Compositional closure for Bayes Risk in probabilistic noninterference

Annabelle McIver[1], Larissa Meinicke[1], 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. We give a sequential model for noninterference security including probability (but not demonic choice), thus supporting reasoning about the *likelihood* that high-security values might be revealed by observations of low-security activity. Our novel methodological contribution is the definition of a *refinement* order ($\sqsubseteq$) and its use to *compare* security measures between specifications and (their supposed) implementations. This contrasts with the more common practice of evaluating the security of individual programs in isolation.

The appropriateness of our model and order is supported by our showing that ($\sqsubseteq$) is the *greatest* compositional relation –the compositional closure– with respect to our semantics and an "elementary" order based on Bayes Risk — a security measure already in widespread use. We also relate refinement to other measures such as Shannon Entropy.

By applying the approach to a non-trivial example, the anonymous-majority *Three-Judges* protocol, we demonstrate by example that correctness arguments can be simplified by the sort of layered developments –through levels of increasing detail– that are allowed and encouraged by compositional semantics.

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

Table of Contents

1 Introduction

We apply notions of testing equivalence and refinement, based on *Bayes Risk*, to the topic of *noninterference security* [10] *with probability* but without demonic choice. Previously, we have studied noninterference for demonic systems without probabilistic choice [26, 27], and we have studied probability and demonic choice without noninterfence [28, 21]. Here thus we are completing a programme of treating these features "pairwise."

Our long-term aim –as we explain in the conclusion– is to treat all three features together, based on the lessons we have learned by treating strict subsets of them. The benefit (should we succeed) would apply not only to security, but also to conventional program development where, in the presence of both probabilistic and demonic choice, the technique of data-transformation (aka. data *refinement* or data *reification*) becomes unexpectedly complex: variables inside local scopes must be treated analogously to "high security" variables in noninterference security.

We take the view, learned from others, that program/system development benefits from a comparison of specification programs with (putative) implementations of them, wherever this is possible, via a mathematically defined "refinement" relation whose formulation depends ultimately on a notion of testing that is agreed-to subjectively by all parties concerned [8]. [4] To explain our position unambiguously, we begin by recalling the well known effects of this approach for conventional, sequential programming.

1.1 Elementary testing and refinement for conventional programs

Consider sequential programs operating over a state-space of named variables with fixed types, including a program **abort** that diverges (such as an infinite loop). We allow demonic nondeterminsm, statements such as $\mathsf{x}:=0 \sqcap \mathsf{x}:=1$, in the now-conventional way in which they represent equally abstraction (we do not care whether x is assigned 0 or 1, as long as it is one or the other), on the one hand, or unpredictable and arbitrary run-time choice on the other.

Having determined a "specification" program S, we address the question of whether we are prepared to accept some program I that purports to "implement" it. Although there is nowadays a widely accepted answer to this, we imagine that we are considering the question for the first time and that we are hoping to find an answer that everybody will accept. For that we search for a test on programs that is "elementary" in the sense that it is conceptually simple and that no "reasonable" person could ever argue that S is implemented by I if it is the case that S always passes the test but I might fail it. [5]

[4] We say "wherever this is possible" since there are many aspects of system development that cannot be pinned down mathematically. But –we argue– those that can be, should be.

[5] There is a possibly dichotomy here between "may testing" and "must testing," and we are taking the latter in this example: if S must pass a certain test, then so must I if it is to be considered an implementation.

A common choice for such an *elementary test* is "can diverge," where divergence is considered to be a bad thing: using it, our criterion becomes "if I indeed implements S and I can diverge, then it must be possible for S to diverge also." We note that the elementary test cannot be objectively justified: it is an "axiom" of the approach that will be built on it; and it is via the subjective axioms (in any approach) that we touch reality, where we avoid an infinite definitional regress.

The elementary test provides an "only if" answer to the implementation question, but not an "if." That is, we do not say that I implements S *if* either I never fails the test or S might fail it: this is not practical, because of context. For an example, let S be **if** $\mathsf{x}\neq 0$ **then abort fi** and let I be simply **abort**. Then indeed S passes the test if I does (because they both fail); but we cannot accept I generally as a replacement for S because context $\mathsf{x}:=0;S$ "protects" S, and passes the test as a whole; but the same context does not protect I, since $\mathsf{x}:=0;I$ (still) fails. This illustrates the inutility of the elementary view taken on its own, and it shows that we need a more sophisticated comparison in order to have a practical tool that respects contexts. (Thus it is clear above that we must add "if executed from the same initial state.") The story leads on from here to a definition, ultimately, of sequential-program refinement ($\sqsubseteq$) as the unique relation such that [6]

(i) *soundness* If $S\sqsubseteq I$ then for all contexts $\mathcal{C}$ we have that $\mathcal{C}(I)$ passes the elementary test if $\mathcal{C}(S)$ does, and
(ii) *completeness* If $S\not\sqsubseteq I$ then there is some context $\mathcal{C}$ such that $\mathcal{C}(I)$ fails the elementary test although $\mathcal{C}(S)$ passes it.

That relation turns out to have the *direct* definition that $S\sqsubseteq I$ just when, for all initial states s, if executing I from s can deliver some final state s' then –from s again– either S can deliver s', as well, or S can diverge. Crucially, it is the direct definition that allows ($\sqsubseteq$) to be determined without examining all possible contexts.

1.2 Elementary testing and refinement for probabilistic noninterference-secure programs

In attempting to follow the trajectory of §1.1 into the modern context of noninterference and probability, we immediately run into the problem that there are competing notions of elementary test. Here are just four of them:

Bayes Risk [34, 5, 1, 2] is based on the probability an attacker can reveal a high-security, "hidden" variable h using a single query "Is h equal to h?" where h is some value in h's type. Here (and below) the elementary testing of S wrt. I requires that the probability of revealing h in I cannot be higher than it is in S.

[6] We say "a" rather thean "the" definition of refinement because this is just an example: other elementary tests, and other possible contexts, lead naturally to other definitions.

marginal guesswork [30, 15] is measured in terms of how many queries of the form "Is h equal to h?" are needed to determine h's value with a given probability.

Shannon Entropy [33] is related to the use of multiple queries of the form "Is h in some set H?" where H is a subset of h's type.

guessing entropy [19, 15] is the average number of "Is h equal to h?" guesses necessary to determine h's value.

Not only do these criteria compete for popularity, it turns out that on their own they are not even objectively comparable. For instance, Pliam [30] finds that there can be no general ordering between marginal guesswork and Shannon Entropy: that is, from a marginal-guesswork judgement of whether S passes all tests that I does, there is no way to determine whether the same would hold for Shannon-entropy judgements, nor vice versa. Similarly, Smith has compared Bayes Risk and Shannon Entropy, and claims that these measures are inconsistent in the same sense [34]. The general view seems to be that none of these (four) methods can be said to be generally more- or less discriminating than any of the others.

In spite of the above, *one of our contributions here* is to show that Bayes Risk is maximally discriminating among those four if context is taken into account.

1.3 Features of our approach: a summary

Our most significant deviation from traditional noninterference is that, rather than calculating security measures of programs in isolation, instead we focus on comparing security measures *between* programs: typically one is supposed to be a specification, and another is supposed to be an implementation of it. What we are looking for is an implementation that is at least as secure as its specification.

Since we never consider the security of programs in isolation, an advantage is that it is possible easily to arrange certain kinds of permissible information flow. For example whenever $s \geq i$ holds, a program I that leaks only the i low-order bits of a hidden integer h is secure with respect to a specification S that leaks the s low-order bits of h — that is, for any implementation of S, the leaking of up to s low-order bits of h is allowed but no more. This way we sometimes can avoid separate tools for declassification: to allow an implementation to release (partial) information, we simply arrange that its specification does so.

Typically it is both functional- and security properties (however we measure them) that are of interest. As such, we would like to define a relation ($\sqsubseteq$) between these programs so that $S \sqsubseteq I$ just when *implementation* I has all the functional and the security properties that *specification* S does, where "all" is interpreted within our terms of reference. For incremental, compositional reasoning with such an order, it has been known from the very beginning [37] that the *refinement* relation ($\sqsubseteq$) must satisfy two key technical properties:

Transitivity If $S \sqsubseteq M \sqsubseteq I$ then also $S \sqsubseteq I$. Because of this a comparison between two large programs S, I can be carried out via $S \sqsubseteq M_1 \sqsubseteq \cdots \sqsubseteq M_N \sqsubseteq I$ through many small steps over a long time.

Monotonicity of contexts If $S \sqsubseteq I$ then also $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$, where $\mathcal{C}$ is any program context. Because of this, a large comparison can be carried out via many small steps *independently* by a large programming team working in parallel.

As argued above, since our comparisons rest ultimately on subjective criteria for failure, we reduce that dependency on what is essentially an arbitrary choice by making those criteria as elementary as possible: when can you be *absolutely sure* that $S \not\sqsubseteq I$, that refinement should fail? For this purpose we identify an elementary testing relation ($\preceq$) based on Bayes Risk, such that if $S \not\preceq I$ then I "certainly" (but still subjectively) does not satisfy the specification S in terms of "reasonable" functional- and probabilistically secure properties.

Because our ($\preceq$) is not respected by all contexts (there exist programs S, I and context $\mathcal{C}$ such that $S \preceq I$, yet $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$ in spite of that) our relation ($\sqsubseteq$) is chosen so that it is smaller –i.e. more restrictive– than ($\preceq$), so that it excludes just those "apparent" refinements that can be voided by context.

Our refinement relation is the *compositional closure* of ($\preceq$), the largest relation ($\sqsubseteq$) such that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq \mathcal{C}(I)$ for all possible contexts $\mathcal{C}$. Abusing terminology slightly, we will for simplicity say that ($\sqsubseteq$) is *compositional* just when it is respected by all possible contexts $\mathcal{C}$ (whereas strictly speaking we should say that all such $\mathcal{C}$'s are ($\sqsubseteq$)-monotonic). Further, we note that if we define equivalence $A \sim B$ to be "bi-refinement" $A \sqsubseteq B$ and $B \sqsubseteq A$ then monotonicity of ($\sqsubseteq$) implies that ($\sim$) is is a congruence for all contexts $\mathcal{C}$.

There are two further, smaller idiosyncracies of our approach. The first is that we allow the high-security, "hidden" variables to be assigned-to by the program, so that it is the secrecy of the *final* value h' of h that is of concern to us, not the initial value h. This is because we could not otherwise meaningfully compare functional properties, nor would we be able to treat (sequential) compositional contexts. The other difference, more a position we take, is that we allow an attacker both *perfect recall* and an awareness of *implicit flow*: that the intermediate values of low-security "visible" program variables are observable, even if subsequently overwritten; and that the control-flow of non-atomic program statements is observable. As shown in our case study (§8.3) it is this which allows us to model distributed applications: there, the values of intermediate variables can be observed (and recalled) if they are sent on an insecure channel, and the control flow of a program may be witnessed (for example) by observing which request an agent is instructed to fulfill.

In summary, our TECHNICAL CONTRIBUTION is that we (i) give a sequential semantics for probabilistic noninterference, (ii) define the above order ($\preceq$) based on Bayes Risk, (iii) show it is *not* compositional, (iv) identify a **compositional** subset of it, a *refinement* order ($\sqsubseteq$) such that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq \mathcal{C}(I)$ for all contexts $\mathcal{C}$ and (v) show that ($\sqsubseteq$) is in fact the compositional **closure** of ($\preceq$), so that in fact we have $S \not\sqsubseteq I$ *only* when $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$ for some $\mathcal{C}$.

Finally, we note (vi) that ($\sqsubseteq$) is sound for the other three, competing notions of elementary test and that therefore Bayes-Risk testing, with context, is maximally discriminating among them.

These technical contributions further our general goal of structuring secure protocols hierarchically and then designing/verifying them in separate pieces, a claim that we illustrate by showing how our model and our secure-program ordering may be used to give an incremental development of *The Three Judges*, an "anonymous majority" protocol we constructed precisely to make this point.

2 A probabilistic, noninterference sequential semantics

We identify *visible* variables (low-security), typically v in some finite type $\mathcal{V}$, and *hidden* variables (high-security), typically h in finite $\mathcal{H}$. Variables are in $\mathsf{sans\ serif}$ to distinguish them from (decorated) values $v{:}\,\mathcal{V}, h{:}\,\mathcal{H}$ they might contain. [7]

As an example, let hidden $\mathsf{h}{:}\,\{0,1,2\}$ represent one of three boxes: Box 0 has two black balls; Box 1 has one black- and one white ball; and Box 2 has two white balls. Then let $\mathsf{v}{:}\,\{w,b,\bot\}$ represent a ball colour: *white*, *black* or *unknown*. Our first experiment in this system is Program S, informally written $\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:\in}\,\{\!\{w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}}\}\!\}; \mathsf{v}{:=}\,\bot$, that chooses box h uniformly, and then draws a ball v from that Box h: from the description above (and the code) we can see that with probability $\mathsf{h}/2$ the ball is white, and with probability $1{-}\mathsf{h}/2$ it is black. Then the ball is replaced. A typical security concern is "How much information about h is revealed by its assignments to v?"

We use this program, and that question, to motivate our program syntax and semantics, to make Program S the above program precise and to provide the framework for asking –and answering– such security questions.

We begin by introducing *distribution* notation, generalising the notations for naïve set theory.

2.1 Distributions: explicit, implicit and expected values over them

We write function application as $f.x$, with "." associating to the left. Operators without their operands are written between parentheses, as $(\preceq)$ for example. *Set* comprehensions are written as $\{s{:}\,S \mid G \bullet E\}$ meaning the set formed by instantiating bound variable s in the expression E over those elements of S satisfying formula G. [8]

By $\mathbb{D}S$ we mean the set of *discrete sub-distributions* on set S that sum to no more than one, and $\mathsf{D}S$ means the *full* distributions that sum to one exactly. The *support* $\lceil\delta\rceil$ of (sub-)distribution $\delta{:}\,\mathbb{D}S$ is those elements s in S with $\delta.s{\neq}0$, and the *weight* $\sum\delta$ of a distribution is $(\sum s{:}\,\lceil\delta\rceil \bullet \delta.s)$, so that full distributions have

[7] We say hidden and visible, rather than high- and low security, because of the connection with data refinement where the same technical issues occur but there are no security implications.

[8] This is a different order from the usual notation $\{E \mid s{\in}S{\wedge}G\}$, but we have good reasons for using it: calculations involving both sets and quantifications are made more reliable by a careful treatment of bound variables and by arranging that the order $S/G/E$ is the same in both comprehensions and quantifications (as in $(\forall s{:}\,S \mid G \bullet E)$ and $(\exists s{:}\,S \mid G \bullet E)$).

weight 1. Distributions can be scaled and summed according to the usual pointwise extension of arithmetic to real-valued functions, so that $(c{*}\delta).s$ is $c{*}(\delta.s)$ for example; the *normalisation* of a (sub-)distribution δ is defined $[\delta] := \delta / \sum \delta$.

Here are our notations for *explicit* distributions (cf. set enumerations):

multiple We write $\{\!|x^{@p}, y^{@q}, \cdots, z^{@r}|\!\}$ for the distribution assigning probabilities $p, q, \cdots, r$ to elements $x, y, \cdots, z$ respectively, with $p{+}q{+}\cdots{+}r \leq 1$.

uniform When explicit probabilities are omitted they are uniform: thus $\{\!|x|\!\}$ is the point distribution $\{\!|x^{@1}|\!\}$, and $\{\!|x, y, z|\!\}$ is $\{\!|x^{@\frac{1}{3}}, y^{@\frac{1}{3}}, z^{@\frac{1}{3}}|\!\}$. And $\delta_1 \oplus \delta_2$ is $\delta_{1\,\frac{1}{2}} \oplus \delta_2$.

