it cannot see h , it can still see the program code and thus knows the value of p . In spite of that, it cannot make the $v=h$ -probability lower than $1/4$ and, in particular, it cannot make it zero as it did above.

With that prologue, we can now explain why we think of this as the “quantum model.”¹⁸

6.4 The act of observing collapses the distribution

We take $h := T_p \oplus h := F$ again as our first step, and in the quantum model (which we now call the above) we reach outcome $\{(v_0, \overline{T_p \oplus F})\}$. But now we take as our second step the command $v := h$ whereby the suspended “quantum state” $T_p \oplus F$ of h is “observed” by an assignment to the visible v . What happens?

In this model, the assignment $v := h$ “collapses” the quantum state, forcing it to resolve its suspended probabilistic choice one way or the other: the outcome overall is $\{(T, \overline{T})_p \oplus (F, \overline{F})\}$, now a *visible* probabilistic choice $_p \oplus$ in both of whose branches the value of h is completely determined: it’s given by the point distribution \overline{T} or by the point distribution \overline{F} .

¹⁷ In the basic model this includes as well, by convention, all interpolations of those four distributions.

¹⁸ And we must say right away that it is not at all (in our view) a real contender for a model of quantum computation, because for a start our probabilities are real-valued: for *qbits* they would instead have to be complex numbers. It does however have some features reminiscent of popular-science accounts of quantum mechanics, as we now see.

As a second example of quantum collapse, we consider the program

$$h := (0 \oplus 1 \oplus 2 \oplus 3); v := h \text{ mod } 2$$

in which the repeated \oplus 's indicate a uniform choice, in this case between four alternatives and so $1/4$ for each. Here the first command produces a fully encapsulated state as before; but the second statement collapses it only partially. The final outcome has denotation $\{(0, 0 \text{ }_{1/2} \oplus 2) \text{ }_{1/2} \oplus (1, 1 \text{ }_{1/2} \oplus 3)\}$, representing a program we could write

$$v := 0; h := 0 \text{ }_{1/2} \oplus 2 \text{ }_{1/2} \oplus v := 1; h := 1 \text{ }_{1/2} \oplus 3$$

in which a visible probabilistic choice $\text{ }_{1/2} \oplus$ occurs between the two possibilities for the low-order bit of h (0 or 1), but the value of the high-order bit is still suspended in the hidden, internal probabilistic choice.

For this program you can be sure that afterwards you will know the parity of h , but you cannot predict beforehand what that parity will be. And –either way– you still know nothing about the high-order bit.

7 The Generalised Quantum Model

However compelling (or not) the Quantum Model of Sec. 6.3 might be, it can only be a step on the way: in general, hidden variable h could be the subject of nondeterministic assignments as well as probabilistic ones; and the model $(\mathcal{V} \times \mathbb{D}\mathcal{H}) \rightarrow \mathbb{P}\mathbb{D}(\mathcal{V} \times \mathbb{D}\mathcal{H})$ is not rich enough to include that, since the encapsulated state for h is only $\mathbb{D}\mathcal{H}$.

The ability to specify nondeterministic and probabilistic assignments to h is especially important for real applications in which the source of randomness, used to conceal a hidden value, is not perfect. For example, the coins used by the Dining Cryptographers of Sec. 2 will probably be both demonic and probabilistic, since as we noted “fair coins” are ideal objects that do not exist in nature. Not only will a typical coin have a heads-probability of “nearly” (but not exactly) 50%, that probability could vary slightly depending on environmental conditions (e.g. ambient temperature): thus we must ultimately deal with not only accuracy but stability. The abstraction $\text{coin} := \top_{[\varepsilon]} \oplus \text{F}$, introduced in Secs. 2.3 & 4, describes that nicely: it’s a coin whose total deviation is ε , centred on being fair. The value ε , which represents the stability of the coin, may be treated as a parameter of the problem.

7.1 Preliminary investigations

Early investigations led us to suppose that the type of a Generalised Quantum model might be

$$\begin{array}{ccc} & \downarrow \textit{outer} & \\ (\mathcal{V} \times \mathbb{P}\mathbb{D}\mathcal{H}) & \rightarrow & \mathbb{P}\mathbb{D}(\mathcal{V} \times \mathbb{P}\mathbb{D}\mathcal{H}) \text{ ,} \\ & \uparrow \textit{inner} & \end{array} \tag{9}$$

in which hidden assignments, such as $h := A_{[\varepsilon]} \oplus B$ would be recorded in the “inner” \mathbb{PD} , and visible assignments, $v := A_{[\varepsilon]} \oplus B$, would be captured in the “outer” \mathbb{PD} . In this model interactions between nondeterminism and probability in such visible (or hidden assignments) would be treated as before: a statement $v := \top_{[\varepsilon]} \oplus F$, for example, is the nondeterministic choice over all probabilities $p \in [\varepsilon]$, that is $|2p-1| \leq \varepsilon$, of the pure probabilistic choice $v := \top_p \oplus F$. If all this were to be done successfully, we would be able to observe continuous range of behaviours as ε varies from 0 up to its maximum value 1: when ε is zero, as small as it can be, we have $(_{[0]} \oplus) = (_{1/2} \oplus)$, i.e. pure probabilistic choice. And when ε is 1, as large as it can be, we have $(_{[1]} \oplus) = (\top)$, pure demonic choice.

In spite of this promising beginning, the proposed fully generalised model (9) stretched our expertise: calculations in it were quite complex, and its logic was elusive [57, 38]. As a strategic move, therefore, we put that path aside and (eventually) decided to address a simpler, but related problem: with a final goal of encapsulated demonic/probabilistic state in mind, and a tentative model of encapsulated probability (alone, as above), it seemed reasonable to attempt a model of encapsulated demonic nondeterminism (alone). That became the Shadow Model of Sec. 5.¹⁹

In this remainder of this section we outline some of the challenges we face in generalising the Quantum model so that the encapsulated hidden state may be both probabilistic and nondeterministic.

7.2 The observers’ view: a difficulty

In a typical security application there may be multiple agents with different views of the system (that is, different partitions into visible and hidden). For example, in the Dining Cryptographers, the waitress has one perspective of the system, and Cryptographer A has another. Similarly, a system comprising many modules, each with different visibility requirements, may be thought of as being composed of multiple agents each one with her own view.

Both the Shadow and Quantum models may be used to model such applications from the perspective of any one of the agents, or *observers*, in the system. In these program models, the state space describes not just the visible state from that observer’s perspective, but what is known about the visible/hidden -state partition. How might we represent an observer’s perspective of the state in a model in which hidden variables may be set probabilistically *and* nondeterministically?

Unfortunately, our earlier guess (9) is too simplistic to specify some programs that we might be interested in. Take for example a system that is to be modelled

¹⁹ This is a revisionist history in a further sense, since the full encapsulated model –as a goal– was sitting happily on the shelf until Engelhardt and Moses, independently, encouraged us to look at their achievements –with van der Meyden– in dealing simultaneously with refinement and knowledge [16, 15].

from the perspective of an onlooker who can see variable v , but not h . What can that observer see, and deduce, after the operation

$$v := \top_{1/2} \oplus \text{F}; h := v \quad \sqcap \quad v := \top_{1/2} \oplus \text{F}; h := \neg v \quad (10)$$

in which both the \sqcap and the $_{1/2}\oplus$'s are hidden?