In general, we write $(\odot\, d{:}\,\delta \bullet E)$ for the *expected value* $(\sum d{:}\,\lceil\delta\rceil \bullet \delta.d * E)$ of expression E interpreted as a random variable in d over distribution δ.[9] If however E is Boolean, then it is taken to be 1 if E holds and 0 otherwise: thus in that case $(\odot\, d{:}\,\delta \bullet E)$ is the combined probability in δ of all elements d that satisfy E.

We write *implicit* distributions (cf. set comprehensions) as $\{\!|d{:}\,\delta \mid R \bullet E|\!\}$, for distribution δ, real expression R and expression E, meaning

$$(\odot\, d{:}\,\delta \bullet R * \{\!|E|\!\}) \;/\; (\odot\, d{:}\,\delta \bullet R) \tag{1}$$

where, first, an expected value is formed in the numerator by scaling and adding point-distribution $\{\!|E|\!\}$ as a real-valued function: this gives another distribution. The scalar denominator then normalises to give a distribution yet again. A missing E is implicitly d itself. If R is missing, however, then $\{\!|d{:}\,\delta \bullet E|\!\}$ is just $(\odot\, d{:}\,\delta \bullet \{\!|E|\!\})$ — in that case we do not multiply by R in the numerator, nor do we divide (by anything).

Thus $\{\!|d{:}\,\delta \bullet E|\!\}$ *maps* expression E in d over distribution δ to make a new distribution on E's type. When R is present, and Boolean, it is converted to 0,1; thus in that case $\{\!|d{:}\,\delta \mid R|\!\}$ is δ's *conditioning* over formula R as predicate on d.

Finally, for *Bayesian belief revision* we let δ be an a-priori distribution over some D, and we let expression R for each d in D be the probability of a certain subsequent result if that d is chosen. Then $\{\!|d{:}\,\delta \mid R|\!\}$ is the a-posteriori distribution over D when that result actually occurs. Thus in the three-box program S let the value first assigned to v be $\hat{v}$. The a-priori distribution over h is uniform, and the probability that the chosen ball is white, that $\hat{v}{=}w$, is therefore $1/3 * (0/2 + 1/2 + 2/2) = 1/2$. But the a-posteriori distribution of h *given that* $\hat{v}{=}w$ is $\{\!|h{:}\,\delta \mid h/2|\!\}$, which from (1) we can evaluate

$$= \quad (\odot\, h{:}\,\{\!|0,1,2|\!\} \bullet \frac{h}{2} * \{\!|h|\!\}) \;/\; (\odot\, h{:}\,\{\!|0,1,2|\!\} \bullet \frac{h}{2}) \quad = \quad \{\!|1^{@\frac{1}{6}}, 2^{@\frac{1}{3}}|\!\} \;/\; \frac{1}{2}\;,$$

that is $\{\!|1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}|\!\}$, to calculate our way to the conclusion that if a white ball is drawn ($\hat{v}{=}w$) then the chance it came from Box 2 is 2/3, the probability of $\mathsf{h}{=}2$ in the a-posteriori distribution.

[9] It is a dot-product between the distribution and the random variable as state-vectors.

2.2 Program denotations over a visible/hidden "split" state-space

We account for the *visible* and *hidden* partitioning of the finite state space $\mathcal{V}\times\mathcal{H}$ in our new model by building *split-states* of type $\mathcal{V}\times\mathsf{D}\mathcal{H}$, whose typical element (v,δ) indicates that we know $\mathsf{v}{=}v$ exactly, but that all we know about h –which is not directly observable– is that it takes value h with probability $\delta.h$.

Programs become functions $(\mathcal{V}\times\mathsf{D}\mathcal{H})\to\mathsf{D}(\mathcal{V}\times\mathsf{D}\mathcal{H})$ from split-states to distributions over them, called *hyper-distributions* since they are distributions with other distributions inside them: the outer distribution is directly visible but the inner distribution(s) over $\mathcal{H}$ are not. Thus for a program P with semantics $[\![P]\!]$, the application $[\![P]\!].(v,\delta)$ is the distribution of final split-states produced from initial (v,δ). Each (v',δ') in the support of that outcome, with probability p say in the outer- (left-hand) D in $\mathsf{D}(\mathcal{V}\times\mathsf{D}\mathcal{H})$, means that with probability p an attacker will observe that v is v' and simultaneously will be able to deduce (via the explicit observation of v and v' and other implicit observations) that h has distribution δ'.

When applied to hyper-distributions, addition, scaling and probabilistic choice $({}_p\oplus)$ are to be interpreted as operations on the outer distributions (as explained in §2.1).

2.3 Program syntax and semantics

The programming language semantics is given in Fig. 1. In this presentation we do not treat loops and, therefore, all our programs are terminating.

When we refer to *classical* semantics, we mean the interpretation of a program without distinguishing its visible and hidden variables, thus as a "relation" of type $(\mathcal{V}\times\mathcal{H})\to\mathsf{D}(\mathcal{V}\times\mathcal{H})$. [10]

Atomic commands Syntactically *atomic* program (fragments), noted $\star$ in Fig. 1, are first interpreted with respect to their classical probabilistic semantics, and are then embedded into the split-state model. To emphasise that they are syntaxtically atomic, we call them "A" (rather than "P") in this section.

Thus the first step is to interpret an atomic program A as a function from $\mathcal{V}\times\mathcal{H}$ -pairs to distributions $\mathsf{D}(\mathcal{V}\times\mathcal{H})$ of them [16, 21] — call that classical interpretation $[\![A]\!]_C$ so that for an initial (v,h) program A produces a final distribution $[\![A]\!]_C.(v,h)$, that is some distribution $\delta'{\in}\mathsf{D}(\mathcal{V}\times\mathcal{H})$.

Given such a distribution δ', define its v-projection $\mathsf{vProj}.\delta'$ to be given by $\{\!\{(v,h){:}\,\delta' \bullet v\}\!\}$, that is the distribution over $\mathcal{V}$, alone, that δ' defines if we ignore (and aggregate) the h-components for each distinct v.

Then define for δ' its v'-conditioning $\mathsf{vCond}.\delta'.v'$, that is the distribution $\{\!\{(v,h){:}\,\delta' \mid v{=}v' \bullet h\}\!\}$ over $\mathcal{H}$ that we get by concentrating on a particular value v'.

[10] Classical relational and *non-probabilistic* semantics over a state-space $\mathcal{V}\times\mathcal{H}$ is strictly speaking $(\mathcal{V}\times\mathcal{H})\leftrightarrow(\mathcal{V}\times\mathcal{H})$ or equivalently $\mathbb{P}((\mathcal{V}\times\mathcal{H})^2)$. Further formulations include however both $(\mathcal{V}\times\mathcal{H})\to\mathbb{P}(\mathcal{V}\times\mathcal{H})$ and $\mathcal{V}\to\mathcal{H}\to\mathbb{P}(\mathcal{V}\times\mathcal{H})$. Because all these are essentially the same, we call $(\mathcal{V}\times\mathcal{H})\to\mathsf{D}(\mathcal{V}\times\mathcal{H})$ a "relational" semantics.

Program type	Program text P	Semantics $[\![P]\!].(v,\delta)$	
Identity	**skip**	$\{\!\{\ (v,\delta)\ \}\!\}$	$\star$
Assign to visible	$\mathsf{v}{:=}\,E.\mathsf{v.h}$	$\{\!\{\ h{:}\,\delta \bullet (E.v.h, \{\!\{h'{:}\,\delta \mid E.v.h'{=}E.v.h\}\!\})\ \}\!\}$	$\star$
Assign to hidden	$\mathsf{h}{:=}\,E.\mathsf{v.h}$	$\{\!\{\ (v, \{\!\{h{:}\,\delta \bullet E.v.h\}\!\})\ \}\!\}$	$\star$
Choose prob. visible	$\mathsf{v}{:\in}\,D.\mathsf{v.h}$	$\{\!\{\ v'{:}\,(\odot\, h{:}\,\delta \bullet D.v.h) \bullet (v', \{\!\{h'{:}\,\delta \mid D.v.h'.v'\}\!\})\ \}\!\}$	$\star$
Choose prob. hidden	$\mathsf{h}{:\in}\,D.\mathsf{v.h}$	$\{\!\{\ (v, (\odot\, h{:}\,\delta \bullet D.v.h))\ \}\!\}$	$\star$
Composition	$P_1; P_2$	$(\odot\,(v',\delta'){:}\,[\![P_1]\!].(v,\delta) \bullet\ [\![P_2]\!].(v',\delta'))$	
General prob. choice	$P_1\ {}_{q.\mathsf{v.h}}\!\oplus\ P_2$	$p * [\![P_1]\!].(v, \{\!\{h{:}\,\delta \mid q.v.h\}\!\})$ $\quad p$ is $(\odot\, h{:}\,\delta \bullet q.v.h)$ $+\ (1{-}p) * [\![P_2]\!].(v, \{\!\{h{:}\,\delta \mid 1{-}q.v.h\}\!\})$	
Probabilistic choice	$P_1\ {}_{p}\!\oplus\ P_2$	$p * [\![P_1]\!].(v,\delta) + (1{-}p) * [\![P_2]\!].(v,\delta)$ $\quad p$ is constant	
Conditional choice	**if** $G.\mathsf{v.h}$ **then** P_t **else** P_f **fi**	$p * [\![P_t]\!].(v, \{\!\{h{:}\,\delta \mid G.v.h\}\!\})$ $\quad p$ is $(\odot\, h{:}\,\delta \bullet G.v.h)$ $+\ (1{-}p) * [\![P_f]\!].(v, \{\!\{h{:}\,\delta \mid \neg G.v.h\}\!\})$	

For simplicity let $\mathcal{V}$ and $\mathcal{H}$ have the same type $\mathcal{X}$. Expression $E.\mathsf{v.h}$ is then of type $\mathcal{X}$, distribution $D.\mathsf{v.h}$ is of type $\mathsf{D}\mathcal{X}$ and expression $G.\mathsf{v.h}$ is Boolean. Expressions p and $q.\mathsf{v.h}$ are of type $[0,1]$.

The syntactically atomic commands marked $\star$ have semantics calculated by taking the classical meaning and then applying Def. 1. The third column for $\star$'d commands is the result of doing that.

Further, the Assign-to semantics are special cases of the Choose-prob. semantics, obtained by making the distribution D equal to the point distribution $\{\!\{E\}\!\}$. And the (simple) probabilistic choice is a special case of the general prob. choice, taking $q.v.h$ to be the constant function always returning p. Finally, conditional choice is the special case of general prob. choice obtained by taking $q.v.h$ to be 1 when $G.v.h$ holds and 0 otherwise.

For distributions in *program texts* we allow the more familiar infix notation ${}_{p}\!\oplus$, so that we can write $\mathsf{h}{:=}\,0\ {}_{\frac{1}{3}}\!\oplus 1$ for $\mathsf{h}{:\in}\,\{\!\{0^{@\frac{1}{3}}, 1^{@\frac{2}{3}}\}\!\}$ and $\mathsf{h}{:=}\,0{\oplus}1$ for the uniform $\mathsf{h}{:\in}\,\{\!\{0,1\}\!\}$. The degenerate cases $\mathsf{h}{:=}\,0$ and $h{:\in}\,\{\!\{0\}\!\}$ are then equivalent, as they should be.

Fig. 1. Split-state semantics of commands

With these two preliminaries, the distribution over $\mathcal{V}{\times}\mathsf{D}\mathcal{H}$ we get by interpreting δ' atomically is defined

$$\mathsf{embed}.\delta' \quad := \quad \{\!\{v'{:}\,\mathsf{vProj}.\delta' \ \bullet\ (v', \mathsf{vCond}.\delta'.v')\}\!\}\ ,$$

which is in essence just the "grouping together" of all elements (v',h') in δ' that have the same v'.

There are two routine steps left to finish off the embedding of whole programs; and they are given here in Def. 1:

Definition 1. *Induced secure semantics for atomic programs* Given a *syntactically atomic* program A we define its *induced secure semantics* $[\![A]\!]$ via

$$[\![A]\!].(v,\delta) \quad := \quad \mathsf{embed}.(\odot\, h{:}\,\delta \ \bullet\ [\![A]\!]_C.(v,h))\ . \tag{2}$$

Thus A is applied to the incoming distribution (v,δ) by applying its classical meaning $[\![A]\!]_C$ to each (v,h)-pair separately, noting that pair's implied weight, and then using those weights to combine the resulting (v',h')-distributions into a single distribution δ' of type $\mathsf{D}(\mathcal{V}\times\mathcal{H})$. That distribution δ' is then embedded into the split-state model as above.

The effect overall is that an embedding imposes the largest possible ignorance of h' that is consistent with seeing v' and knowing the classical semantics $[\![A]\!]_C$. □

We illustrate the definitions in Fig. 1 by looking at some simple examples.

Program **skip** modifies neither v nor h, nor does it change an attacker's knowledge of h. Assignments to v or h can use an expression $E.\mathsf{v}.\mathsf{h}$ or a distribution $D.\mathsf{v}.\mathsf{h}$; and assignments to v might reveal information about h. For example, from Fig. 1 we can explore various assignments to v:

(i) A direct assignment of h to v reveals everything about h:
$$[\![\mathsf{v}{:=}\,\mathsf{h}]\!].(v,\delta) = \{\!\{h{:}\,\delta \bullet (h, \{\!\{h\}\!\})\}\!\}$$
(ii) Choosing v from a distribution independent of h reveals nothing about h:
$$[\![\mathsf{v}{:=}\,0\ {}_{1/3}\!\oplus 1]\!].(v,\delta) = \{\!\{(0,\delta)^{@\frac{1}{3}}, (1,\delta)^{@\frac{2}{3}}\}\!\}$$
(iii) Partially h-dependent assignments to v might reveal something about h:
$$[\![\mathsf{v}{:=}\,\mathsf{h}\ \mathbf{mod}\ 2]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(0,\{\!\{0,2\}\!\})^{@\frac{2}{3}}, (1,\{\!\{1\}\!\})^{@\frac{1}{3}}\}\!\}$$

As a further illustration, we calculate the effect of the first assignment to v in Program S as follows:

$$[\![\mathsf{v}{:}\in \{\!\{w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}}\}\!\}]\!].(v,\{\!\{0,1,2\}\!\})$$

$$= \quad \{\!\{\ v'{:}\,1/3{*}(\{\!\{b\}\!\} + \{\!\{w,b\}\!\} + \{\!\{w\}\!\}) \bullet \qquad \text{“Choose prob. visible”}$$
$$(v',\ \{\!\{h'{:}\,\{\!\{0,1,2\}\!\} \mid \{\!\{w^{@\frac{h'}{2}}, b^{@1-\frac{h'}{2}}\}\!\}.v'\}\!\})\ \}\!\}$$

$$= \qquad \text{“simplify the summation”}$$
$$\{\!\{\ v'{:}\,\{\!\{w,b\}\!\} \bullet (v', \{\!\{h'{:}\,\{\!\{0,1,2\}\!\} \mid \{\!\{w^{@\frac{h'}{2}}, b^{@1-\frac{h'}{2}}\}\!\}.v'\}\!\})\ \}\!\}$$

$$= \qquad \text{“evaluate outer comprehension”}$$
$$\{\!\{\ (w, \{\!\{h'{:}\,\{\!\{0,1,2\}\!\} \mid \tfrac{h'}{2}\}\!\}),\ (b, \{\!\{h'{:}\,\{\!\{0,1,2\}\!\} \mid 1{-}\tfrac{h'}{2}\}\!\})\ \}\!\}$$

$$= \quad \{\!\{\ (w, \{\!\{1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}\}\!\}),\ (b, \{\!\{0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}\}\!\})\ \}\!\}\ . \quad \text{“evaluate conditional distributions”}$$

As for assignments to h, we see that they affect δ directly; thus Choosing hidden h might

(iv) increase our uncertainty of h: $[\![\mathsf{h}{:=}\,0{\oplus}1{\oplus}2]\!].(v,\{\!\{0,1\}\!\}) = \{\!\{(v,\{\!\{0,1,2\}\!\})\}\!\}$
(v) or reduce it: $[\![\mathsf{h}{:=}\,0{\oplus}1]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(v,\{\!\{0,1\}\!\})\}\!\}$
(vi) or leave it unchanged: $[\![\mathsf{h}{:=}\,2{-}\mathsf{h}]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(v,\{\!\{2,1,0\}\!\})\}\!\}$

In all of the above, we saw that the assignment statements were atomic — an attacker may not directly witness the evaluation of their right-hand sides. For instance, the atomic probabilistic choice $\mathsf{v}{:=}\,\mathsf{h}{\oplus}\neg\mathsf{h}$ does *not* reveal which of the equally likely operands of $(\oplus)$ was used.