Since the \sqcap in particular is hidden and the outcome of the visible variable v is indistinguishable in both alternatives, the denotation of the program's result cannot use the outer \mathbb{P} to express that choice: we are limited to denotations of the form $\mathbb{D}(\mathcal{V} \times \mathbb{PDH})$. In fact (abusing notation) it must be of the form

$$\mathcal{V} \times \mathbb{PDH} \quad_{1/2}\oplus \quad \mathcal{V} \times \mathbb{PDH}$$

since the observed distribution of v between \top and F will be 50% no matter which branch is taken. What then are the candidates for the left- and the right \mathbb{PDH} ? It can't be $\{\bar{\top}, \bar{\text{F}}\}$ on both sides, since that loses the property of (10) that the h -values are also 50% distributed.

But how else might the observer describe what he can and cannot see about that operation within the program model proposed? It doesn't seem as though it can be done without either revealing too much about the value of the hidden variable, or retaining too little information about its distribution.

This suggests that a state space richer than $\mathcal{V} \times \mathbb{PDH}$ is going to be needed to describe an observer's view of the program state. That is one aspect of our current research.

7.3 Multiple agents: who makes the choice?

In The Shadow- and in The Quantum Model the state space is used to describe the behaviour of a program from the perspective of *one* of the agents in the system. Systems with multiple agents may thus be analysed by considering the system view of each agent separately. (The Quantum Model is limited in this respect since it cannot be used to model the perspective of any agent for which the hidden state is subject to nondeterminism.)

For example, take a system with Agent A , who can see variable a only, Agent B that can see variable b only and (implicitly) an anonymous observer who can't see either. Suppose A sets a nondeterministically, and then B does the same with b as in the program

$$\mathbf{vis}_A a; \mathbf{vis}_B b; \quad a: \in_A \{\top, \text{F}\}; \quad b: \in_B \{\top, \text{F}\},$$

where " \mathbf{vis}_A " means visible only to Agent A (and similar), and " $:\in_A$ " means chosen according to what Agent A can see.

With the Shadow Model, we see that from the point of view of Agent A the system is $\mathbf{vis} a; \mathbf{hid} b \dots$, i.e. with a visible and b hidden. To Agent B it is complementary, that is as if declared $\mathbf{hid} a; \mathbf{vis} b \dots$ — and for the anonymous observer it is $\mathbf{hid} a, b \dots$.

The reason that each agent may be treated separately in the Shadow semantics (which is defined using a “one-test run testing regime” — see Sec. 8.3 to come) is that we do not care (and cannot tell) whether nondeterministic choices are made independently or not. For example, when the Shadow is used to model the system from Agent A ’s perspective, we cannot tell whether the choice $b:\in_B \{\mathsf{T}, \mathsf{F}\}$ can be influenced by the value of a , and so as Agent A we do not need to know what Agent B ’s view of the system is. (The Quantum model avoids these issues altogether by disallowing hidden-state nondeterminism.)

Unfortunately, the situation is unlikely to be as simple in the Generalised Quantum model. Consider the similar system

$$\mathbf{vis}_A a; \mathbf{vis}_B b; \quad a := \mathsf{T} \text{ }_{1/2} \oplus \mathsf{F}; \quad b:\in_B \{\mathsf{T}, \mathsf{F}\}$$

in which the first choice is probabilistic. From A ’s point of view it is still **vis** a ; **hid** b but, in spite of the fact that A cannot see b it is still important to A whether the choice $b:\in_B \{\mathsf{T}, \mathsf{F}\}$ can depend on a . This is because the outcome of a subsequent code fragment **vis** a' ; $a' := a \oplus b$ is affected by that question.²⁰ That is, even from A ’s point of view it has become important what B can see, and that B makes the choice $b:\in_B \{\mathsf{T}, \mathsf{F}\}$ rather than for example $b:\in_A \{\mathsf{T}, \mathsf{F}\}$. For this reason it might not be possible to split our analysis neatly into separate views as we do in The Shadow. That is, if we would like to model probabilistic programs with multiple agents, and have so-called “nuanced” nondeterministic choices as above, controlled by different agents, then we will require a semantics that retains information about the different agent’s views of the system.

Instead of working directly on such a feature-rich program model, we suggest that a next step forward would be to build a model where hidden nondeterministic choices must be made with either total ignorance, *oblivious* choice, or knowledge of the entire state (*omniscient* choice). It would be sufficient in the dining cryptographers problem, for example, to model all the choices as being oblivious.

8 The Cryptographers’ Café: Reprise

Long ago (it must seem) in Sec. 2 we posed the problem of an accurate analysis of the Cryptographers’ Café; in the meantime we have examined a number of potential program models and background issues. We can note immediately that the basic probabilistic/demonic model of Sec. 4 is not rich enough to formalise our problem, because the coins must be hidden. And since the cryptographers’ coins involve both probability (because we suspect they must be fair coins) and demonic choice (because we cannot expect to find physical coins that are exactly fair), neither the technique of Sec. 5 (for hidden demonic choice) nor of Sec. 6 (for hidden probability) is sufficient on its own. In Sec. 7 just above we investigated therefore some of the issues to be faced when joining hidden probability and hidden nondeterminism together. In this section, we speculate on what we might find once we have done so.

²⁰ We write \oplus for exclusive-or.

```

hid  $y_A, y_B, y_C$ ;
reveal  $y_A \oplus y_B \oplus y_C$ 

```

(a) Specification

```

hid  $y_A, y_B, y_C$ ;
[[ hid  $c_A, c_B, c_C$ : Bool;
    $c_A$ : $\in$  Bool;
    $c_B$ : $\in$  Bool;
    $c_C$ : $\in$  Bool;
   reveal  $y_A \oplus c_B \oplus y_C$ ;
   reveal  $y_B \oplus c_C \oplus y_A$ ;
   reveal  $y_C \oplus c_A \oplus y_B$ ;
  ]]

```

(b) Implementation

The three cryptographers' choices are $y_{\{A,B,C\}}$; the three coins are $c_{\{A,B,C\}}$ with c_A being opposite y_A etc. and so hidden from her. We assume the cryptographer's choices are made before the protocol begins; the coins, which are local variables, are flipped during the protocol.

The statement **reveal** E publishes to everyone the value of E but does not expose explicitly any of E 's constituent sub-expressions [43, 49, 37].

Thus the specification reveals (to everyone) the exclusive-or of the cryptographers' actions. The implementation reveals three separate Booleans whose exclusive-or equals the value revealed by the specification. The key question is whether the implementation reveals *more* than that.

Fig. 3. The Dining Cryptographers: Specification and Implementation

8.1 The Shadow suffices for one-off lunches

We must remember right at the start that the Dining Cryptographers Protocol is not itself under suspicion. Rather we are using it—in various formulations—to test our approaches to capturing the essential features of protocols of the kind it exemplifies. We have already showed elsewhere [54, 49] that the Shadow Model can prove the refinement (a) \sqsubseteq (b) in Fig. 3.

Yet we argued in Sec. 2 that Fig. 3(b) is *not* a correct implementation in the case that the cyptographers are observed over many trials. What exactly then did our purported proofs show? The answer depends crucially on the notion of what tests we are allowed to perform on the implementation.²¹

8.2 Testing standard classical sequential programs

By *standard* programs we mean “non-probabilistic,” and by *classical* we mean “without hiding.”