Non-atomic commands The first, *Composition* $P_1;P_2$, gives an attacker **perfect recall** after P_2 of the visible variable v as it was after P_1, even if P_2 overwrites v.[11] To see the effects of this, we compare the three-box Program S from the start of §2, that is

$$\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:\in}\, \{\!|\, w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \,|\!\}; \mathsf{v}{:=}\,\bot \ ,$$

with the simpler Program I_1 defined $\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:=}\,\bot$ in which no ball is drawn: the final hyper-distributions are respectively

$$\{\!|\ (\bot, \{\!|1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}|\!\}),\ (\bot, \{\!|0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}|\!\})\ |\!\} \qquad (\Delta'_S)$$

and

$$\{\!|\ (\bot, \{\!|0,1,2|\!\})\ |\!\}\ . \qquad (\Delta'_{I_1})$$

We calculated Δ'_S as follows:

$$[\![\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:\in}\, \{\!|\, w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \,|\!\}; \mathsf{v}{:=}\,\bot]\!].(v,\delta)$$

$=$ $[\![\mathsf{v}{:\in}\, \{\!|\, w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \,|\!\}; \mathsf{v}{:=}\,\bot]\!].(v, \{\!|0,1,2|\!\})$ "Choose hidden; Composition"

$=$ $(\odot\,(\hat{v},\hat{\delta}){:}\,[\![\mathsf{v}{:\in}\, \{\!|\, w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \,|\!\}]\!].(v, \{\!|0,1,2|\!\}) \bullet [\![\mathsf{v}{:=}\,\bot]\!].(\hat{v},\hat{\delta}))$ "Composition"

$=$ "assignment $\mathsf{v}{:=}\,\bot$ independent of h"
$(\odot\,(\hat{v},\hat{\delta}){:}\,[\![\mathsf{v}{:\in}\, \{\!|\, w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \,|\!\}]\!].(v, \{\!|0,1,2|\!\}) \bullet \{\!|(\bot,\hat{\delta})|\!\})$

$=$ "Choose prob. visible (see earlier calculation)"
$(\odot\,(\hat{v},\hat{\delta}){:}\,\{\!|(w, \{\!|1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}|\!\}), (b, \{\!|0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}|\!\})|\!\} \bullet \{\!|(\bot,\hat{\delta})|\!\})$

$=$ $\{\!|(\bot, \{\!|1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}|\!\}), (\bot, \{\!|0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}|\!\})|\!\}$. "evaluate expected value"

In neither case Δ'_S nor Δ'_{I_1} does the final value $\bot$ of v reveal anything about h. But Δ'_{I_1} is a point (outer) distribution (thus concentrated on a single split-state), whereas Δ'_S is a uniform distribution over two split-states each of which recalls implicitly the observation of an intermediate value $\hat{v}$ of v that was made during the execution leading to that state. Generally, if two split-states (v',δ'_1) and (v',δ'_2) occur with $\delta'_1{\neq}\delta'_2$ then it means an attacker can deduce whether h's distribution is δ'_1 or δ'_2 even though v has the same final value v' in both cases. Although the direct evidence $\hat{v}$ has been overwritten, the distinct split-states preserve the attacker's deductions from it.

The meaning of *General prob. choice* $P_1\,{}_{p.\mathsf{v}.\mathsf{h}}{\oplus}\,P_1$ –of which both *Probabilistic choice* and *Conditional choice* are specific instances– makes it behave like $[\![P_1]\!]$ with probability $p.\mathsf{v}.\mathsf{h}$ and $[\![P_2]\!]$ with the remaining probability. The definition allows an attacker to observe which branch was taken and, knowing that, she might be able to deduce new facts about h. Thus unlike for (v) above we have $[\![\mathsf{h}{:=}\,0 \oplus \mathsf{h}{:=}\,1]\!].(v,\delta) = \{\!|(v, \{\!|0|\!\}), (v, \{\!|1|\!\})|\!\}$, which is an example of **implicit flow**.

[11] It is effectively the Kleisli composition over the outer distribution.

A similar implicit information flow in any Conditional choice with guard $G.\mathsf{v.h}$ makes it possible for an attacker to deduce the value of the guard exactly.

For General prob. choice $P_1 \,{}_{p.\mathsf{v.h}}\!\oplus\, P_2$ however, the implicit flow might only partially reveal the value of the expression $p.\mathsf{v.h}$. For example, suppose we execute the probabilistic assignment $\mathsf{h}{:=}\,\frac{1}{4} \oplus \frac{1}{2}$, which establishes that h is either $\frac{1}{4}$ or $\frac{1}{2}$ with equal probability of each: its output is $\{\!\{(v, \{\!\{\frac{1}{4}, \frac{1}{2}\}\!\})\}\!\}$. Then we execute program $\mathbf{skip}\,{}_{\mathsf{h}}\!\oplus\,\mathbf{skip}$ from there, and we find that we do not *entirely* discover the value of h. But still we do discover something: we find that

$$[\![\mathbf{skip}\,{}_{\mathsf{h}}\!\oplus\,\mathbf{skip}]\!].(v, \{\!\{\tfrac{1}{4}, \tfrac{1}{2}\}\!\}) \quad = \quad \{\!\{(v, \{\!\{\tfrac{1}{4}^{@\frac{1}{3}}, \tfrac{1}{2}^{@\frac{2}{3}}\}\!\})^{@\frac{3}{8}}, (v, \{\!\{\tfrac{1}{4}^{@\frac{3}{5}}, \tfrac{1}{2}^{@\frac{2}{5}}\}\!\})^{@\frac{5}{8}}\}\!\}\ ,$$

and see that indeed the chance of guessing h's value has increased, though we still do not know it for certain. Our probability initially of guessing h is $1/2$. But after the choice we will guess $\mathsf{h}{=}\frac{1}{2}$ when we see the choice went left, which happens with probability $3/8$; but if we saw the choice going right we will guess $\mathsf{h}{=}\frac{1}{4}$, which happens with probability $5/8$. Our average chance of guessing h is thus $(2/3){*}(3/8){+}(3/5){*}(5/8) = 5/8$, which is more than the $1/2$ it was initially: that increased knowledge is what was revealed by the $({}_{\mathsf{h}}\!\oplus)$.

3 The Bayes-Risk based elementary testing order

The elementary testing order comprises functional- and security characteristics.

Say that two programs are *functionally equivalent* iff from the same input they produce the same *overall* output distribution [16, 21], defined for hyper-distribution Δ' to be $\mathsf{ft}.\Delta' {:=} \{\!\{(v', \delta') {:} \Delta'; h' {:} \delta' \bullet (v', h')\}\!\}$. [12] We consider state-space $\mathcal{V}{\times}\mathcal{H}$ jointly, i.e. not $\mathcal{V}$ alone, because differing distributions over h alone can be revealed by the context $(-\,;\, \mathsf{v}{:=}\,\mathsf{h})$ that appends an assignment $\mathsf{v}{:=}\,\mathsf{h}$.

We measure the *security* of a program with "Bayes Risk" [34, 5, 1, 2], which determines an attacker's chance of guessing the final value of h in one try. The most effective such attack is to determine which split-state (v', δ') in a final hyper-distribution actually occurred, and then to guess that h has some value h' that maximises δ', i.e. so that $\delta' h' = \sqcup\delta'$. For a whole hyper-distribution we average the attacks over its elements, weighted by the probability it gives to each, and so we we define the **Bayes Vulnerability** of Δ' to be $\mathsf{bv}.\Delta' {:=} (\odot\, (v', \delta') {:} \Delta' \bullet \sqcup\delta')$. [13]

For Program S the vulnerability is the chance of guessing h by remembering v's intermediate value, say $\hat{v}$, and then guessing that h at that point had the value most likely to have produced that $\hat{v}$: when $\hat{v}{=}w$ (probability $1/2$), guess $\mathsf{h}{=}2$;

[12] Two program *texts* $P_{\{1,2\}}$ denote functionally equivalent secure programs just when their classical denotations agree, that is when $[\![P_1]\!]_C{=}[\![P_2]\!]_C$. The function ft expresses that semantically, and the connection is thus that $[\![P_1]\!]_C{=}[\![P_2]\!]_C$ just when $\mathsf{ft}.([\![P_1]\!].(v,\delta)){=}\mathsf{ft}.([\![P_2]\!].(v,\delta))$ for all (v,δ).

[13] We use vulnerability rather than risk because "greatest chance of leak" is more convenient than the dual "least chance of no leak." Our definition corresponds to Smith's *vulnerability* [34].

when $\hat{v}{=}b$, guess $\mathsf{h}{=}0$. Via $\mathsf{bv}.\Delta'_S$ that vulnerability is $1/2{*}2/3 + 1/2{*}2/3 = 2/3$. For I_1, however, there is no "leaking" $\hat{v}$, and so it is less vulnerable, having $\mathsf{bv}.\Delta'_{I_1} = 1/3$.

The elementary testing order on hyper-distributions is then defined $\Delta_S \preceq \Delta_I$ iff $\mathsf{ft}.\Delta_S{=}\mathsf{ft}.\Delta_I$ and $\mathsf{bv}.\Delta_S{\geq}\mathsf{bv}.\Delta_I$, and it extends pointwise to the **elementary testing order** on whole programs. That is, we say that $S \preceq I$ just when for corresponding inputs (i) S, I are functionally equivalent and (ii) the vulnerability of I is no more than the vulnerability of S. Thus $S \preceq I_1$ because they are functionally equivalent and the vulnerabilities of S, I_1 are $2/3, 1/3$ resp.

The direction of the inequality ($\preceq$) corresponds to increasing security (and thus decreasing vulnerability). This agrees with other notions of security that increase with increasing entropy of the hidden distribution.

4 Non-compositionality of the elementary testing order

Although $S \not\preceq I$ is an (elementary) failure of implementation, the complementary $S \preceq I$ is not necessarily a success: it is quite possible, in spite of that, that there is a context $\mathcal{C}$ with $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$. That is, simply having $S \preceq I$ does not mean that I is safe to use in place of S in general.

Thus for stepwise development we require more than just $S \preceq I$: we must ensure that $\mathcal{C}(S) \preceq \mathcal{C}(I)$ holds for *all* contexts $\mathcal{C}(\cdot)$ in which S, I might be placed — and we do not know in advance what those contexts might be.

Returning to the boxes, we consider now another variation Program I_2 in which both Boxes 0,1 have two black balls: thus the program code becomes $\mathsf{h}{:=}\,0 \oplus 1 \oplus 2; \mathsf{v}{:\in}\, \{\!\{ w^{@(\mathsf{h} \div 2)}, b^{@1-(\mathsf{h} \div 2)} \}\!\}; \mathsf{v}{:=}\,\bot$ with final hyper-distribution

$$\{\!\{\ (\bot, \{\!\{2\}\!\})^{@\frac{1}{3}}\ ,\ (\bot, \{\!\{0,1\}\!\})^{@\frac{2}{3}}\ \}\!\}\ . \qquad (\Delta'_{I_2})$$

The vulnerability of I_2 is $1/3{*}1{+}2/3{*}1/2$, again $2/3$ so that $S \preceq I_2$. Now if context $\mathcal{C}$ is defined $(-\,;\, \mathsf{h}{:=}\,\mathsf{h}{\div}2)$, the vulnerability of $\mathcal{C}(S)$ is $1/2{*}2/3 + 1/2{*}1 = 5/6$: it is more than for S alone because there are fewer final h-values to choose from. But for $\mathcal{C}(I_2)$ it is greater still, at $1/3{*}1 + 2/3{*}1 = 1$.

Thus $S \preceq I_2$ but $\mathcal{C}(S) \not\preceq \mathcal{C}(I_2)$, and so ($\preceq$) is not compositional. This makes ($\preceq$) unsuitable, on its own, for secure-program development of any size; and its failure of compositionality is the principal problem we solve.

5 The refinement order, and compositional closure

The compositional closure of an "elementary" partial order over programs, call it ($\leq_E$), is the largest subset of that order that is preserved by composition with other programs, that is with being placed in a program context. Call that closure ($\leq_C$).

The utility of ($\leq_C$) is first that $A \leq_C B$ implies $A \leq_E B$, so that $A \leq_C B$ suffices if $A \leq_E B$ is all that we want: but it implies further that $\mathcal{C}(A) \leq_E \mathcal{C}(B)$ for all contexts $\mathcal{C}$, as well. Its being the *greatest* such subset of ($\leq_E$) means that it

relates as many programs as possible, never claiming that $A \not\preceq_C B$ unless there is some context $\mathcal{C}$ that forces it to do so because in fact $\mathcal{C}(A) \not\preceq_E \mathcal{C}(B)$.

Thus to address the non-compositionality exposed in §4, we seek the *compositional closure* of $(\preceq)$, the unique *refinement* relation $(\sqsubseteq)$ such that (*soundness*) if $S \sqsubseteq I$ then for all $\mathcal{C}$ we have $\mathcal{C}(S) \preceq \mathcal{C}(I)$; and (*completeness*) if $S \not\sqsubseteq I$ then for some $\mathcal{C}$ we have $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$. Soundness gives refinement the property (§4) we need for stepwise development; and completeness makes refinement as liberal as possible consistent with that.

We found above that $S \not\sqsubseteq I_2$; we show later (§6.4) that we do have $S \sqsubseteq I_1$.

6 Constructive definition of the refinement order

Although saying thet $(\sqsubseteq)$ is the compositional closure of $(\preceq)$ does define it completely, it is of little use if to establish $S \sqsubseteq I$ in practice we have to evaluate and compare $\mathcal{C}(S) \preceq \mathcal{C}(I)$ for all contexts $\mathcal{C}$. Instead we seek an explicit construction that is easily verified for specific cases. We give a detailed example to help introduce our definition.

For integers x, n, let $x \,\mathbf{rnd}\, n$ be a distribution over the multiple(s) of n closest to x: usually there will be exactly two such multiples, one on either side of x and, in that case, the probabilities of each are inversely proportional to their distance from x. Thus $1 \,\mathbf{rnd}\, 4$ is $\{\!\{ 0^{@\frac{3}{4}}, 4^{@\frac{1}{4}} \}\!\}$ and $2 \,\mathbf{rnd}\, 4$ is $\{\!\{ 0^{@\frac{1}{2}}, 4^{@\frac{1}{2}} \}\!\}$ and $3 \,\mathbf{rnd}\, 4$ is $\{\!\{ 0^{@\frac{1}{4}}, 4^{@\frac{3}{4}} \}\!\}$. If however x happens to be an integer multiple of n then the outcome is definite, a point distribution: thus $0 \,\mathbf{rnd}\, 4 = \{\!\{ 0 \}\!\}$ and $4 \,\mathbf{rnd}\, 4 = \{\!\{ 4 \}\!\}$.

Now consider the two programs

$$\begin{array}{rrl} & P_2 := & \mathsf{h} := 1 \oplus 2 \oplus 3;\ \mathsf{v} :\in \mathsf{h} \,\mathbf{rnd}\, 2;\ \mathsf{v} := \mathsf{h} \,\mathbf{mod}\, 2 \\ \text{and} & P_4 := & \mathsf{h} := 1 \oplus 2 \oplus 3;\ \mathsf{v} :\in \mathsf{h} \,\mathbf{rnd}\, 4;\ \mathsf{v} := \mathsf{h} \,\mathbf{mod}\, 2\ . \end{array} \tag{3}$$

Both reveal $\mathsf{h} \,\mathbf{mod}\, 2$ in v's final value v', but each P_n also reveals in the overwritten visible $\hat{v}$, say, something about $\mathsf{h} \,\mathbf{rnd}\, n$; and intuition suggests that $P_n \sqsubseteq P_m$ for $n \leq m$ only. Yet in fact the vulnerability is $5/6$ for both $P_{2,4}$, which we can see from their final hyper-distributions; they are Δ'_{P_2} and Δ'_{P_4} given by

$$\{\!\{\ (0, \{\!\{2\}\!\})^{@\frac{1}{3}},\ (1, \{\!\{1\}\!\})^{@\frac{1}{6}}, (1, \{\!\{1,3\}\!\})^{@\frac{1}{3}}, (1, \{\!\{3\}\!\})^{@\frac{1}{6}}\ \}\!\} \qquad (\Delta'_{P_2})$$

$$\{\!\{\ (0, \{\!\{2\}\!\})^{@\frac{1}{3}}, \underbrace{(1, \{\!\{1^{@\frac{3}{4}}, 3^{@\frac{1}{4}}\}\!\})^{@\frac{1}{3}}}, (1, \{\!\{1^{@\frac{1}{4}}, 3^{@\frac{3}{4}}\}\!\})^{@\frac{1}{3}}\ \}\!\} \qquad (\Delta'_{P_4})$$

With overall probability $1/3{*}3/4 + 1/3{*}1/4 = 1/3$ the final v' will be 1 and $\hat{v}$ will be 0; since v' is 1 then h must be 1 or 3; but if $\hat{v}$ was 0 that h is three times as likely to have been 1.

so that e.g. $1/3{*}1 + 1/3{*}3/4 + 1/3{*}3/4 = 5/6$ for P_4. The overall distribution of (v', h') is $\{\!\{(0,2),(1,1),(1,3)\}\!\}$ in both cases, so that $P_{2,4}$ are functionally equivalent; but they have different residual uncertainties of h.

6.1 Hyper-distributions as partitions of fractions

In our definition of refinement we will consider the hyper-distributions corresponding to each value of v separately.

In the example above, if we consider just the h-distributions associated with $v'{=}1$ then we can, by multiplying through their associated probabilities from the hyper-distributions, present them as a collection of *fractions*, that is sub-distributions over $\mathcal{H}$. We call such collections *partitions* and here they are given for P_2 and P_4 respectively by [14]

$$\text{when } v'{=}1 \quad \begin{cases} \Pi'_{P_2}\colon & \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle \\ \Pi'_{P_4}\colon & \langle \{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{4}} \}\!\} \rangle \,. \end{cases} \tag{4}$$

In general, let the function $\mathsf{fracs}.\Delta.v$ for hyper-distribution Δ and value v give the partition of fractions extracted from Δ for $\mathsf{v}{=}v$, as we extracted $\Pi'_{\{2,4\}}$ from $\Delta'_{\{2,4\}}$ and $v'{=}1$ at (4) above.

6.2 Operations on fractions and partitions

Distribution operations such as support ($\lceil\cdot\rceil$) and weight ($\sum$) and normalise $[\cdot]$ apply to fractions, and for example we have that $\{\!\{ 1^{@\frac{1}{6}} \}\!\} + \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}$ is $\{\!\{ 1^{@\frac{1}{3}}, 3^{@\frac{1}{6}} \}\!\}$ and $\sum \{\!\{ 1^{@\frac{1}{3}}, 3^{@\frac{1}{6}} \}\!\}$ is $1/2$ and $[\{\!\{ 1^{@\frac{1}{3}}, 3^{@\frac{1}{6}} \}\!\}]$ is $\{\!\{ 1^{@\frac{2}{3}}, 3^{@\frac{1}{3}} \}\!\}$. For partitions Π we write $\sum \Pi$ as shorthand for $\langle (\sum \pi\colon \Pi \bullet \pi) \rangle$, so that

$$\sum \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\} \rangle \qquad \text{is} \qquad \langle \{\!\{ 1^{@\frac{1}{3}}, 3^{@\frac{1}{6}} \}\!\} \rangle \,.$$

Note that the sum of a partition is still a partition, albeit always with only a single fraction in it. Scaling, when applied partition is applied pointwise to each of its fractions. An empty partition is written $\langle\rangle$, and a zero(-weight) fraction is written $\{\!\{\}\!\}$; thus $\langle \{\!\{\}\!\} \rangle$ is a zero-weight partition containing exactly one fraction.

Finally, the Bayes Vulnerability of a partition $\mathsf{bv}.\Pi$ is $(\sum \pi\colon \Pi \bullet \sqcup\pi)$, and the Bayes Vulnerability of a hyper-distribution may be equivalently expressed using partitions as $(\sum v\colon \mathcal{V} \bullet \mathsf{bv}.(\mathsf{fracs}.\Delta.v))$.

6.3 Relationships between fractions and partitions

Say that two non-zero fractions $\pi_{\{1,2\}}$ are *similar*, written $\pi_1 {\approx} \pi_2$, just when their normalisations are equal, that is when $[\pi_1]{=}[\pi_2]$ so that they are multiples of each other: this is an equivalence relation. For example we have $\{\!\{ 1^{@\frac{1}{3}}, 2^{@\frac{2}{3}} \}\!\} \approx \{\!\{ 1^{@\frac{1}{4}}, 2^{@\frac{1}{2}} \}\!\}$ because both normalise to the former.

Say that a partition is *reduced* just when it contains no two similar fractions, and no zero fractions at all. [15] For any hyper-distribution Δ and value v, we have that $\mathsf{fracs}.\Delta.v$ is in reduced form by construction. Thus partitions are more expressive than hyper-distributions.

The *reduction* of a partition is obtained by by adding-up all its similar fractions and removing its all-zero fractions, that is by *reducing* it, and we say that

[14] Strictly speaking, partitions are multisets of fractions, i.e. without order but possibly having repeated elements.

[15] Allowing zero fractions, in the unreduced case, simplifies some proofs.

two partitions $\Pi_{\{1,2\}}$ are *similar*, written $\Pi_1 \approx \Pi_2$, just when they have the same reduction. Thus for example we have

$$\langle \{\!\{\}\!\}, \{\!\{0^{@\frac{1}{3}}\}\!\}, \{\!\{1^{@\frac{1}{6}}, 2^{@\frac{1}{6}}\}\!\}, \{\!\{1^{@\frac{1}{6}}, 2^{@\frac{1}{6}}\}\!\} \rangle \;\approx\; \langle \{\!\{0^{@\frac{1}{3}}\}\!\}, \{\!\{1^{@\frac{1}{9}}, 2^{@\frac{1}{9}}\}\!\}, \{\!\{1^{@\frac{2}{9}}, 2^{@\frac{2}{9}}\}\!\} \rangle \,,$$

because both reduce to $\langle \{\!\{0^{@\frac{1}{3}}\}\!\}, \{\!\{1^{@\frac{1}{3}}, 2^{@\frac{1}{3}}\}\!\} \rangle$. If two partitions are similar then. for any distribution δ over $\mathcal{H}$, the probability that an attacker may deduce that h is distributed according to δ is the same in either partition.

Say that one partition Π_1 is *as fine as* another Π_2, written $\Pi_1 \sqsubseteq\!\!\!\!\sim \Pi_2$, just when Δ_2 can be obtained by adding-up one or more groups of fractions in Δ_1. Thus for example we have

$$\langle \{\!\{0^{@\frac{1}{3}}\}\!\}, \{\!\{1^{@\frac{1}{9}}, 2^{@\frac{2}{9}}\}\!\}, \{\!\{1^{@\frac{2}{9}}, 2^{@\frac{1}{9}}\}\!\} \rangle \quad \sqsubseteq\!\!\!\!\sim \quad \langle \{\!\{0^{@\frac{1}{3}}\}\!\}, \{\!\{1^{@\frac{1}{3}}, 2^{@\frac{1}{3}}\}\!\} \rangle$$

by adding-up the second and third fractions on the left. For as-fine-as the added-up fractions do not have to be similar: if however they *are* similar, then we have $\Pi_1 \approx \Pi_2$ as well as $\Pi_1 \sqsubseteq\!\!\!\!\sim \Pi_2$; if they are not similar, we can write $\Pi_1 \sqsubseteq\!\!\!\!\not\sim \Pi_2$.

Combining two dissimilar fractions in a partition represents removal of the implicit observations that distinguished them. Hence if $\Pi_1 \sqsubseteq\!\!\!\!\not\sim \Pi_2$ then partition Π_2 conceals h strictly better than Π_1 does.

Note that in both cases $\Pi_1 \approx \Pi_2$ and $\Pi_1 \sqsubseteq\!\!\!\!\sim \Pi_2$ we have $\sum \Pi_1 = \sum \Pi_2$, i.e. that neither relation allows a change in the overall probability assigned to each of the elements.

6.4 Constructive definition of refinement

We use the relations $(\approx)$ and $(\sqsubseteq\!\!\!\!\sim)$ between partitions to define refinement.

Definition 2. *Secure refinement* We say that hyper-distribution Δ_S is securely-refined by Δ_I, written $\Delta_S \sqsubseteq \Delta_I$, just when for every v there is some intermediate partition Π of fractions so that first (i) $\mathsf{fracs}.\Delta_S.v$ is similar to Π and then (ii) Π is as fine as $\mathsf{fracs}.\Delta_I.v$.[16] That is, we have

$$\Delta_S \sqsubseteq \Delta_I \quad \text{iff} \quad \mathsf{fracs}.\Delta_S.v \;\approx\; \Pi \;\sqsubseteq\!\!\!\!\sim\; \mathsf{fracs}.\Delta_I.v \quad \text{for some partition } \Pi.$$

The fractions of Δ_S are first split-up into similar sub-fractions; and then some of those sub-fractions are rejoined to create the fractions of Δ_I.

Refinement of hyper-distributions extends pointwise to the programs that produce them. □

Note that since both $(\approx)$ and $(\sqsubseteq\!\!\!\!\sim)$ preserve partition-sum, we have that $(\sqsubseteq)$ from Def. 2 implies functional equality. Informally speaking, refinement may not change the functional behaviour of a secure program, but it may reduce the implicit observations available to an attacker, and hence the deductions an attacker can make about h.

[16] In our earlier *qualitative* work [27] refinement reduces to taking unions of equivalence classes of hidden values, so-called "Shadows." Köpf et al. observe similar effects [15].

We return to Δ'_S for an example, getting $\langle \{\!\{ 1^{@\frac{1}{6}}, 2^{@\frac{1}{3}} \}\!\}, \{\!\{ 0^{@\frac{1}{3}}, 1^{@\frac{1}{6}} \}\!\} \rangle$ for $\mathsf{fracs}.\Delta'_S.\bot$ by multiplying through. For Δ'_{I_1} we get $\langle \{\!\{ 0^{@\frac{1}{3}}, 1^{@\frac{1}{3}}, 2^{@\frac{1}{3}} \}\!\} \rangle$ similarly for $\mathsf{fracs}.\Delta'_{I_1}.\bot$. The two fractions of the former sum to the single fraction of the latter, and so $S \sqsubseteq I_1$ according to our definition Def. 2 of secure refinement.

For the more detailed $\Delta'_{P_2} \sqsubseteq \Delta'_{P_4}$ and $v'=1$, we need the intermediate partition $\Pi := \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle$, whose middle two fractions turn out to be equal, thus certainly similar: summing them gives the middle $\{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}$ of Π'_{P_2}, so that $\Pi'_{P_2} \approx \Pi$. On the other hand, summing the first two fractions of Π gives $\{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}$, the first fraction of Π'_{P_4}, and summing the last two give the second fraction of Π'_{P_4}; thus $\Pi \underset{\sim}{\sqsubseteq} \Pi'_{P_4}$. Partition $\langle \{\!\{ 2^{@\frac{1}{3}} \}\!\} \rangle$ deals trivially with $v'=0$, and so indeed we have $P_2 \sqsubseteq P_4$ altogether. In §D we show however that $P_4 \not\sqsubseteq P_2$.

6.5 Properties of refinement

The refinement relation ($\sqsubseteq$) is a partial order (hence it is transitive), and program contexts preserve it (thus it is monotonic). Consequently, we can reason incrementally and compositionally about refinement relation between large programs.

Theorem 1. *Partial order* The refinement relation ($\sqsubseteq$) is a partial order over the set of hyper-distributions; and so, by extension, it is a partial order over programs.

Proof: See §C.1. □

Theorem 2. *Monotonicity of refinement* If $S \sqsubseteq I$ then $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$ for all contexts $\mathcal{C}$ built from programs as defined in Fig. 1.

Proof: See §C.2. □

Furthermore, we define *strict* refinement such that $S \sqsubset I$ when $S \sqsubseteq I$ but $I \not\sqsubseteq S$.

7 Refinement ($\sqsubseteq$) is the compositional closure of ($\preceq$)

In this proof we will manipulate partitions, sequential composition, refinement and Bayes Vulnerability in terms of matrices, as follows.

7.1 Matrix representation and manipulation of partitions

Partitions as matrices Assume *wlog* that $\mathcal{H}$ is the integers $1..H$. For a particular input (v, δ) and a chosen visible output v', a program P will produce as output a partition $\Pi = \mathsf{fracs}.(\llbracket P \rrbracket.(v, \delta)).v'$ over hidden values containing some number F of fractions that we index $1..F$. Each fraction on its own is a vector of length H of probabilities; if we put them together as rows, we get an $F \times H$-matrix that represents the partition as a whole. For example, we have from (4)

the following matrix representations of partitions output from Programs $P_{\{2,4\}}$ for v'=1:

$$\Pi'_{P_2}\colon \quad \begin{pmatrix} 1/6 & 0 & 0 \\ 1/6 & 0 & 1/6 \\ 0 & 0 & 1/6 \end{pmatrix} \qquad\qquad \Pi'_{P_4}\colon \quad \begin{pmatrix} 1/4 & 0 & 1/12 \\ 1/12 & 0 & 1/4 \end{pmatrix} . \tag{5}$$

There are three possible values of h in each case, so that H=3; and P_2's partition has 3 fractions, so that F_2=3 and thus it generates a 3×3 matrix. Program P_4's partition has only 2 fractions, so that F_4=2 and it generates a 2×3 matrix.

For simplicity in the proof, we will arrange that H=F so that all matrices are of the same (square) dimension N×N. This is without loss of generality, since we can extend $\mathcal{H}$ with extra, unused values; and we can extend our partitions with extra, zero fractions. For instance Π'_{P_4} becomes a 3×3 matrix, as Π'_{P_2} is already, if we add an extra row underneath (representing an all-zero fraction):

$$\Pi'_{P_4}\colon \quad \begin{pmatrix} 1/4 & 0 & 1/12 \\ 1/12 & 0 & 1/4 \\ 0 & 0 & 0 \end{pmatrix} . \tag{6}$$

(A) Sequential composition as matrix multiplication In our completeness proof, our program-differentiating context $\mathcal{C}$ will post-compose a probabilistic assignment $\mathsf{h}{:\in}\,D.\mathsf{h}$ so that, for each of its incoming values h, the output value h' will be chosen from the distribution $D.h$, thus with probability $D.h.h'$. In effect the context redistributes variable h in a way that depends on its current value.