²¹ This is true when assessing the suitability of any notion of computer-system correctness: one must know exactly what tests it is expected to pass.

A test for a standard classical program Q is a context \mathbb{C} into which the program Q can be put. The combination $\mathbb{C}(Q)$ is then run *a single time*, and it's observed whether $\mathbb{C}(Q)$ diverges or not. We have $P \sqsubseteq Q$ *for testing* just when, for all \mathbb{C} , if $\mathbb{C}(Q)$ fails the test by diverging, on any single run, then $\mathbb{C}(P)$ can fail also.²²

How does this fit in with *wp*-style refinement? There we have the definition $P \sqsubseteq_{wp} Q$ just when for all postconditions ψ the implication $wp.P.\psi \Rightarrow wp.Q.\psi$ holds [3, 61, 50, 51, 5], where a postcondition is a subset of the state space (equivalently a predicate over the program variables).

Now if $P \sqsubseteq_{wp} Q$ then by monotonicity of programs and the permissible program operators we have that $\mathbb{C}(P) \sqsubseteq_{wp} \mathbb{C}(Q)$, and so we have as a consequence the implication $wp.\mathbb{C}(P).\text{true} \Rightarrow wp.\mathbb{C}(Q).\text{true}$ — that is, if $\mathbb{C}(Q)$ can diverge ($\neg wp.\mathbb{C}(Q).\text{true}$) then so can $\mathbb{C}(P)$ diverge ($\neg wp.\mathbb{C}(P).\text{true}$). That establishes that *wp*-refinement implies testing refinement as we defined it above.

For the opposite direction, we suppose that $P \not\sqsubseteq_{wp} Q$ so that for some state s_0 and postcondition ψ we have $wp.P.\psi.s_0 \wedge \neg wp.Q.\psi.s_0$. Choose context \mathbb{C} so that $\mathbb{C}(X)$ is $s := s_0; X; \{\psi\}$, where the assumption $\{\psi\}$ acts as **skip** in states satisfying (belonging to) ψ and diverges otherwise [51, Sec. 1.8]. Then $\mathbb{C}(P)$ does not diverge but $\mathbb{C}(Q)$ can diverge, so that we have $P \not\sqsubseteq Q$ for testing also, and have established that in fact \sqsubseteq_{wp} and single-time-testing refinement are exactly the same.

8.3 Testing standard programs with hiding

To capture hiding, our tests are extended: they now include *as well as the above* (Sec. 8.2) an attacker's *guess* of the value of h within its type \mathcal{H} : it's expressed as a subset H of \mathcal{H} of the value the hidden variable h might have. A program-in-context $\mathbb{C}(Q)$ is deemed to fail the test H if either it diverges (as before) or it converges but in that latter case the attacker can prove conclusively on the basis of his observations that $h \in H$.

With these richer tests we can see more structure in their outcomes. Whereas in Sec. 8.2 there were just two outcomes *diverge* and *converge*, now there are many: we have *diverge* as before, but now also *converge with h provably in H* for any $H \subseteq \mathcal{H}$. The two-outcome case Sec. 8.2 is just the special case in which we fix $H := \emptyset$. Clearly there is a refinement order on these outcomes, too: divergence is at bottom, and larger guesses are above smaller ones — we could call this order \sqsubseteq_{obs} .

With the differing detail but unifying theme of this section and the previous, the use of the context, and the general idea behind it, should now be clear: given two programs P, Q whose refinement status might be complex, we define $P \sqsubseteq Q$ just when for all contexts \mathbb{C} we have $\mathbb{C}(P) \sqsubseteq_{obs} \mathbb{C}(Q)$. The point of that is the

²² Here we borrow notions best known from process algebra [10]; those ideas featured prominently in the football matches [2].

simplicity of \sqsubseteq_{obs} — it is necessarily a subjective definition, and so must be one that the community generally can accept as self-evidently reasonable.²³

8.4 The “single-test” convention applies to The Shadow

In both Sec. 8.2 and Sec. 8.3 above we stipulated “single tests,” and this is surprisingly important. Its consequence is that an implementation cannot be invalidated on the basis of considering two or more testing outcomes at the same time: each test must be considered on its own. This is not a theorem, or even a necessity: rather it is an explicit choice we make about how we define “is implemented by.”

It’s this single-time definition of test that prevents our telling the difference between “cold” nondeterminism that is unknown but fixed (like having either a heads-only or tails-only coin, but not being able to predict which it will be), and “hot” nondeterminism that can vary (like an ordinary coin that can reveal a different face on each flip, even though we know it’s either heads or tails). We do not judge this distinction either as worthwhile, or worthless: it depends on the circumstances. Nevertheless if one adopts the single-time testing regime, then there is no point in having a semantics that distinguishes hot- from cold nondeterminism. Given the style of computing that was dominant at the time of flow-charts [18], and the introduction of Hoare logic [28] and weakest preconditions [14], it’s no wonder that single-time testing was assumed: most programs were sequential, and ran on mainframes. That assumption is one of the reasons that the relational calculus —so simple— is sufficient for that kind of sequential program: it need not (cannot) distinguish hot and cold.

The Shadow too, in that tradition, is based on single tests — as are many other formal approaches to non-interference [20, 9, 36, 63]: and that is why it does not *necessarily* apply to the Café in which, effectively, many tests are made.

It is also the reason that The Shadow does not stipulate whether later hidden nondeterministic choices can be influenced by earlier hidden nondeterministic outcomes. (Recall Sec. 7.3.) It seems a reasonable question: given that the hidden choice $h_0:\in\{\text{T}, \text{F}\}$ cannot be seen by the attacker, can it be seen by a subsequent hidden choice $h_1:\in\{\text{T}, \text{F}\}$ to a different variable? The surprising answer, as we saw, is that it doesn’t matter, since in single-time testing you cannot determine whether the h_1 -choice was influenced by the h_0 -choice or not.

8.5 Proving (in-)correctness in the Café

Having addressed the question posed at the end of Sec. 8.1, the next step is to ask how the purported refinement of Fig. 3 would fare in a probabilistic/demonic-

²³ A particularly good example of this is found in probabilistic/demonic process algebras [1], where the notion of refinement is very complex and not generally agreed. Research there concentrates on finding a testing regime that uses contexts to reduce refinement to something as simple as decreasing the probability of divergence, or increasing the probability of performing some single distinguished “success action” [10].

with-hiding semantics if we constructed one along the lines suggested in Sec. 7. The answer is that *we expect it to fail*. In this section we sketch speculatively just how the as-yet-to-be-constructed model would signal that failure.

Rather than address the whole protocol, we concentrate on the *Encryption Lemma*, a key part of its qualitative proof [54, 43, 49]. That lemma states that in the context of a global hidden Boolean h the following refinement holds:

$$\mathbf{skip} \sqsubseteq \llbracket \mathbf{hid} \ h'; \ h' : \in \mathbf{Bool}; \mathbf{reveal} \ h \oplus h' \rrbracket . \quad ^{24} \quad (11)$$

This via refinement states that the right-hide side reveals nothing about h , since clearly the left-hand side does not.

Now suppose that the global h has been set previously via an assignment $h := \mathbf{true}_p \oplus \mathbf{false}$. (In so doing, we leave the world of single-time testing, since probabilities cannot be expressed or detected there.) Does (11) still hold?