We can consider D itself to be an N×N matrix whose value in row h and column h' is just $D.h.h'$. If we do that, then the output partition Π' that results from executing $\mathsf{h}{:\in}\,D.\mathsf{h}$ on input partition Π is just Π×D, where (×) is matrix multiplication. For example, suppose our post-composed context were

$$\mathsf{h}{:\in}\ (\ \{\!\{1^{\mathbb{Q}\frac{1}{2}}, 2^{\mathbb{Q}\frac{1}{4}}, 3^{\mathbb{Q}\frac{1}{4}}\}\!\}\ \textbf{ if } \mathsf{h}{=}1 \textbf{ else }\ \{\!\{2^{\mathbb{Q}\frac{1}{2}}, 3^{\mathbb{Q}\frac{1}{2}}\}\!\}\ ) , \tag{7}$$

so that matrix D would be

$$\begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 0 & 1/2 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix} .$$

From (5) we take the incoming partition Π to the post-composed context (7) to be the outgoing partition Π'_{P_2} from Program P_2, and so determine the outgoing partition Π' from $(P_2'; \mathsf{h}{:\in}\,D.\mathsf{h})$ overall to be

$$\begin{pmatrix} 1/6 & 0 & 0 \\ 1/6 & 0 & 1/6 \\ 0 & 0 & 1/6 \end{pmatrix} \times \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 0 & 1/2 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix} \quad = \quad \begin{pmatrix} 1/12 & 1/24 & 1/24 \\ 1/12 & 1/8 & 1/8 \\ 0 & 1/12 & 1/12 \end{pmatrix} .$$

(B) Refinement as matrix multiplication Also refinement can be formulated as matrix multiplication, since it is essentially a rearranging of fractions within a partition that, therefore, boils down to rearrangement of rows within a matrix. For example, from §6.4 we recall that to refine Π'_{P_2} into Π'_{P_4} we split the middle fraction of the former into two equal pieces, and add them to the other two, and that is achieved by the left-hand matrix in the pre-multiplication shown here:

$$\begin{pmatrix} 1 & 1/2 & 0 \\ 0 & 1/2 & 1 \\ 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1/6 & 0 & 0 \\ 1/6 & 0 & 1/6 \\ 0 & 0 & 1/6 \end{pmatrix} \quad = \quad \begin{pmatrix} 1/4 & 0 & 1/12 \\ 1/12 & 0 & 1/4 \\ 0 & 0 & 0 \end{pmatrix} .$$

In general a partition Π_S is refined by Π_I iff there exists a *refinement matrix* R, a matrix whose columns are non-negative and one-summing, such that $R \times \Pi_S$ equals Π_I. Entry (r, c) of such a refinement matrix describes what proportion of the c^{th} fraction (row) of Π_S is to contribute by addition to the r^{th} fraction of Π_I.

(C) Bayes Vulnerability as matrix multiplication Finally we bring Bayes Vulnerability into the matrix algebra as well. For a partition Π as a matrix, the vulnerability is found by taking the individual row maxima and adding them together: the result is a scalar. Thus for Π'_{P_4}, for example, we have the matrix

$$\begin{pmatrix} \mathbf{1/4} & 0 & 1/12 \\ 1/12 & 0 & \mathbf{1/4} \\ 0 & 0 & 0 \end{pmatrix} \quad \text{with maxima selected by the } \textit{strategy} \text{ matrix } G\text{:} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

whose maxima have been set in bold and are selected by the 1 entries in the matrix G at right. Note that strategy matrices have the same shape as the matrix from which they select, and that they are 0/1 matrices with exactly one 1 per row. [17]

To determine the vulnerability associated with the Π, we calculate

$$(\sqcup \text{ strategy matrices } G \bullet \mathsf{tr}.(G^{\mathbf{T}} \times \Pi)) \tag{8}$$

in general, where $(\cdot)^{\mathbf{T}}$ is matrix transpose and tr takes the *trace* of a square matrix, i.e. the sum of its diagonal. Note that the maximum is actually attained, for some G, since there are only finitely many of them. In this particular case we use the G above to calculate $\mathsf{tr}.(G^{\mathbf{T}} \times \Pi'_{P_4})$, and have therefore

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} \mathbf{1/4} & 0 & 1/12 \\ 1/12 & 0 & \mathbf{1/4} \\ 0 & 0 & 0 \end{pmatrix} \quad = \quad \begin{pmatrix} \mathbf{1/4} & 0 & 1/12 \\ 0 & \mathbf{0} & 0 \\ 1/12 & 0 & \mathbf{1/4} \end{pmatrix} ,$$

whose trace is $1/4 + 0 + 1/4 = 1/2$ to give the Bayes Vulnerability of Π'_{P_4}.

[17] Of course in an all-zero row it makes no difference which entry is selected.

(D) The connection between strategy matrices and refinement For any strategy matrix G, the transpose $G^{\mathbf{T}}$ has exactly one 1 in each column, and thus can be regarded as a *simple* refinement matrix, one of those which (when pre-multiplied with a partition) merges only whole fractions. If we denote the set of $N\times N$ strate$\mathcal{G}$y matrices by $\mathcal{G}_N$, the set of $N\times N$ $\mathcal{R}$efinement matrices by $\mathcal{R}_N$, and the si$\mathcal{M}$ple subset of these (having only one non-zero entry per row) by $\mathcal{M}_N$, we thus have that

$$\{G{:}\,\mathcal{G}_N \bullet G^{\mathbf{T}}\} \quad = \quad \mathcal{M}_N \quad \subseteq \quad \mathcal{R}_N \ .$$

Furthermore, it can be shown that the complete set of refinement matrices $\mathcal{R}_N$ is in fact the convex closure of its simple subset:

$$\mathcal{R}_N \quad = \quad \mathsf{ccl.}(\mathcal{M}_N) \ . \tag{9}$$

From (8) and by linearity of matrix operations multiplication and trace, we thus have for any $N\times N$-dimensional Π that the Bayes Vulnerability is given by

$$\mathsf{bv.}\Pi \quad = \quad (\sqcup G{:}\,\mathcal{G}_N \bullet \mathsf{tr.}(G^{\mathbf{T}}\times\Pi)) \quad = \quad (\sqcup R{:}\,\mathcal{R}_N \bullet \mathsf{tr.}(R\times\Pi)) \ , \tag{10}$$

because the extra elements in $\mathcal{R}_N$ but not ${\mathcal{G}_N}^{\mathbf{T}}$ are only interpolations, and so cannot increase the maximum of a linear expression. (Recall from above that this maximum is attained for some R.)

Additionally, $\mathcal{R}_N$ forms a monoid under matrix multiplication, that is

$$(\mathcal{R}_N, \times, \mathbf{1}_N) \quad \text{is a monoid,} \tag{11}$$

where $\mathbf{1}_N$ is the $N\times N$ unit of matrix multiplication.[18] We refer to §A for a proof of Properties (9) and (11).

7.2 Soundness

Here from $S\sqsubseteq I$ we must show that $\mathcal{C}(S)\preceq\mathcal{C}(I)$ for all contexts $\mathcal{C}$. From monotonicity (Thm. 2) it suffices to show that $S\sqsubseteq I$ implies $S\preceq I$.

Fix an initial split-state and construct the output hyper-distributions $\Delta'_{\{S,I\}}$ that result from S, I respectively. Then since we assume $S\sqsubseteq I$ we must have $\Delta'_S\sqsubseteq\Delta'_I$. We now show that this implies $\Delta'_S\preceq\Delta'_I$.

Since $S\sqsubseteq I$ trivially guarantees that $\mathsf{ft.}\Delta'_S = \mathsf{ft.}\Delta'_I$ –recall Def. 2– we need to show that the Bayes-Vulnerability condition in the elementary testing order is satisfied. Since $\mathsf{bv.}\Delta = (\sum v{:}\,\mathcal{V} \bullet \mathsf{bv.}(\mathsf{fracs.}\Delta.v))$, it is enough to show that for each $v{:}\,\mathcal{V}$ the vulnerability of $\Pi'_S{:=}\,\mathsf{fracs.}\Delta'_S.v$ is no less than that of $\Pi'_I{:=}\,\mathsf{fracs.}\Delta'_I.v$.

For any such $\Pi'_{\{S,I\}}$ assume *wlog* that they are represented as $N\times N$ matrices. We then have that

[18] Note that it is not a group because only the matrices in $\mathcal{R}_N$ that permute –but do not combine– fractions have inverses.

$$
\begin{array}{lll}
 & \mathsf{bv}.\Pi'_S & \\
= & (\sqcup R{:}\,\mathcal{R}_N \bullet \mathsf{tr}.(R{\times}\Pi'_S))\ . & \text{“from (D), Property (10)”} \\
= & (\sqcup R_1, R_2{:}\,\mathcal{R}_N \bullet \mathsf{tr}.(R_1{\times}R_2{\times}\Pi'_S))\ ) & \text{“from (D), Property (11)”} \\
\geq & (\sqcup R_1{:}\,\mathcal{R}_N \bullet \mathsf{tr}.(R_1{\times}\widehat{R_2}{\times}\Pi'_S))\ ) & \text{“for any } \widehat{R_2}\text{”} \\
= & (\sqcup R_1{:}\,\mathcal{R}_N \bullet \mathsf{tr}.(R_1{\times}\Pi'_I))\ . & \text{“from (B), choose } \widehat{R_2} \text{ so that } \widehat{R_2}{\times}\Pi'_S = \Pi'_I\text{”} \\
= & \mathsf{bv}.\Pi'_I\ . & \text{“from (D), Property (10)”}
\end{array}
$$

That gives us

Theorem 3. *Refinement is sound for Bayes Risk* If $S{\sqsubseteq}I$ then $\mathcal{C}(S){\preceq}\mathcal{C}(I)$ for all contexts $\mathcal{C}$. □

7.3 Completeness

Here from $S{\not\sqsubseteq}I$ we must discover a context $\mathcal{C}$ such that $\mathcal{C}(S){\not\preceq}\mathcal{C}(I)$. (The proof here is self-contained; but as background we give a fully worked example in §D).

Since $S{\not\sqsubseteq}I$, there must be an initial split-state (v,δ) from which S, I yield hyper-distributions $\Delta'_{\{S,I\}}$ with $\Delta'_S{\not\sqsubseteq}\Delta'_I$. We can assume however that $\Delta'_{S,I}$ give equal overall probabilities to *visible* variables since, if they did not, they would be functionally different, giving $S{\not\preceq}I$ immediately. This being so, we can assume that for some final v' we have that partition $\Pi_S{:=}\,\mathsf{fracs}.\Delta'_S.v'$ cannot be transformed into partition $\Pi_I{:=}\,\mathsf{fracs}.\Delta'_I.v'$ via the two steps (i), (ii) in Def. 2. That is, we have $\Pi_S \not\sqsubseteq \Pi_I$.

We will define a distribution D such that the context $(-\,;C)$ where C is

$$\textbf{if } \mathsf{v}{=}v' \textbf{ then } \mathsf{h}{:\in}\,D.\mathsf{h} \textbf{ else } \mathsf{h}{:=}\,0 \textbf{ fi}$$

can be used to differentiate S from I using elementary testing.

We dispose of the simple case first: if $v''{\neq}v'$ then $\mathsf{fracs}.([\![S;C]\!].(v,\delta)).v''$ equals $\mathsf{fracs}.([\![I;C]\!].(v,\delta)).v''$ since, first, hyper-distributions $\Delta'_{\{S,I\}}$ give equal probabilities to that v'' and, second, the final value h' of h is zero for both $S;C$ and $I;C$ in that case. The vulnerability associated with these partitions is therefore the same. To establish $[\![S;C]\!] \not\preceq [\![I;C]\!]$ for our chosen C, it is thus enough to show that the vulnerability of $\mathsf{fracs}.([\![I;C]\!].(v,\delta)).v'$ is strictly greater than for $\mathsf{fracs}.([\![S;C]\!].(v,\delta)).v'$. Treating $\Pi_{\{S,I\}}$ as $N{\times}N$ matrices, we calculate

$$
\begin{array}{lll}
 & \text{“Bayes Vulnerability of } \Pi_S;C\text{”} & \\
= & \text{“Bayes Vulnerability of } \Pi_S{\times}D\text{”} & \text{“(A) above; definition of } C \text{ based on } D\text{”} \\
= & \mathsf{tr}.(\widehat{R}{\times}\Pi_S{\times}D) & \text{“(10) in (D) above; for some maximising } \widehat{R}{\in}\mathcal{R}_N\text{”} \\
= & \mathsf{tr}.(\widehat{\Pi}{\times}D) & \text{“(B) above; for refinement } \widehat{\Pi} = \widehat{R}{\times}\Pi_S \text{ of } \Pi_S\text{”} \\
 & & \\
< & \mathsf{tr}.(\Pi_I{\times}D) & \text{“}D \text{ was chosen in advance, using the } \textit{Separating} \\
 & & \quad \textit{Hyperplane Lemma}\text{, and does not depend on } \widehat{\Pi}\text{: see below”} \\
 & & \\
= & \mathsf{tr}.(1{\times}\Pi_I{\times}D) & \text{“identity”} \\
\leq & (\sqcup R{:}\,\mathcal{R}_N \bullet \mathsf{tr}.(R{\times}\Pi_I{\times}D)) & \text{“}1{\in}\mathcal{R}_N\text{”} \\
= & \text{“Bayes Vulnerability of } \Pi_I{\times}D\text{”} & \text{“(10) in (D) above”}
\end{array}
$$

$=$ "Bayes Vulnerability of $\Pi_I; C$" . "(A) above; definition of C"

The structure of the argument is basically a reformulation on the S side, an appeal to the separation property of the "pre-selected" matrix D, and then a complementary un-reformulation on the I side. Thus for "see below" we argue as follows.

To prepare D we consider all possible refinements of Π_S together. These refinements $\{\!\{R{:}\,\mathcal{R}_N \bullet R{\times}\Pi_S\}\!\}$ comprise a *convex* set of $N{\times}N$ matrices, (where convexity follows from (9) and linearity of matrix multiplication). Since Π_I is not a refinement of Π_S, we know Π_I is not in that set. If we "flatten out" the matrices into vectors of length N^2, say by glueing their rows together, then we have a "point" Π_I in Euclidean space that is strictly outside of that convex set and by the *Separating Hyperplane Lemma* [35] there must be a plane with normal X that strictly separates that whole set of refinements (including $\widehat{\Pi} = \widehat{R}{\times}\Pi_S$) from the single point Π_I. The point X too will be a vector of length N^2 and, written with matrices, the strict-separation condition is then that

$$\mathsf{tr}.(\widehat{\Pi}{\times}X^{\mathbf{T}}) \quad < \quad \mathsf{tr}.(\Pi_I{\times}X^{\mathbf{T}}) \qquad \text{for all } \widehat{\Pi} \text{ refining } \Pi_S$$

since the dot-product of two N^2-vectors A, B written as matrices of size $N{\times}N$ is just $\mathsf{tr}.(A{\times}B^{\mathbf{T}})$. This is precisely what we required above; and so our D is made by taking the direction numbers of the separating hyperplane in Euclidean N^2-space and turning them back into a matrix, and transposing the result.

We admit that there is no guarantee that the D constructed as above will have one-summing rows. However, we can choose D to have all non-negative coefficients because Π_I and all the refinements $\widehat{\Pi}$ of Π_S have the same weight, and thus we can add any constant to all elements of D without affecting its separating property; similarly we can scale it by any positive number. Thus we can assume *wlog* that D is non-negative and that all its rows sum to no more than 1. To then make each row of D sum to one exactly we can extend it with an extra "column zero" whose entries are chosen just for that purpose. We then need to guarantee –as a technical detail– that neither the Bayes Vulnerability strategy matrix for Π_S or Π_I chooses h to be zero. We do that, if necessary, by adding a second context program that acts as **skip** when $\mathsf{h}{\neq}0$; but when $\mathsf{h}{=}0$ it executes a large probabilistic choice over h to distribute the 0 value over enough new values $-1, -2 \ldots$ to make sure none of them individually will have a large enough probability to attract a maximising choice. [19] That gives us

Theorem 4. *Refinement is complete for Bayes Risk* If $S \not\sqsubseteq I$ then $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$ for some context $\mathcal{C}$. □

7.4 Maximal discrimination of the Bayes-Risk elementary order

In this section only, we write "$\preceq_{\mathsf{B}}$" for the Bayes-Risk based elementary testing order ($\preceq$), and we write "$\preceq_1$" etc. to stand generically for any similar order based on one of the four alternative entropy measures set out in §1.2.