In our semantics we might expect that the denotation of the right-hand side of (11) is a set of behaviours that may be observed by an onlooker, in which each of those observable behaviours is associated with a set of “hidden” probabilistic states that could actually have produced it. In this program an onlooker can see the outcome of the reveal statement, and so an observable behaviour is the distribution of true and false values shown by the reveal. Given that the demonic choice $h' : \in \mathbf{Bool}$ is interpreted as “choose h' with some unknown probability q ” [41, 59], where $0 \leq q \leq 1$, for each observable behaviour of the reveal statement we can calculate that what h' must have been from the fact that h was set deterministically using probability p . This means that there can only be one “hidden” probabilistic state associated with every visible outcome. For all visible outcomes other than that created by choosing h' to be 1/2, there will be a correlation between the outcome of the reveal statement and the hidden distribution of h -values: that means that our knowledge of h 's value will not have been left unchanged by the operation. And this is not something that our mathematical definition of refinement would accept as a refinement of **skip** since, that would –of course– have left h 's distribution unchanged.

8.6 Testing probabilistic programs with hiding

Given the conceptual programme established by our discussion of testing, the guesswork of Sec. 8.5 must be accompanied by a discussion of the testing regime we have in mind. As remarked there, it cannot any longer be single-time.

Here we expect to be using *single-tabulation* testing: the program Q under test is run repeatedly in some context \mathbb{C} , and we tabulate the outcomes and their frequencies. What we are not allowed to do is to run the program repeatedly on two separate occasions and then to draw conclusions from the two separate tabulations that result.

Such a regime is implicit in our earlier probabilistic work where however we do not model hiding [41, 59]. We would therefore –without hiding– be dealing

²⁴ We use brackets $\llbracket \cdot \rrbracket$ to delimit the scope of local declarations.

with a quantitative version of the tests of Sec. 8.2 where the refinement criterion for $P \sqsubseteq Q$ would be “if program-in-context $\mathbb{C}(Q)$ can diverge with a probability at least p , then so can $\mathbb{C}(P)$.” Note that this single-tabulation restriction again makes the issue of hot- vs. cold nondeterminism irrelevant: if you are restricted to a single tabulation of outcomes, there is no way in which you can tell the difference with respect to a demonic coin-choice $h: \in \text{Bool}$ whether this is implemented via a number of different coins with (therefore) varying bias, or whether it is a single one whose bias is unknown.²⁵

In our speculative semantics –with hiding– our single-tabulation testing could be a simple probabilistic powerdomain [32, 31] built over the underlying order on H -guesses sketched in Sec. 8.3.

8.7 The importance of fair coins

We saw in Sec. 2.1 that the Dining Cryptographers’ Protocol, and others like it, is described in terms of fair coins (1) even though the fairness was not explicitly used in our correctness argument (2). One reason for this (we think) is that the correctness relies crucially on the independence of the coins from other phenomena (such as the cryptographers’ already-made decisions) and indeed from each other. That independence in its abstract *demonic* form is very difficult to formalise, and so an easy route to ensuring it is realised is to make the choices with a fair-coin -flip since –barring sophisticated nanotechnology within the coin itself– it is going to implement precisely the kind of independence we mean but which we cannot pin-down with a precise mathematical description.²⁶

In spite of that, we saw in Sec. 8.5 that the fairness really does matter; and in Chaum’s original presentation it mattered too. That is, in our speculative semantics it matters because we would expect the refinement

$$\text{skip} \sqsubseteq \llbracket \text{hid } h'; h' := \text{true}_{1/2} \oplus \text{false}; \text{reveal } h \oplus h' \rrbracket \quad (12)$$

²⁵ Imagine watching a single referee throughout a football season. Could you tell with a single tabulation for the whole season whether he decides the kickoff using one coin or several? A multiple-tabulation testing regime would allow you to compare one season with another, and then (only then) you would be able to distinguish the two cases.

²⁶ There are many examples of this “abstraction aversion” in everyday life. For example, in a two-battery torch (flashlight) the batteries must be inserted in the same direction; but either way will do. (We ignore issues of the shape of the contacts, and concentrate on electricity.) A picture describing this abstraction is hard to imagine, however: if both alternatives were shown, would people think they needed four batteries? The usual solution is to replace the specification (either way, but consistent) with an implementation (both facing inwards) and then cynically to present that as the specification. Most people won’t even notice that the abstraction has been stolen.

Similarly, what ideally would be an abstract description (“independent mutually ignorant demonic choices”) is replaced pragmatically by a concrete one (“separate fair coins”) — that’s the best we can do with our current tools.

to go through even though (11) does not. Firstly, the denotation of the right-hand side would no longer be a demonic choice, since the choice of h' is deterministic (i.e. no longer demonic, although still probabilistic). Secondly, if one asks the Bayesian question “What is the *a posteriori* distribution of h after the right-hand side of (12)?” the answer will be that it is the same as the *a priori*, because the distribution of h' is fair. (That is exactly the key point of Chaum’s proof.) With any other distribution of h' , however, the *a priori* and *a posteriori* distributions of h would differ. Thus in this case only –an exactly fair coin– the refinement is accepted.²⁷

8.8 The non-existence of fair coins

Our conclusion –generalised from the discussion of Sec. 8.7– seems to be that the Dining Cryptographers’ Protocol suffices for use in the Café just when the coins are exactly fair, and not otherwise. But they never are exactly fair, and so with a definition of refinement as “yes or no,” we seems to be stranded: our refinement proofs apply only to *ideal* situations, ones that never occur in reality. That is, the semantics we are hoping to develop (Sec. 7) seems to be running the risk of becoming “ideal” in a similar sense: pejorative.

There are two complementary steps possible to avoid, at this point, being relegated to the Platonic School of Formal Methods. One is to introduce *approximate refinement*, where we would say (intuitively) that $P \sqsubseteq_p Q$ means that P is refined by Q with probability p , with a special case being probabilistic equivalence [66, 67, 30, 13, 11, 12]. The “with probability p ” applied to refinement can be made precise in various ways, and the first thing one would prove is that \sqsubseteq_1 coincides with ordinary, certain refinement; similarly \sqsubseteq_0 would be the universal relation.

The reason we don’t take this approach is that we cannot see how to couple it with a notion of testing in the case that the probability p of refinement is intermediate, not equal to zero or one. And we think the testing connection

²⁷ The usual example of this is a nearly reliable test for a disease: say with 99% reliability if the test detects the disease, the patient really is diseased; the remaining 1% is called a “false positive.” We assume 1% for false negatives also. Now we suppose that in the population just 1% of the people have the disease, so that if a person is selected at random, then this is the *a priori* probability that she has it. If she tests positive, what now is the probability that she really does have the disease?

The probability of her testing positive is $0.99 \times 0.01 + 0.01 \times 0.99 = 0.0198$, and the (conditional) probability she has the disease, i.e. *given* that she tested positive, is then $0.01 \times 0.99 / 0.0198 = 50\%$. This is the *a posteriori* probability, and it has changed from the *a priori* probability (which was 1%): the test is not equivalent to **skip**.