[19] At most N new values will be required for such a context.

The problem discussed in §1.2 was that one could have $A \prec_1 B$ and yet $B \prec_2 A$ for programs A, B and competing elementary orders $(\preceq_1)$ and $(\preceq_2)$. Similarly, for any of the four $(\prec_1)$ including $(\prec_{\mathsf{B}})$ itself, it's easy to manufacture examples where we have $A \prec_1 B$ but there is a context $\mathcal{C}$ that reverses the comparison, so that $\mathcal{C}(B) \prec_1 \mathcal{C}(A)$. This seems a hopelessly confused situation.

Luckily it turns out (§G) that refinement $(\sqsubseteq)$ is sound not only for $(\preceq_{\mathsf{B}})$ but for the other three orders as well and –since $(\sqsubseteq)$ is complete for Bayes Risk– that gives us

Theorem 5. *Bayes Risk is maximally discriminating* With context, Bayes Risk is maximally discriminating among the orders of §1.2: that is if $(\preceq_1)$ is an order derived from one of the entropies of §1.2, then whenever for two programs S, I and all contexts $\mathcal{C}$ we have $\mathcal{C}(S) \preceq_{\mathsf{B}} \mathcal{C}(I)$ we also have $\mathcal{C}(S) \preceq_1 \mathcal{C}(I)$ for all $\mathcal{C}$.

Equivalently, if two programs A, B are distinguished by any $(\preceq_1)$ from §1.2, that is $A \not\preceq_1 B$, then there is a context $\mathcal{C}$ such that $(\preceq_{\mathsf{B}})$ in particular distinguishes $\mathcal{C}(A)$ and $\mathcal{C}(B)$, that is such that $\mathcal{C}(A) \not\preceq_{\mathsf{B}} \mathcal{C}(B)$.

Proof: The equivalence of the first and second formulations is straightforward; [20] we prove the second, reasoning

$$
\begin{array}{lll}
 & A \not\preceq_1 B & \\
\Rightarrow & A \not\sqsubseteq B & \text{“soundness of } (\sqsubseteq) \text{ for } (\preceq_1), \text{ see §G”} \\
\Rightarrow & \mathcal{C}(A) \not\preceq_{\mathsf{B}} \mathcal{C}(B)\ . & \text{“completeness of } (\sqsubseteq) \text{ for } (\preceq_{\mathsf{B}}); \text{ some context } \mathcal{C}”
\end{array}
$$

□

It's the completeness result for $(\preceq_{\mathsf{B}})$ that makes it maximal, i.e. that seems to single it out from among the other orders. Whether or not the other orders are also complete is an open problem.

8 Case study: The *Three Judges* protocol

The motivation for our case study is to suggest and illustrate techniques for reasoning compositionally from specification to implementation of noninterference [27, 23, 11]. Our previous examples include (unboundedly many) Dining Cryptographers [6], Oblivious Transfer [32] and Multi-Party Shared Computation [39]. All of them however used our *qualitative* model for compositional noninterference [27, 23]; here of course we are using instead a *quantitative* model.

[20] First implies second:

> If $A \not\preceq_1 B$ then, appealing to the identity context in the conclusion of the first formulation, for some $\mathcal{C}$ we have $\mathcal{C}(A) \not\preceq_{\mathsf{B}} \mathcal{C}(B)$.

Second implies first:

> Assume $\mathcal{C}(S) \not\preceq_1 \mathcal{C}(I)$ for some $\mathcal{C}$, whence immediately from the second formulation we have $\mathcal{D}(\mathcal{C}(S)) \not\preceq_{\mathsf{B}} \mathcal{D}(\mathcal{C}(I))$ for some context $\mathcal{D}(\mathcal{C}(\cdot))$.

The example is as follows. Three judges A, B, C are to give a majority decision, innocent or guilty, by exchanging messages but concealing their individual votes a, b and c, respectively. [21]

We describe this protocol with a program fragment, a specification which captures exactly the functional and security properties we want. Its variables are Boolean, equivalently $\{0, 1\}$ and, including some notational conventions explained below, it evaluates $(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)$ atomically, and reveals the value of the expression to everyone:

$$\begin{array}{rll} \mathbf{vis}_A\ \mathsf{a};\ \mathbf{vis}_B\ \mathsf{b};\ \mathbf{vis}_C\ \mathsf{c}; & \leftarrow \text{These are global variables.} & \\ \mathbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)\ . & \leftarrow \text{Atomically evaluate} & (12) \\ & \quad \text{and reveal expression.} & \end{array}$$

Note that this specification is *not* noninterference-secure in the usual sense: for example when a judges "not guilty" (false) and yet the defendant is found guilty by majority, Agent A learns that both b, c must have judged "guilty " — and that is a release of information. This allows a similar behaviour in the implementation, strictly speaking a declassification: but we need no special measures to deal with it.

We interpret the specification as follows. The system comprises four agents: the judges A, B, C and (say) some Agent X as an external observer. The participating agents (A, B, C) are distributed, each with its own state-space; and the external observer has no state. The annotations $\mathbf{vis}_{\{A,B,C\}}$ above indicate that the variables $\mathsf{a}, \mathsf{b}, \mathsf{c}$ are located with the agents A, B, C respectively and are visible only to them: that is, only Agent A can see variable a etc. [22]

The **reveal** command (explained in more detail below) publishes its argument for all agents to see.

The *location* of a variable has no direct impact on semantics (in our treatment here); but it does affect our judgement of what is directly executable and what is not. In particular, an expression is said to be *localised* just when all its variables are located at the same agent, and only localised expressions can be directly executed (by that agent, thus). Thus $\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2$ is not localised, in spite of its being meaningful in the sense of having a well defined value; and it is precisely because it is not localised that we must develop the specification further. Assignment statements $\mathsf{a}{:=}E$, where a is in Agent A, say, and E is localised in Agent B, are implemented by B's calculating E and then sending its value in a message to A.

The *visibility* of a variable does affect semantics. A variable annotated $\mathbf{vis}_A$, for example, is treated as if it were simply annotated **vis** when we are reasoning

[21] Though this is similar to the (generalised) Dining Cryptographers, it is more difficult: we do not reveal anonymously the total number of guilty votes; rather we reveal only whether that total is a majority [11, `Morgan:09a`].

[22] In principle we could have separate annotations for visibility and for location, allowing thus variables located at A that however A cannot see, and (complementarily) variables located at B that A can see. But in this example we do not need that fine control, and so we use **vis** for both.

from Agent A's point of view; from Agents' B, C points of view, it is treated as if it were annotated **hid**; and the same applies analogously to the other agents. Thus in the example we will treat three agents A, B, C each with her own view: variables visible to one (declared **vis**) will be hidden from another (declared **hid**) — and vice versa. The "extra" Agent X (mentioned above) sees none of $\mathsf{a}, \mathsf{b}, \mathsf{c}$, but does observe the **reveal**. This simple approach is possible for us because we are not dealing with agents whose actions can be influenced by other agents' knowledge.

In principle the **vis**-subscripting convention means that protocol development, e.g. as in §8.3ff. to come, will require a separate proof for each observer (since the patterns of variables' visibility might differ); but in practice we can usually find a single chain of reasoning each of whose steps is valid for two or even all three observers at once.

Before incrementally developing (12) into an implementation in order to localise its expressions, we introduce some further extensions, including the **reveal** statement mentioned above [22], that will be used in the subsequent program derivation.

8.1 Further program-language extensions

Multiple- and local variables To this point we have had just two variables, visible v and hidden h, and a split-state $\mathcal{V}\times\mathsf{D}\mathcal{H}$ to describe their behaviour. In practice each of $\mathcal{V}, \mathcal{H}$ will each comprise many variables, represented in the usual Cartesian way. Thus if we have variables $\mathsf{a}{:}\,\mathcal{A}, \mathsf{b}{:}\,\mathcal{B}, \mathsf{c}{:}\,\mathcal{C}, \mathsf{d}{:}\,\mathcal{D}$ with the first two a, b visible and the last two c, d hidden, then $\mathcal{V}$ is $\mathcal{A}\times\mathcal{B}$ and $\mathcal{H}$ is $\mathcal{C}\times\mathcal{D}$ so that the state-space is $\mathcal{A}\times\mathcal{B}\times\mathsf{D}(\mathcal{C}\times\mathcal{D})$. Assignments and projections are handled as normal.

We allow local variables, both visible and hidden, which are treated (also) as normal: within the scope of a visible local-variable declaration $|||[\ \mathbf{vis}\ \mathsf{x}{:}\,\mathcal{X}\cdots\ ]|||$, the $\mathcal{V}_{\text{local}}$ used is $\mathcal{X}\times\mathcal{V}_{\text{global}}$. Hidden variables are treated similarly.[23]

Revelations Command **reveal** E publishes expression E for all to see: it is equivalent to the local block

$$|||[\ \mathbf{vis}\ \mathsf{v};\ \mathsf{v}{:=}\,E\ ]|||\ , \tag{13}$$

but it avoids the small extra complexity of declaring the temporary visible-to-all variable v and the having to introduce the scope brackets as (13) does. The attraction of this is that the **reveal** command has a simple algebra of its own, including for example that **reveal** E = **reveal** F just when E and F are interdeducible given the values of (other) visible variables [22, 24]. Thus for example (and slightly more generally) we have

$$\mathbf{reveal}\ \mathsf{a}{\nabla}\mathsf{b};\ \mathbf{reveal}\ \mathsf{b}{\nabla}\mathsf{c}\quad=\quad\mathbf{reveal}\ \mathsf{b}{\nabla}\mathsf{c};\ \mathbf{reveal}\ \mathsf{c}{\nabla}\mathsf{a}\ ,$$

[23] Implicitly local variables are assumed to be initialised by a uniform choice over their finite state space. In our examples however, we always initialize local variables explicitly, to avoid confusion.

using (∇) to denote exclusive-or, because from $\mathsf{a}\nabla\mathsf{b}$ and $\mathsf{b}\nabla\mathsf{c}$ an observer can deduce both $\mathsf{b}\nabla\mathsf{c}$ and $\mathsf{c}\nabla\mathsf{a}$, and vice versa.

Bulk atomicity In Fig. 1 we introduced the semantics of commands and remarked that for syntactically atomic commands the secure semantics is given by Def. 1, based on the classical semantics of the same command. With atomicity brackets $\langle\!\langle\cdot\rangle\!\rangle$ we make *groups* of commands atomic "by fiat," so that Def. 1 applies to them as well. We have

Definition 3. *Secure semantics atomicity brackets* Given any program P we define

$$[\![\langle\!\langle P\rangle\!\rangle]\!].(v,\delta) \quad := \quad \mathsf{embed}.(\odot\, h{:}\,\delta \;\bullet\; [\![P]\!]_C.(v,h)) \;. \tag{14}$$

The effect overall, as earlier, is to impose the largest possible ignorance of h' that is consistent with seeing v' and knowing the classical semantics $[\![P]\!]_C$ of the program between brackets. In particular, perfect recall and implicit flow are both suppressed by $\langle\!\langle\cdot\rangle\!\rangle$. □

A comparison of Defs. 1 and 3 shows immediately that for any syntactically atomic command A we have $A = \langle\!\langle A\rangle\!\rangle$, just as one would expect. [24] With groups of commands of course the equality does not hold in general: for example we cannot reason

$$\begin{array}{lll}
 & \mathsf{v}{:=}\,\mathsf{h};\ \mathsf{v}{:=}\,0 & \\
= & \langle\!\langle \mathsf{v}{:=}\,\mathsf{h}\rangle\!\rangle;\ \langle\!\langle \mathsf{v}{:=}\,0\rangle\!\rangle & \text{“syntactically atomic, both”} \\
?= & \langle\!\langle \mathsf{v}{:=}\,\mathsf{h};\ \mathsf{v}{:=}\,0\rangle\!\rangle & \text{“\textit{invalid step}”} \\
= & \langle\!\langle \mathsf{v}{:=}\,0\rangle\!\rangle & \text{“classical equality”} \\
= & \mathsf{v}{:=}\,0\ , & \text{“syntactically atomic”}
\end{array}$$

because –as we have often stressed– an assignment of h to v *does* reveal h to an observer, even of v is immediately overwritten. The invalid step violates the conditions of Lem. 1 immediately below, which gives an important special situation in which we do have distribution of atomicity inwards:

Lemma 1. *Distribution of atomicity* Given is a sequential composition of two programs $P;Q$. If by observation of the visible variable v before the execution of P and after the execution Q it is always possible to determine the value v had *between* P and Q, then we do have

$$\langle\!\langle P;Q\rangle\!\rangle \quad = \quad \langle\!\langle P\rangle\!\rangle;\langle\!\langle Q\rangle\!\rangle \;.$$

Proof: (sketch) The full proof is given in §E.

It can be shown that the left- and right-hand sides' classical effect on v and h are the same, and so the only possible difference between the two can be the

[24] Note that although **reveal** E looks syntactically atomic, it is via (13) actually an abbreviation of a compound command: thus in fact $\langle\!\langle$**reveal** $E\rangle\!\rangle \neq$ **reveal** E in general. Actually $\langle\!\langle$**reveal** $E\rangle\!\rangle =$ **skip** in all cases, whereas **reveal** $E =$ **skip** only when E is visible.

degree to which h is hidden. On the left, variable h must be maximally hidden since that is the (defined) effect of the atomicity brackets $\langle\!\langle \cdot \rangle\!\rangle$. Thus we need only argue that h is maximally hidden on the right as well.

Since h is maximally hidden after $\langle\!\langle P \rangle\!\rangle$, the only way h can fail to be maximally hidden after the subsequent $\langle\!\langle Q \rangle\!\rangle$ is if there are two (or more) distinct values of v after P, say $\hat{v}_{\{0,1\}}$ each with its associated hidden distribution $\delta_{\{0,1\}}$ of h, that are brought together to the *same* final value v' by execution of Q. For that would mean that after Q we could have two distinct distributions $\delta'_{\{0,1\}}$ of h associated with that single v', which is precisely what it means not to be maximally hidden. Each $\delta'_{\{0,1\}}$ would have been derived from the corresponding $\delta_{\{0,1\}}$ in between.

That scenario cannot occur if for any particular starting v before P that leads to two (or more) values $\hat{v}_{\{0,1\}}$ between P and Q, we never have Q bringing those values back together again to a single final value v'. That amounts to being able to determine the intermediate value $\hat{v}$ of v from its values before (v) and after (v'). □

In fact our invalid step $\langle\!\langle \mathsf{v}{:=}\,\mathsf{h}; \mathsf{v}{:=}\,0 \rangle\!\rangle \neq \langle\!\langle \mathsf{v}{:=}\,\mathsf{h} \rangle\!\rangle; \langle\!\langle \mathsf{v}{:=}\,0 \rangle\!\rangle$ above shows off the condition exactly. Although v's intermediate value is indeed determined by the initial h, that is not good enough because we cannot see that h: we have access only to the initial v. And v's final value is always 0, again hiding v's intermediate value from us. Knowing v before and after, in this example, does not tell us its intermediate value (which is in fact h).

By definition, semantic equivalence of P and Q in the classical model entails semantic equivalence of $\langle\!\langle P \rangle\!\rangle$ and $\langle\!\langle Q \rangle\!\rangle$ — that is why *within* atomicity brackets we can use classical equality reasoning.