Now finally suppose that the test is only 50% reliable, either way. The calculation becomes $0.99 \times 0.5 + 0.5 \times 0.99 = 0.5$, i.e. the probability of testing positive is 0.5 (and in fact would be that no matter what the distribution of the disease in the population); and the *a posteriori* probability is $0.99 \times 0.5 / 0.5 = 99\%$, just the same as the *a priori*. This 50%-reliable test *is* equivalent to **skip**.

is essential. There are other more technical questions that loom as well: for example, if we have $P \sqsubseteq_p Q \sqsubseteq_q R$, what then is the refinement relation between P and R ? Is it \sqsubseteq_{pq} perhaps? But what if the fact that p and q are not both one is due to the same cause, that they are “common-mode” failures? Then we would expect a composite probability higher than pq — but how mathematically and quantitatively do we capture the common-mode dependence?

Thus we take a different approach: we use our ability to combine nondeterminism and probability to reflect the implementation’s inadequacy back into a corresponding inadequacy, a weakening of the specification. An example of this technique is the probabilistic steam boiler [48, 41] and the tank monitor [64] in which probabilistically unreliable sensors in the implementation force, via a yes-or-no refinement relation, a probabilistically unreliable *specification*. The refinement relation remains definite.

In applying that idea to our current examples (11 & 12) we would postulate a refinement

$$\text{“not-quite-skip”} \sqsubseteq \llbracket \mathbf{hid} \ h'; \ h' := \text{true}_{[\varepsilon]} \oplus \text{false}; \mathbf{reveal} \ h \oplus h' \rrbracket \quad (13)$$

in which we have to figure out what “not-quite-skip” is. (Recall Secs. 2.3 & 4: the subscript $[\varepsilon]$ abbreviates $[(1-\varepsilon)/2, (1+\varepsilon)/2]$.)

In fact a good candidate for the definition of “not-quite-skip” is provided by (13) itself. The statement $\mathbf{reveal} \ E$, where E is an expression possibly containing hidden variables, publishes E to the world at large — but any calculations done to determine E are *not* published [43, 49]. Thus for example $\mathbf{reveal} \ h \oplus h'$ publishes whether h and h' are equal but, beyond that, says nothing about either of them.

The Shadow definition of $\mathbf{reveal} \ E$ was chosen so that it was equal to the program fragment

$$\llbracket \mathbf{vis} \ v; \ v := E \rrbracket, \quad (14)$$

where a local visible variable v is introduced temporarily to receive E ’s value. Because v is visible, everyone can see its final value (thus publishing E); but because it is local, it does not affect the rest of the program in any other way. We extend the revelation command so that the expression can be probabilistic and even nondeterministic, postulating that its definition will still respect (14). Thus we would have

$$\mathbf{reveal} \ h_p \oplus \neg h \quad = \quad \llbracket \mathbf{vis} \ v; \ v := (h_p \oplus \neg h) \rrbracket,$$

in which the assignment to v is atomic.²⁸ That suggests our weakened specification, and respects our (still) definite refinement, in fact equality:

$$\mathbf{reveal} \ (h_{[\varepsilon]} \oplus \neg h) \quad = \quad \llbracket \mathbf{hid} \ h'; \ h' := \text{true}_{[\varepsilon]} \oplus \text{false}; \mathbf{reveal} \ h \oplus h' \rrbracket. \quad (15)$$

²⁸ For those with some familiarity with The Shadow: the atomicity requirement distinguishes the command from the composite (non-atomic) fragment $v := h_p \oplus v := \neg h$ in which the branch taken –left, or right– would be visible to the attacker. The composite fragment is in fact for any p equal to the simpler $\mathbf{reveal} \ h$ since, after its execution and knowing which assignment was executed, an attacker would also know whether to “un-negate” the revelation or not in order to deduce h ’s value.

<pre> hid y_A, y_B, y_C; reveal y_A $[\delta] \oplus \neg y_A$; reveal y_B $[\delta] \oplus \neg y_B$; reveal y_C $[\delta] \oplus \neg y_C$; reveal $y_A \oplus y_B \oplus y_C$ </pre>	<pre> hid y_A, y_B, y_C; [[hid c_A, c_B, c_C: Bool; $c_A := (\mathbf{true} [\varepsilon] \oplus \mathbf{false})$; $c_B := (\mathbf{true} [\varepsilon] \oplus \mathbf{false})$; $c_C := (\mathbf{true} [\varepsilon] \oplus \mathbf{false})$; reveal $y_A \oplus c_B \oplus y_C$; reveal $y_B \oplus c_C \oplus y_A$; reveal $y_C \oplus c_A \oplus y_B$;]] </pre>
(a) Specification	(b) Implementation

Here the coins, which are local variables, are flipped during the protocol and are within ε of being fair.

The specification reveals to everyone the exclusive-or of the cryptographers' actions collectively, and a little bit about the cryptographers' actions individually. The δ depends on ε : it should tend to zero as ε tends to zero and to 1 as ε tends to 1.

Below we argue informally that $\delta := \varepsilon^2$ is correct, though perhaps not the best we can do.

Fig. 4. The Dining Cryptographers: Specification and Implementation revisited

8.9 Weakening the Café's specification

In Fig. 4(a) we have modified the Café's specification along the lines suggested in Sec. 8.8. Our informal justification for the suggested choice $\delta := \varepsilon^2$ of specification tolerance is as follows.

Each cryptographer's act –say y_A for example– is exclusive-or'd with a composite Boolean: it is $c_B \oplus c_C$ in the case of y_A . From our earlier calculation (4) we know that if $c_{\{B,C\}}$ are at most ε -biased then the composite $c_B \oplus c_C$ is at most $\varepsilon\varepsilon$ -biased. The postulated reveal statements are then by definition (of the extended **reveal**) biased by that value, that is by ε^2 .

The reason we are not sure whether this is as good as it could be is that the three composite values (the other two being $c_C \oplus c_A$ and $c_A \oplus c_B$) are not independent: it might be that revealing a lot about one cryptographer has a limiting effect on what can be revealed about another.²⁹

We now discuss the importance of the δ parameter from the specifier's, that is the customer's point of view: we have gone to a lot of trouble to formulate it; we should now explore how to use it. We'll look at three cases.

Exactly fair coins In this case $\varepsilon=0$ and so $\delta=0$ also. Thus in the specification we find **reveal** $y_{A[0]} \oplus \neg y_A$ etc., that is **reveal** $y_{A1/2} \oplus \neg y_A$ which is equivalent to **skip**. Thus all three revelations of $y_{\{A,B,C\}}$ can be removed, and we are

²⁹ In fact we should recall that this whole construction is speculative, since we do not yet have a finalised semantics in which to prove it.

left with the original specification of Fig. 3. The refinement then establishes that using fair coins results in no leaks at all.

Coins of completely unknown bias In this case $\varepsilon=1$ and so $\delta=1$ also. Thus in the specification we find **reveal** $y_A \sqcap \neg y_A$, that is by definition the command **reveal** $y_A \sqcap \neg y_A$. This, by definition again (14), is equal to

$$\llbracket \mathbf{vis} \ v; \ v := (y_A \sqcap \neg y_A) \rrbracket .^{30} \quad (16)$$

Interpreting Program (16) throws into relief one of the major issues in the construction of our sophisticated model: what can the nondeterminism “see”? We are assuming –speculatively– that it can see nothing, and so (16) is describing an experiment where a cryptographer chooses an act (y_A), and it is decided *independently of that choice* whether to reveal the act or its complement. Once that decision is made, the value (or its complement) is revealed *but we still do not know whether the complement was taken*. What can we deduce about the cryptographer’s bias in her actions? We reason as follows.

Suppose the revealed value has distribution $\mathbf{true} _{1/4} \oplus \mathbf{false}$. If the cryptographer is sufficiently biased, it is possible since the nondeterminism in (16) could have been resolved e.g. to just **reveal** y_A . Then, from our earlier arithmetic, we would know that in fact the cryptographer has bias at least $1/2$, and we know that *without* being told anything about how (16) was resolved. Thus we *can* deduce something in this case: the *specification* **reveal** $y_A \sqcap \neg y_A$ expresses an insecurity, as it should — for the implementation is insecure also.

It’s instructive also to look at the situation from the cryptographer’s point of view: suppose she is indeed $1/2$ -biased but, embarrassed about that, she wants to keep her bias secret. Should she risk dining with the others, according to the implementation given in Fig. 4(b)? Rather than study the implementation, she studies the specification (having been well schooled in Formal Methods). She sees that from the specification an observer could learn that her bias is *at least* $1/2$, since that is what would be deduced if the \sqcap were always taken the same way, as we saw above. (The reason for the “at least” is that a $\mathbf{true} _{1/4} \oplus \mathbf{false}$ outcome in the reported result could also result from a greater bias in the cryptographer which is masked by some lesser bias in the \sqcap .)

Coins that are nearly fair Finally, in this last case, we examine whether being able to limit the bias really does give us access to a continuum of insecurity: so far, we have seen either “all” or “nothing.” We continue with the

³⁰ In the standard Shadow (non-probabilistic) this would reveal nothing, since we could reason

$$v := (y_A \sqcap \neg y_A) \quad = \quad v \in \{y_A, \neg y_A\} \quad = \quad v \in \{\mathbf{true}, \mathbf{false}\} .$$

And indeed in a single-run testing scenario it does not reveal anything. It’s important therefore that we remember here that our scenario is more general.

embarrassed cryptographer of the previous case: can she deduce that in the current case her potential for embarrassment is smaller than before?

Yes she can; and to explain that we return one final time, in the next section, to the issue of testing.

8.10 Hypothesis testing

If a cryptographer is exactly fair, then also her utterances will be exactly fair no matter what the bias of the coins. Given that, we can reason that over 100 lunches, say, with probability 95% she will say she paid for between 40 and 60 of them.³¹

However, just as there are no exactly fair coins, neither are there going to be exactly fair cryptographers: let's say that a cryptographer is *reasonable* if her bias is no more than 10%. In this case it does make a difference to her utterances whether the coins she faces are biased, and indeed her most biased utterances will result from the most biased coins. What range of utterances should we expect?

If the cryptographer is only just reasonable, and *wlog* is on the miserly side, then she is described by $y := \mathbb{T}_{0.45} \oplus \mathbb{F}$. With completely biased coins –the worst case– then *wlog* this will be the description of her utterances too.³² Given that, we can reason that over 100 lunches with probability 95% she will say “I paid” at least 36 times.³³ Thus –undoing our *wlog*'s– a reasonable cryptographer will in 100 trials with 95% probability say “I paid” between 36 and 64 times.

Now let's look at the case of the unreasonable, indeed miserly 50%-biased cryptographer $y := \mathbb{T}_{0.25} \oplus \mathbb{F}$. Using fully biased coins, with what probability will

³¹ With -1 assigned to \mathbb{F} and 1 to \mathbb{T} , the mean of her outcomes y is 0 and the variance is 1 ; over 100 independent trials the distribution of the sum y_{100} will therefore have mean 0 and variance 100 . From the Central Limit Theorem [22] the derived variable $\hat{y} := y_{100}/\sqrt{100}$ will be approximately normally distributed, and so we have $-1.96 \leq \hat{y} \leq 1.96$ with probability 95% (from tables of the Normal Distribution). Thus $-19.6 \leq y_{100} \leq 19.6$, so that out of 100 lunches she is slightly more than 95% certain to say “I paid” for between 40 and 60 of them.

Note this is *not* the same as saying “If the frequency of I-paid utterances is between 40 and 60 out of 100, then with 95% probability the cryptographer is fair.” That kind of statement requires a prior distribution on the cryptographers, which we are not assuming.

³² The “without loss of generality”'s mean that we could have taken the complementary $y := \mathbb{T}_{0.55} \oplus \mathbb{F}$ for the cryptographer, and that we could have allowed the biased coins always to invert her outcome. But none of that makes any difference to the analysis to come.

³³ The mean of her utterances y is -0.1 and the variance is 0.99 ; the distribution of the sum y_{100} has mean -10 and variance 99 . The derived variable is $\hat{y} := (y_{100} + 10)/\sqrt{99}$ and will satisfy $-1.65 \leq \hat{y}$ with probability 95%. Thus $-1.65 \times \sqrt{99} - 10 = -26.4 \leq y_{100}$, so that she is slightly more than 95% certain to say “I paid” for at least $(100 - 26.4)/2 \approx 36$ lunches.

her utterances fall *outside* of the range 36–64 that we with confidence 95% would expect of a reasonable cryptographer? That probability turns out to be 99.7%.³⁴

Thus if we set 36–64 “I paid”’s in 100 trials as our “reasonableness” criterion in the case of potentially fully biased coins, a reasonable cryptographer will be unfairly judged as biased only 5% of the time while a 50%-biased cryptographer will escape detection only 0.3% of the time.

All the above was prelude to our discussion of *partially* biased coins: we now assume for argument’s sake that the coins’ (composite) bias is no more than 50%, and we re-do the above calculations to see how it affects the miserly cryptographer’s chance of escaping detection.

As before, we determine our criteria by considering how a reasonable cryptographer would be likely to behave: facing a 50%-biased coin, she will *wlog* have utterances with distribution at worst $y := T_{0.475} \oplus F$. Over 100 lunches with probability 95% she will say “I paid” between 39 and 61 times.³⁵

The miserly cryptographer will *wlog* have utterances at worst $y := T_{0.375} \oplus F$ — in both cases, the coins’ bias mitigates the cryptographer’s intrinsic bias to give a smaller bias in her utterances. With what probability will the miser’s utterances fall *outside* of the range 39–61 that we might expect of a reasonable cryptographer facing a 50%-biased coin? It turns out to be 64.8%, which though still high is a much less rigorous test than the 99.7% she faced before.³⁶

Thus if we set 39–61 “I paid”’s in 100 trials as our “reasonableness” criterion in the case of 50%-biased coins, so that as before a reasonable cryptographer will be unfairly judged as biased only 5% of the time, a 50%-biased cryptographer will now escape judgement 35.6% of the time (in fact roughly 100 times as often as she did in the fully biased case). The miser is being (partially) protected by the (partial) secrecy implemented by the (partially) secure protocol.

It is interesting to speculate whether our model will have

$$\text{reveal } h_{[\varepsilon_1]} \oplus \neg h \quad \sqsubseteq \quad \text{reveal } h_{[\varepsilon_0]} \oplus \neg h$$

whenever $\varepsilon_1 \geq \varepsilon_0$.