8.2 Subprotocols: qualitative vs. quantitative reasoning

Rather than appeal constantly to the basic semantics (Fig. 1) instead we have accumulated, with experience, a repertoire of identities –a program algebra– which we use to reason at the source level. Those identities themselves are proved directly in the semantics but, after that, they become permanent members of the designer's toolkit. One of the most common is the *Encryption Lemma*.

The Encryption Lemma Let statement $(\mathsf{v}\nabla\mathsf{h}){:=}\,E$ set Booleans v, h so that their exclusive-or $\mathsf{v}\nabla\mathsf{h}$ equals Boolean E: there are exactly two possible ways of doing so. In our earlier work [27], we proved that when the choice is made demonically, on a single run nothing is revealed about E; in our refinement style we express that as

$$\textbf{skip} \quad = \quad |\!|\![\ \textbf{vis}\ \mathsf{v}; \textbf{hid}\ \mathsf{h};\ (\mathsf{v}\nabla\mathsf{h}){:=}\,E\ ]\!|\!| \ . \tag{15}$$

In our current model we can prove that *exactly the same identity holds* provided the choice of possible values for v and h is made uniformly:

Lemma 2. *The Encryption Lemma* For any Boolean expression E we have that the following block is equal to **skip**, and so reveals nothing:

$$|||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ (\mathsf{v}\nabla\mathsf{h}){:=}\,E\ ]|| \ .$$

For this we require that the implicit choice in $(\mathsf{v}\nabla\mathsf{h}){:=}\,E$ is made uniformly.

Proof: We calculate

$$\begin{array}{lll}
& |||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ (\mathsf{v}\nabla\mathsf{h}){:=}\,E\ ]|| & \\
= & |||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ \langle\!\langle (\mathsf{v}\nabla\mathsf{h}){:=}\,E \rangle\!\rangle\ ]|| & \text{“syntactically atomic”} \\
= & |||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ \langle\!\langle \mathsf{v}{:=}\,\mathsf{true}\oplus\mathsf{false}; \mathsf{h}{:=}\,\mathsf{v}\nabla E \rangle\!\rangle\ ]|| & \text{“classical equality (†)”} \\
= & |||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ \langle\!\langle \mathsf{v}{:=}\,\mathsf{true}\oplus\mathsf{false} \rangle\!\rangle; \langle\!\langle \mathsf{h}{:=}\,\mathsf{v}\nabla E \rangle\!\rangle\ ]|| & \text{“Lem. 1”} \\
= & |||[\ \mathbf{vis}\ \mathsf{v}; \mathbf{hid}\ \mathsf{h};\ \mathsf{v}{:=}\,\mathsf{true}\oplus\mathsf{false}; \mathsf{h}{:=}\,\mathsf{v}\nabla E\ ]|| & \text{“syntactically atomic”} \\
= & |||[\ \mathbf{vis}\ \mathsf{v};\ \mathsf{v}{:=}\,\mathsf{true}\oplus\mathsf{false};\ |||[\ \mathbf{hid}\ \mathsf{h};\ \mathsf{h}{:=}\,\mathsf{v}\nabla E\ ]||\ ]|| & \text{“}\mathbf{hid}\ \mathsf{h}\text{ does not capture”} \\
= & |||[\ \mathbf{vis}\ \mathsf{v};\ \mathsf{v}{:=}\,\mathsf{true}\oplus\mathsf{false}\ ]|| & \text{“assignment to local hidden is }\mathbf{skip}\text{”} \\
= & \mathbf{skip}\ . & \text{“value assigned to local }\mathsf{v}\text{ is known already (‡)”}
\end{array}$$

□

The crucial step in the proof above was the classical equality at (†), and we note that other variations are possible: for example we also have the classical equality

$$(\mathsf{v}\nabla\mathsf{h}){:=}\,E \qquad = \qquad \mathsf{h}{:=}\,\mathsf{true}\oplus\mathsf{false};\ \mathsf{v}{:=}\,E\nabla\mathsf{h}\ , \tag{16}$$

which suggests the operational procedure of “flipping a private coin h” and then revealing (via assignment to local v) the exclusive-or of that private coin with some expression E. The above reasoning shows that also to be equal to **skip**.

Finally, we recall that $(\oplus)$ means choose *uniformly*, and we now show that it is essential for the (†) step and for the equality (16): if for example we had $\mathsf{h}{:=}\,\mathsf{true}\ {}_p\!\oplus\ \mathsf{false};\ \mathsf{v}{:=}\,E\nabla\mathsf{h}$ in (16) on the right-hand side, but with $p{\neq}1/2$, it would not be possible to rewrite that in the form $\mathsf{v}{:=}\,\mathsf{true}\ {}_{1/2}\!\oplus\ \mathsf{false}; \mathsf{h}{:=}\,\mathsf{v}\nabla E$ as we had at (†) but with the $1/2$ here exposed. Instead we'd have

$$\mathsf{v}{:=}\,\neg E\,{}_p\!\oplus E; \quad \mathsf{h}{:=}\,\mathsf{v}\nabla E$$

and the last step (‡) would then be invalid if E contained hidden variables (as it usually would). The role of $p{=}1/2$ is thus that the equality

$$\mathsf{v}{:=}\,\neg E\ {}_{1/2}\!\oplus\ E \qquad = \qquad \mathsf{v}{:=}\,\mathsf{true}\ {}_{1/2}\!\oplus\ \mathsf{false}$$

holds no matter what expression E is, and in particular even if it contains hidden variables — but only (in general) when the choice is with probability $1/2$.

Lem. 2 means that extant qualitative source-level proofs that rely only on “upgradeable identities” like (15) can be used *as is* for quantitative results provided the demonic choices involved are converted to uniform choice. And that is the case with our current example.

Beyond the Encryption Lemma, we use *Two-party Conjunction* [39] and *Oblivious Transfer* [32] in our implementation. Just as for the Encryption Lemma, the algebraic proofs of their implementations [27, 23] apply quantitatively provided we interpret the (formerly) demonic choice as uniform. We now look briefly at those subprotocols.

Two-Party Conjunction In the Two-Party Conjunction subprotocol, the conjunction of two privately held Booleans is published without revealing either Boolean separately. It is an instance of Yao's *Multi-party Computation* technique [39] and we have given a formal derivation of it elsewhere [23]. Its specification is

Two-Party Conjunction

$$\begin{array}{l} \mathbf{vis}_B\ \mathsf{b}; \mathbf{vis}_C\ \mathsf{c}; \qquad \leftarrow \text{These are global variables.} \\ \quad \mathbf{reveal}\ \mathsf{b} \wedge \mathsf{c}\ , \end{array} \tag{17}$$

and its similarity to (12) is clear: a compound outcome $\mathsf{b}\wedge\mathsf{c}$ is published without revealing the components b, c — except that, just as before, if for example the revealed outcome is false but b is true, then B can deduce that c must have been false (and similar).

We develop an implementation of (17) in several steps, as follows. Note that for some steps the *justification* varies depending on the agent although we have arranged that the claimed equality is valid for all of them. We have

$$\begin{array}{lll}
(17) = & \mathbf{skip}; & \text{“identity”} \\
& \mathbf{reveal}\ \mathsf{b} \wedge \mathsf{c} & \\
\\
= & |\!|[\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0]|\!|; & \text{“Encryption Lemma for } A, C; \\
& \mathbf{reveal}\ \mathsf{b} \wedge \mathsf{c} & \text{obvious for } B; \text{ see below } (\ddagger).\text{”} \\
\\
= & |\!|[\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; & \text{“Revelation algebra: in this context } \mathsf{b} \wedge \mathsf{c} \equiv \mathsf{b}_0 \nabla \mathsf{b}_\mathsf{c} \quad \dagger \\
& \quad (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0; & \text{where } \mathsf{b}_\mathsf{c} := (\mathsf{b}_1\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{b}_0).\text{”} \\
& \quad \mathbf{reveal}\ \mathsf{b}_\mathsf{c} & \\
&]|\!| & \\
\\
= & |\!|[\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; & \text{“Delegate second revelation to Agent } C.\text{”} \\
& \quad (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0; & \\
& \quad |\!|[\ \mathbf{vis}_C\ \mathsf{c}_0;\ \mathsf{c}_0 := \mathsf{b}_\mathsf{c};\ \mathbf{reveal}\ \mathsf{c}_0\ ]|\!| & \\
&]|\!| & \\
\\
= & |\!|[\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0; & \text{“Rearrange declarations; clean up.”} \\
& \quad (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0; & \leftarrow \text{This done by Agent } B. \\
& \quad \mathsf{c}_0 := \mathsf{b}_\mathsf{c}; & \leftarrow \text{An “Oblivious Transfer” between } B, C. \\
& \quad \mathbf{reveal}\ \mathsf{c}_0 & \leftarrow \text{This done by Agent } C. \\
&]|\!|\ . &
\end{array}$$

At $(\ddagger)$ we find a case where the same equality applies to all agents, although in fact the reasons for its validity use agent-specific reasoning. For example, for Agents A, C the fragment is effectively

$$|\!|[\ \mathbf{hid}\ \mathsf{b}_0, \mathsf{b}_1;\ (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0]|\!|\ ,$$

which is a version of the Encryption Lemma in which b_0, being revealed, takes the role of the local visible variable. Variable b_1 is the local hidden, and (hidden)

variable b is the expression E (on which there are no restrictions). On the other hand, for Agent B the fragment is

$$|||[\ \mathbf{vis}\ \mathsf{b}_0, \mathsf{b}_1;\ (\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b};\ \mathbf{reveal}\ \mathsf{b}_0]|||$$

with b a global visible: this is trivially equal to **skip** because all variables are visible.

At (†) we use the revelation algebra mentioned in §8.1 to reason that once b_0 is revealed, going on to reveal $\mathsf{b} \wedge \mathsf{c}$ is equivalent to revealing just b_c since –knowing b_0– we can calculate each of $\mathsf{b} \wedge \mathsf{c}$ and b_c from the other. [25]

More interesting than any of that, however, is that in the last step we appeal to a further subprotocol by including the *specification* of the "Oblivious Transfer Protocol" [31, 32]. Here Agent C has a private $\{0,1\}$-valued variable c and obtains from Agent B either b_0 or b_1, depending on c. But Agent B does not discover what c is, and Agent C does not discover $\mathsf{b}_{\neg\mathsf{c}}$. We give a rigorous implementation of the protocol elsewhere [27]; an informal explanation may be found in §F.

Finally, to emphasise our earlier point about declassification, we suppose b is true but c is false and thus that B learns c by noting that false is revealed overall; note that this is a property of the *specification*. Now, in the implementation, we can see how this happens: when b is true the local variables $\mathsf{b}_{0,1}$ will be complementary and so –in spite of not learning c while the Oblivious Transfer is carried out– Agent B will still learn c afterwards by comparing c_0 with her own $\mathsf{b}_{0,1}$.

8.3 The Three-Judges implementation: first attempt

We begin with an implementation attempt that fails, because this will illustrate two things. The first is that our model prevents incorrect developments, that is it stops us from constructing implementations less secure than their specifications: arguably this "negative" aspect of a method is its most important property, since it would be trivial to describe a method that allowed secure refinements... and all others as well. The key is what is *not* allowed.

The second thing illustrated here is that a conditional **if** E $\cdots$ **fi** should be considered to reveal its condition E implicitly. This *implicit flow* is a property forced upon us by our advocacy of program algebra and our use of compositionality: since the **then**- and the **else** branch of a conditional can be developed differently *after* the conditional has been introduced, we must expect that those differences might reveal to an attacker which branch is being executed (and hence the condition implicitly). This is exactly what we are about to see.

[25] We have

$$\mathsf{b}_0 \nabla \mathsf{b}_\mathsf{c} \ \equiv\ (\mathsf{b}_0 \nabla \mathsf{b}_1\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{b}_0 \nabla \mathsf{b}_0) \ \equiv\ (\mathsf{b}\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{false}) \ \equiv\ \mathsf{b} \wedge \mathsf{c}\ .$$

We start with some Boolean algebra

$$(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \quad \equiv \quad \mathsf{a} \wedge (\mathsf{b} \vee \mathsf{c}) \;\vee\; \mathsf{b} \wedge \mathsf{c} \quad \equiv \quad (\mathsf{b} \vee \mathsf{c} \;\textbf{if}\; \mathsf{a} \;\textbf{else}\; \mathsf{b} \wedge \mathsf{c}) \;,$$

and that suggests the first development steps

$$\begin{array}{ll}
 & \textbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \\
\stackrel{?}{=} & \qquad\qquad\qquad\qquad\qquad\qquad \text{“}P \stackrel{?}{=} \textbf{if}\ E\ \textbf{then}\ P\ \textbf{else}\ P\ \textbf{fi}\text{”} \\
 & \textbf{if}\ \mathsf{a}\ \textbf{then reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \\
 & \qquad \textbf{else reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \\
 & \textbf{fi} \\
= & \textbf{if}\ \mathsf{a} \qquad\qquad\qquad\qquad \text{“After } \textbf{then} \text{ we can assume } a\text{;} \\
 & \quad \textbf{then reveal}\ \mathsf{b} \vee \mathsf{c} \qquad\quad \text{after } \textbf{else} \text{ we can assume } \neg a\text{.”} \\
 & \quad \textbf{else}\ \ \textbf{reveal}\ \mathsf{b} \wedge \mathsf{c} \\
 & \textbf{fi} \ldots
\end{array} \tag{18}$$

Now we can deal immediately with the **else**-part by adapting the Two-Party Conjunction Protocol of §8.2 so that it reveals $\mathsf{b}{\wedge}\mathsf{c}$ only to Agent A; we introduce a pair of A-private local variables for that purpose. The result is

$$\begin{array}{ll}
\ldots = & |\!|\![\ \textbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \textbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \textbf{vis}_C\ \mathsf{c}_0; \qquad\qquad \text{“Adapting §8.2”} \\
 & \quad \textbf{if}\ \mathsf{a} \\
 & \quad\ \ \textbf{then reveal}\ \mathsf{b} \vee \mathsf{c} \\
 & \quad\ \ \textbf{else}\ (\mathsf{b}_0 \nabla \mathsf{b}_1) {:=} \mathsf{b}; \qquad \leftarrow \text{Done privately by Agent } B\text{.} \\
 & \qquad\quad \mathsf{a}_B {:=} \mathsf{b}_0; \qquad \leftarrow \text{Message } B{\to}A\text{.} \\
 & \qquad\quad \mathsf{c}_0 {:=} \mathsf{b}_\mathsf{c}; \qquad \leftarrow \text{Oblivious Transfer } B{\to}C\text{.} \\
 & \qquad\quad \mathsf{a}_C {:=} \mathsf{c}_0; \qquad \leftarrow \text{Message } C{\to}A\text{.} \\
 & \qquad\quad \textbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C \quad \leftarrow \text{Agent } A \text{ announces majority verdict.} \\
 & \quad \textbf{fi} \\
 &]\!|\!| \ldots
\end{array}$$

For the **then**-part we write $\mathsf{b} \vee \mathsf{c}$ as $\neg(\neg\mathsf{b} \wedge \neg\mathsf{c})$ and adapt the **else**-part accordingly; the effect overall turns out to be replacing the initial b by $\neg\mathsf{b}$ and changing the following assignment. Once we factor out the common portion of the conditional, we have

$$\begin{array}{ll}
\ldots = & |\!|\![\ \textbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \textbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \textbf{vis}_C\ \mathsf{c}_0; \qquad\qquad \text{“Using } \textit{de Morgan}\text{”} \\
 & \quad \textbf{if}\ \mathsf{a}\ \textbf{then}\ (\mathsf{b}_0 \nabla \mathsf{b}_1) {:=} \neg\mathsf{b}; \mathsf{a}_B {:=} \mathsf{b}_1\ \textbf{else}\ (\mathsf{b}_0 \nabla \mathsf{b}_1) {:=} \mathsf{b}; \mathsf{a}_B {:=} \mathsf{b}_0\ \textbf{fi}; \\
 & \quad \mathsf{c}_0 {:=} \mathsf{b}_\mathsf{c};\ \mathsf{a}_C {:=} \mathsf{c}_0; \\
 & \quad \textbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C \\
 &]\!|\!| \ .
\end{array}$$

Now we see that the problem with going further is that Agent A must somehow arrange that B carries out either $(\mathsf{b}_0 \nabla \mathsf{b}_1) := \neg \mathsf{b}$ or $(\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b}$, with that arrangement depending on the value of A's private variable a. Since B's two potential computations are different, there is no way this can occur without B's learning the value of a in the process: this code is already incorrect.