³⁴ The mean of her utterances y is -0.5 and the variance is 0.75 ; the distribution of the sum y_{100} has mean -50 and variance 75 . The derived variable is $\hat{y} := (y_{100} + 50)/\sqrt{75}$ which, to appear reasonable, from Footnote 33 must satisfy $(-26.4 + 50)/\sqrt{75} = 2.72 \leq \hat{y}$. It does that with probability 0.3% .

³⁵ The mean of her utterances y is -0.05 and the variance is 0.9975 ; the distribution of the sum y_{100} has mean -5 and variance 99.75 . The derived variable is $\hat{y} := (y_{100} + 5)/\sqrt{99.75}$ and will (as before) satisfy $-1.65 \leq \hat{y}$ with probability 95% . Thus $-1.65 \times \sqrt{99.75} - 5 = -21.5 \leq y_{100}$, so that she is slightly more than 95% certain to say “I paid” for at least 39 lunches.

³⁶ The mean of her outcomes y is -0.25 and the variance is 0.875 ; the distribution of the sum y_{100} has mean -25 and variance 87.5 . The derived variable is $\hat{y} := (y_{100} + 25)/\sqrt{87.5}$ which to appear reasonable must satisfy $(-21.5 + 25)/\sqrt{87.5} = 0.37 \leq \hat{y}$. It does that with probability 35.6% .

9 Conclusions

In this paper we have explored the abstract phenomena underpinning secure software systems which operate in partially predictable environments; we have suggested a number of models and investigated how they measure up to the problems software designers face. Yet however successful a mathematical model is at capturing the underlying phenomena, we understand that its impact may be limited and its results marginalised if the formal techniques it can support cannot be made accessible to actual program developers.

We say that a formal technique is *usable* only if it can be practically applied, and *relevant* only if it delivers accurate results. Unfortunately usability and relevance can normally coexist only if the theoretical basis for the technique is both simple enough to be supported by automated tools³⁷ and yet complicated enough to yield accurate results. As the examples of this paper show it is unlikely that there is such a *one model to fit all* for any problem domain which includes all the features of hiding, probability and multiple agents. That’s not to say that there is no solution at all — but as formal methods researchers the challenge is to find accurate models with usable abstractions, models whose relationships to each other are well enough understood to support tool-based analyses and to apply after all to generic and relevant scenarios.

10 Epilogue: A café-strength proof of the Cryptographers’ Protocol

The Shadow Model captures the qualitative correlation between hidden and visible state. It is simple enough to support an algebra which is amenable to routine calculation: we have twice earlier published refinement-style proofs of the Cryptographers’ Protocol [54, 49], the novelty being of course not in the protocol (published long ago) nor in its correctness (established by its inventor) but in the style of proof (program algebra and refinement). In doing so we hoped to encourage Formal Methods to expand further into these application areas.

Our earlier proofs — as for the other approaches based on qualitative noninterference [20, 9, 36, 63] — abstracted from probability, from the fairness of the coins, and as we explained at length above that left open the question of their validity when repeated trials of the protocol are carried out — which is its usual environment, after all. And we proposed several methods *in spe* that might legitimise a fully quantitative proof. What would that proof look like?

There is a good chance that the café-strength proof of the Cryptographers’ Protocol, thus valid for repeated trials, will have exactly the same structure as our original qualitative proof.

That is because the original proof’s use of the “qualitative” Encryption Lemma (11) seems to be its only point of vulnerability in the more general

³⁷ This is rather a stark definition, as it ignores the huge benefits to be gained in a formal methods education, which might not be based on tool support.

context; and the less-abstract “uniform” Encryption Lemma (12) is *not* similarly vulnerable. All the other algebraic properties appear to carry through. If we are right, then there will be a large class of security protocols which, if proven correct with a carefully chosen *qualitative* repertoire of algebraic laws, will retain their correctness in the more general context of repeated runs — provided certain crucial demonic choices (the ones that play a part in uses of the Encryption Lemma) are replaced by uniform probabilistic choices. No other changes should be required.

As we have seen elsewhere, with purely qualitative treatments of probabilistic fairness [46, 23], that would have a significant impact on the applicability of tools, by allowing them to remain Boolean rather than having to deal with proper probabilities.

Watch this space...

References

1. A large literature on probabilistic process algebras from 1990 or before.
2. A series of meetings between Oxford and Manchester over the general principles of data refinement (reification) and its completeness, 1986. Participants included Jifeng He, Tony Hoare, Cliff Jones, Peter Lupton, Carroll Morgan, Tobias Nipkow, Ken Robinson, Bill Roscoe, Jeff Sanders, Ib Sørensen and Mike Spivey.
3. R.-J.R. Back. On the correctness of refinement steps in program development. Report A-1978-4, Dept Comp Sci, Univ Helsinki, 1978.
4. R.-J.R. Back. Data refinement in the refinement calculus. In *Proceedings 22nd Hawaii International Conference of System Sciences*, Kailua-Kona, January 1989.
5. R.-J.R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.
6. M.J. Butler and P.H. Hartel. Reasoning about Grover’s quantum search algorithm using probabilistic *wp*. *ACM Trans Prog Lang Sys*, 21(3):417–30, 1999.
7. O. Celiku and A. McIver. Cost-based analysis of probabilistic programs mechanised in HOL. *Nordic Jnl Comp*, 11(2):102–128, 2004.
8. D. Chaum. The Dining Cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.
9. E.S. Cohen. Information transmission in sequential programs. *ACM SIGOPS Operatings Systems Review*, 11(5):133–9, 1977.
10. M. de Nicola and M. Hennessy. Testing equivalence for processes. *Theo Comp Sci*, 34, 1984.
11. Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. *Electronic Notes in Theoretical Computer Science*, 153(2):79–96, 2006.
12. Yuxin Deng and Wenjie Du. Kantorovich metric in computer science: A brief survey. 2009. To appear in *Proceedings of the 7th Workshop on Quantitative Aspects of Programming Languages*.
13. J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc of the 17th Annual IEEE Symp Logic in Computer Science*, pages 413–422. IEEE, 2002.
14. E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

15. K. Engelhardt, Y. Moses, and R. van der Meyden. Unpublished report, Univ NSW, 2005.
16. K. Engelhardt, R. van der Meyden, and Y. Moses. A refinement theory that supports reasoning about knowledge and time. In Robert Nieuwenhuis and Andrei Voronkov, editors, *LPAR*, volume 2250 of *Lecture Notes in Computer Science*, pages 125–41. Springer, 2001.
17. C Fidge and C Shankland. But what if I don't want to wait forever? *Formal Aspects of Computing*, 14(3):281–94, 2003.
18. R.W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Mathematical Aspects of Computer Science*, number 19 in Proc Symp Appl Math., pages 19–32. American Mathematical Society, 1967.
19. P.H.B. Gardiner and C.C. Morgan. Data refinement of predicate transformers. *Theo Comp Sci*, 87:143–62, 1991. Reprinted in [60].
20. J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc IEEE Symp on Security and Privacy*, pages 75–86, 1984.
21. C. Gonzalia and A.K. McIver. Automating refinement checking in probabilistic system design. In *Proc ICFEM '07*, volume 4789 of *LNCS*. Springer, 2007.
22. G.R. Grimmett and D. Welsh. *Probability: an Introduction*. Oxford Science Publications, 1986.
23. S. Hallerstede and T.S. Hoang. Qualitative probabilistic modelling in Event-B. In *Integrated Formal Methods*, volume 4591 of *LNCS*. Springer, 2007.
24. Jifeng He, Karen Seidel, and AK McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:171–92, 1997.
25. Thai Son Hoang. *The Development of a Probabilistic B-Method and a Supporting Toolkit*. PhD thesis, Computer Science and Engineering, 2005.
26. T.S. Hoang, A.K. McIver, C.C. Morgan, K.A. Robinson, and Z.D. Jin. Probabilistic invariants for probabilistic machines. volume 2651 of *LNCS*. Springer, 2003.
27. T.S. Hoang, C.C. Morgan, K.A. Robinson, and Z.D. Jin. Refinement in probabilistic B: Foundation and case study. In H. Treharne and S. Schneider, editors, *Proc ZB 2005*, volume 3455 of *LNCS*. Springer, 2005.
28. C.A.R. Hoare. An axiomatic basis for computer programming. *Comm ACM*, 12(10):576–80, 583, October 1969.
29. Joe Hurd, A.K. McIver, and C.C. Morgan. Probabilistic guarded commands mechanised in HOL. *Theo Comp Sci*, 346(1):96–112, 2005.
30. E.P. de Vink, J.I. den Hartog and J.W. de Bakker. Metric semantics and full abstractness for action refinement and probabilistic choice. *Electronic Notes in Theo Comp Sci*, 40, 2001.
31. C. Jones. Probabilistic nondeterminism. Monograph ECS-LFCS-90-105, Edinburgh University, 1990. (Ph.D. Thesis).
32. 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.
33. C.B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, 1986.
34. D. Kozen. Semantics of probabilistic programs. *Jnl Comp Sys Sci*, 22:328–50, 1981.
35. D. Kozen. A probabilistic PDL. *Jnl Comp Sys Sci*, 30(2):162–78, 1985.
36. K.R.M. Leino and R. Joshi. A semantic approach to secure information flow. *Science of Computer Programming*, 37(1–3):113–38, 2000.
37. A.K. McIver. The secure art of computer programming. In *Proc. ICTAC 2009*, 2009. Invited presentation.

38. A.K. McIver and C.C. Morgan. A quantified measure of security 2: a programming logic. Available at [62, key McIVER:98A], 1998.
39. A.K. McIver and C.C. Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Inf*, 37(4/5):329–54, 2001.
40. A.K. McIver and C.C. Morgan. Abstraction and refinement of probabilistic systems. In J.-P. Katoen, editor, *ACM SIGMetrics Performance Evaluation Review*, volume 32. ACM, 2005.
41. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Tech Mono Comp Sci. Springer, New York, 2005.
42. A.K. McIver and C.C. Morgan. Developing and reasoning about probabilistic programs in *pGCL*. In Ana Cavalcanti, J.C.P. Woodcock, and A. Sampaio, editors, *The Pernambuco Formal-Methods Summer School*. Springer, 2006.
43. A.K. McIver and C.C. Morgan. A calculus of revelations, October 2008. Presented at VSTTE Theories Workshop www.cs.york.ac.uk/vstte08/.
44. A.K. McIver and C.C. Morgan. Sums and lovers: Case studies in security, compositionality and refinement. In A. Cavalcanti and D. Dams, editors, *Proc FM '09*, LNCS. Springer, 2009.
45. A.K. McIver, C.C. Morgan, and C. Gonzalia. Proofs and refutations for probabilistic systems. In T. Maibaum and J. Cuellar, editors, *Proc FM '08*, volume 5014 of LNCS. Springer, 2008.
46. A.K. McIver, C.C. Morgan, and T.S. Hoang. Probabilistic termination in *B*. In D. Bert, J.P. Bowen, S. King, and M. Walden, editors, *Proc ZB '03*, volume 2651 of LNCS, pages 2–6–239. Springer, 2003.
47. A.K. McIver, C.C. Morgan, and J.W. Sanders. Probably Hoare? Hoare probably! In J.W. Davies, A.W. Roscoe, and J.C.P. Woodcock, editors, *Millennial Perspectives in Computer Science*, Cornerstones of Computing, pages 271–82. Palgrave, 2000.
48. A.K. McIver, C.C. Morgan, and E. Troubitsyna. The probabilistic steam boiler: a case study in probabilistic data refinement. In J. Grundy, M. Schwenke, and T. Vickers, editors, *Proc International Refinement Workshop, ANU, Canberra*, Discrete Mathematics and Computer Science, pages 250–65. Springer, 1998. Also [41, Chap.4].
49. Annabelle McIver and Carroll Morgan. The thousand-and-one cryptographers. In *Festschrift in Honour of Tony Hoare*, 2009. To appear.
50. C.C. Morgan. The specification statement. *ACM Trans Prog Lang Sys*, 10(3):403–19, July 1988. Reprinted in [60].
51. C.C. Morgan. *Programming from Specifications*. Prentice-Hall, second edition, 1994. web.comlab.ox.ac.uk/oucl/publications/books/PfS/.
52. C.C. Morgan. Proof rules for probabilistic loops. In He Jifeng, John Cooke, and Peter Wallis, editors, *Proc BCS-FACS 7th Refinement Workshop*, Workshops in Computing. Springer, July 1996. ewic.bcs.org/conferences/1996/refinement/papers/paper10.htm.
53. C.C. Morgan. The generalised substitution language extended to probabilistic programs. In Didier Bert, editor, *Proc 2nd Int B Conf (B'98)*, volume 1393 of LNCS, pages 9–25. Springer, 1998.
54. 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*.
55. C.C. Morgan. How to brew-up a refinement ordering. In E. Boiten, J. Derrick, and S. Reeves, editors, *Proc 2009 Refine Workshop, Eindhoven*, 2009.
56. C.C. Morgan. *The Shadow Knows*: Refinement of ignorance in sequential programs. *Science of Computer Programming*, 74(8), 2009. Treats *Oblivious Transfer*.

57. C.C. Morgan and A.K. McIver. A quantified measure of security 1: a relational model. Available at [62, key MORGAN:98A], 1998.
58. C.C. Morgan and A.K. McIver. *pGCL*: Formal reasoning for random algorithms. *South African Comp Jnl*, 22:14–27, March 1999.
59. 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.
60. C.C. Morgan and T.N. Vickers, editors. *On the Refinement Calculus*. FACIT Series in Computer Science. Springer, Berlin, 1994.
61. J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, December 1987.
62. Probabilistic Systems Group. Publications. At www.cse.unsw.edu.au/~carrollm/probs.
63. A. Sabelfeld and D. Sands. A PER model of secure information flow. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
64. S. Schneider, T.S. Hoang, K.A. Robinson, and H. Treharne. Tank monitoring: a *pAMN* case study. *Formal Aspects of Computing*, 18(3):308–28, 2006.
65. R. Tix, K. Keimel, and G.D. Plotkin. Semantic domains for combining probability and non-determinism. *ENTCS*, 129:1–104, 2005.
66. Franck van Breugel. *Comparative Metric Semantics of Programming Languages: Nondeterminism and Recursion*. Theoretical Computer Science. Birkhäuser, 1997.
67. Mingsheng Ying and Martin Wirsing. Approximate bisimilarity. In Teodor Rus, editor, *AMAST*, volume 1816 of *Lecture Notes in Computer Science*, pages 309–22. Springer, 2000.