Thus we must abandon this attempt, and admit that the questionable step at (18) above was indeed wrong. In order to allow us to develop distributed implementations, we make the (reasonable) assumption that each agent knows the code it is instructed to execute, with those instructions coming possibly from another agent. In this case Agent B must execute either $(\mathsf{b}_0 \nabla \mathsf{b}_1) := \neg \mathsf{b}; \mathsf{a}_B := \mathsf{b}_1$ or $(\mathsf{b}_0 \nabla \mathsf{b}_1) := \mathsf{b}; \mathsf{a}_B := \mathsf{b}_0$, depending on the value of a which is supposed to be visible only to Agent A.

Our semantics recognises implicit flow, and does not allow in general the transformation of P into **if** E **then** P **else** P **fi**, for exactly this reason. [26] Similarly, a fragment $\mathsf{a} := \mathsf{b}; \mathsf{a} := \mathsf{c}$ represents two messages, one $B \to A$ and then a second one $C \to A$; with perfect recall we recognise that A can learn b by examining a after the first message has arrived, but before the second.

8.4 The Three-Judges implementation: second attempt (sketch)

An "obvious" remedy for §8.3's problem, that Agent B's is aware of which procedure she must follow, is to make B follow *both* procedures, speculatively: she does not know which one A will actually use.

The difficulty is now with Agent A, who learns both $\mathsf{b} \wedge \mathsf{c}$ and $\mathsf{b} \vee \mathsf{c}$. Although those two values do not (always) determine b and c themselves, they do provide strictly more information to A than her knowing a and $(\mathsf{a}+\mathsf{b}+\mathsf{c} \geq 2)$ would have provided on their own. [27] Thus this approach fails also.

Our attention is therefore drawn to arranging for B (and C) to do both two-party calculations, but then for A to get the results of only one of them. That leads naturally to the approach of the next section, a combination of two two-party computations (letting Agents B, C do both calculations) and two (more) oblivious transfers (letting Agent A learning about only one of them.) [28]

8.5 The Three-Judges implementation: successful development

To repair the problem we encountered above we must arrange that Agents B, C as far as possible carry out procedures independent of A's variable a, in particular

[26] In our related work for noninterference with demonic choice and without probability [26, 27], we give further arguments for this point of view, but based directly on program algebra. The extra feature there is that even *classical* programs have a non-trivial refinement relation; here, we have proper refinement only for secure programs.

[27] If a and $(\mathsf{a}+\mathsf{b}+\mathsf{c} \geq 2)$ are both false, then Agent A concludes $\neg(\mathsf{b} \wedge \mathsf{c})$, for which there are the three possibilities $\mathsf{false}/\mathsf{false}, \mathsf{true}/\mathsf{false}, \mathsf{false}/\mathsf{true}$. Agent A's additionally knowing $\mathsf{b} \vee \mathsf{c}$ would eliminate at least one of those three.

[28] The "more" refers to the fact that the two-party computations have oblivious transfers inside of them.

so that calculations relating to $\mathsf{b} \vee \mathsf{c}$ and to $\mathsf{b} \wedge \mathsf{c}$ both occur, irrespective of which result A actually needs.

To achieve this we need a slightly more general form of two-party computation. We begin by introducing the specification of such a two-party conjunction, with its variables made local so that the introduced code is equivalent to **skip**:

$$
\begin{array}{lll}
 & \textbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) & \\
 & & \\
= & [\![\ \textbf{vis}_B\ \mathsf{b}_0; \textbf{vis}_C\ \mathsf{c}_0;\ (\mathsf{b}_0 \nabla \mathsf{c}_0) {:=} \mathsf{b} \wedge \mathsf{c};\ ]\!]; & \text{“Two-party conjunction”} \\
 & \textbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) &
\end{array}
$$

From Agent A's point of view, the introduced statement is trivially equivalent to **skip**: all assignments are to local variables that A cannot see. From Agent B, C's points of view, it is equivalent to **skip** because it is an instance of the Encryption Lemma: each of those two agents can see only one of the two variables assigned-to, and so learns nothing about the expression $\mathsf{b} \wedge \mathsf{c}$. [29]

The statement $(\mathsf{b}_0 \nabla \mathsf{c}_0) {:=} \mathsf{b} \wedge \mathsf{c}$ we have introduced is a more general form of two-party conjunction than the **reveal** $\mathsf{b} \wedge \mathsf{c}$ we illustrated earlier in §8.2 — that is because the conjunction is not actually revealed, not yet; instead it is split into two “shares,” one belonging to each party B, C. Since each party has only one share, the conjunction is not revealed at all at this stage. But those shares can be used as inputs to further two-party computations, while preserving the security, and the contribution of the conjunction to a larger computation is revealed at a later point.

The extra generality introduced by the shares does not cause us extra work here, since we are using only the specification for our reasoning and that (we will see) suffices. When we come to *implement* the general two-party conjunction in more primitive terms, however, we would then have further work to do. We have given such an implementation elsewhere [23].

With exactly the same reasoning as above we can introduce two-party disjunction and, with both conjunction and disjunction present, perform some reorganisation:

---

[29] The following bogus counter-argument is an example of what having a careful definition of equality and refinement helps us to avoid.

“Agent B might know that b is false, and then perhaps receive false also in $\mathsf{b}_\wedge$. She concludes that $\mathsf{c}_\wedge$ is also false, which is a leak since $\mathsf{c}_\wedge$ is supposed to be private to C, invisible to B.”

In fact this is not a leak, because to judge it so we must refer to the *specification* of this fragment. But that is simply **skip** and there is no $\mathsf{c}_\wedge$ declared there: the revealed variable is local to the implementation only.

That is, publishing the value of a hidden variable declared only in the implementation might *look* like a leak in the conventional interpretation –consider $[\![\ \textbf{vis}_C\ \mathsf{c};\ \cdots ; \textbf{reveal}\ \mathsf{c}\ ]\!]$ for example– but it is actually a leak only if that variable c has come to contain information (via assignments say in the “$\cdots$” portion) about other, more global hidden variables that were present in the specification, originally. Our semantics checks for that automatically.

$$
\begin{array}{lll}
\ldots = & |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0; \mathbf{vis}_C\ \mathsf{c}_0;\ (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c};\ ]\!|\!| ; & \text{“Two-party disjunction”} \\
& |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_1;\ (\mathsf{b}_1 \nabla \mathsf{c}_1) := \mathsf{b} \vee \mathsf{c};\ ]\!|\!| ; & \\
& \mathbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) & \\
= & |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; & \text{“Reorganise declarations and scoping”} \\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c}; & \\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1) := \mathsf{b} \vee \mathsf{c}; & \\
& \quad \mathbf{reveal}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) & \\
&]\!|\!| & \\
= & |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; & \text{“Boolean algebra”} \\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c}; & \\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1) := \mathsf{b} \vee \mathsf{c}; & \\
& \quad \mathbf{reveal}\ \mathsf{b}_a \nabla \mathsf{c}_a & \\
&]\!|\!| \ldots &
\end{array}
$$

Now since $\mathsf{b}_a \nabla \mathsf{c}_a$ is revealed to everyone, and thus to A in particular, it does no harm first to capture that value in variables of A, and then to have Agent A reveal those instead:

$$
\begin{array}{lll}
\ldots = & |\!|\![\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C; & \text{“Introduce local private variables of } A\text{”} \\
& \quad \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; & \\
& \quad \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; & \\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c}; & \\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1) := \mathsf{b} \vee \mathsf{c}; & \\
& \quad (\mathsf{a}_B \nabla \mathsf{a}_C) := \mathsf{b}_a \nabla \mathsf{c}_a; & \\
& \quad \mathbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C & \\
&]\!|\!| \ldots &
\end{array}
$$

The point of using two variables $\mathsf{a}_{\{B,C\}}$ rather than one is to be able to split the transmission of information $B, C{\rightarrow}A$ into two separate oblivious transfers $B{\rightarrow}A$ and $C{\rightarrow}A$.

Thus the protocol boils down to three two-party computations: a conjunction $\mathsf{b}{\wedge}\mathsf{c}$, a disjunction $\mathsf{b}{\vee}\mathsf{c}$ and an exclusive-or $\mathsf{b}_a \nabla \mathsf{c}_a$. The *rhs* of the last is actually within an atomic conditional on a, that is $(\mathsf{b}_0 \nabla \mathsf{c}_0)\,\mathbf{if}\,\mathsf{a}{=}0\,\mathbf{else}\,(\mathsf{b}_1 \nabla \mathsf{c}_1)$.

8.6 Two-party exclusive-or

Our final step is to split the two-party exclusive-or into two separate assignments. This is achieved by introducing a local shared variable h that is visible to B, C only, i.e. not to A, and encrypting both hidden variables with it. Thus we take the step

$$(\mathsf{a}_B \nabla \mathsf{a}_C) := \mathsf{b}_\mathsf{a} \nabla \mathsf{c}_\mathsf{a}$$

$$
\begin{array}{ll}
= & |\!|[\ \mathbf{vis}_{B,C}\ \mathsf{h};\\
& \quad \mathsf{h}{:=}\,\mathsf{true} \oplus \mathsf{false};\\
& \quad \mathsf{a}_B{:=}\,\mathsf{b}_a \nabla \mathsf{h};\\
& \quad \mathsf{a}_C{:=}\,\mathsf{c}_a \nabla \mathsf{h}\\
&]|\!| \ ,
\end{array}
$$

justified trivially for B, C since the only assignments of non-constants are to variables visible only to A. For A the justification comes from the the use of classical equality reasoning within a temporary atomicity block (refer Lem. 1): the effect of the two fragments above on $\mathsf{a}_B, \mathsf{a}_C$ is identical, and there are no overwritten visible values.

We will now show that in fact the extra variable h is not necessary: by absorbing it into earlier statements, and with some rearrangement of scopes we can rewrite our code at the end of §8.5 as

$$
\begin{array}{ll}
\ldots = & |\!|[\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1;\ \ \mathbf{vis}_{B,C}\ \mathsf{h};\\
& \quad \mathsf{h}{:=}\,\mathsf{true} \oplus \mathsf{false};\\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\,\mathsf{b} \wedge \mathsf{c};\\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\,\mathsf{b} \vee \mathsf{c};\\
& \quad \mathsf{a}_B{:=}\,\mathsf{b}_a \nabla \mathsf{h};\\
& \quad \mathsf{a}_C{:=}\,\mathsf{c}_a \nabla \mathsf{h};\\
& \quad \mathbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C\\
&]|\!| \ldots
\end{array}
$$

where in fact we have moved the declaration and initialisation of h right to the beginning. We now absorb it into the earlier two-party computations by introducing temporarily variables $\mathsf{b}'_{\{0,1\}}$ and $\mathsf{c}'_{\{0,1\}}$ which correspond to their unprimed versions except that they, too, are encrypted with h. That gives

$$
\begin{array}{ll}
\ldots = & |\!|[\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C; \qquad\qquad \text{“Boolean reasoning”}\\
& \quad \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1, \mathsf{b}'_0, \mathsf{b}'_1;\\
& \quad \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1, \mathsf{c}'_0, \mathsf{c}'_1;\\
& \quad \mathbf{vis}_{B,C}\ \mathsf{h};\\
& \quad \mathsf{h}{:=}\,\mathsf{true} \oplus \mathsf{false};\\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\,\mathsf{b} \wedge \mathsf{c}; \quad \mathsf{b}'_0, \mathsf{c}'_0{:=}\,\mathsf{b}_0 \nabla \mathsf{h}, \mathsf{c}_0 \nabla \mathsf{h};\\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\,\mathsf{b} \vee \mathsf{c}; \quad \mathsf{b}'_1, \mathsf{c}'_1{:=}\,\mathsf{b}_1 \nabla \mathsf{h}, \mathsf{c}_1 \nabla \mathsf{h};\\
& \quad \mathsf{a}_B{:=}\,\mathsf{b}'_a; \qquad\qquad \leftarrow \text{Note primes, justified by earlier.}\\
& \quad \mathsf{a}_C{:=}\,\mathsf{c}'_a; \qquad\qquad \leftarrow \text{Assignments to } \mathsf{b}'_{\{0,1\}}, \mathsf{c}'_{\{0,1\}}.\\
& \quad \mathbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C\ ]|\!| \ldots
\end{array}
$$

where we have replaced the $\mathsf{b}_a \nabla \mathsf{h}$ and $\mathsf{c}_a \nabla \mathsf{h}$ at the end of the code with their simpler, primed versions where the encryption in built-in. Now we can rearrange the statements using h so that not only h but also the unprimed $\mathsf{b}_{\{0,1\}}$ and $\mathsf{c}_{\{0,1\}}$ become auxiliary; that is, we have for the conjunction

$$(\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\,\mathsf{b} \wedge \mathsf{c}; \quad \mathsf{b}'_0, \mathsf{c}'_0{:=}\,\mathsf{b}_0 \nabla \mathsf{h}, \mathsf{c}_0 \nabla \mathsf{h}$$

$$
\begin{array}{l}
|||[\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \\
\quad \mathsf{b}_0 := \mathsf{true} \oplus \mathsf{false}; \\
\quad \mathsf{b}_1 := \mathsf{true} \oplus \mathsf{false}; \\
\\
\left.\begin{array}{l}
\quad \mathsf{c}_0 := \ (\mathsf{b} \nabla \mathsf{b}_0\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{b}_0); \\
\quad \mathsf{c}_1 := \ (\neg \mathsf{b}_1\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{b} \nabla \mathsf{b}_1); \\
\quad \mathsf{a}_B := (\mathsf{b}_1\ \mathbf{if}\ \mathsf{a}\ \mathbf{else}\ \mathsf{b}_0); \\
\quad \mathsf{a}_C := (\mathsf{c}_1\ \mathbf{if}\ \mathsf{a}\ \mathbf{else}\ \mathsf{c}_0);
\end{array}\right\} \textit{Four oblivious transfers.} \\
\\
\quad \mathbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C\ ]||| \\
\end{array}
$$

We replace the two Two-Party 'junctions by their implementations as oblivious transfers: each becomes two statements instead of one. The random flipping of bits $\mathsf{b}_{\{0,1\}}$ is then collected at the start.

The preservation of correctness is guaranteed by the compositionality of the security semantics.

Fig. 2. Three-Judges Protocol assuming Oblivious Transfers as primitives

$$
\begin{array}{lll}
= & \langle\!\langle\ (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c};\quad \mathsf{b}'_0, \mathsf{c}'_0 := \mathsf{b}_0 \nabla \mathsf{h}, \mathsf{c}_0 \nabla \mathsf{h}\ \rangle\!\rangle & \text{“introduce atomicity”} \\
= & \langle\!\langle\ (\mathsf{b}'_0 \nabla \mathsf{c}'_0) := \mathsf{b} \wedge \mathsf{c};\quad \mathsf{b}_0, \mathsf{c}_0 := \mathsf{b}'_0 \nabla \mathsf{h}, \mathsf{c}'_0 \nabla \mathsf{h}\ \rangle\!\rangle & \text{“classical equality”} \\
= & (\mathsf{b}'_0 \nabla \mathsf{c}'_0) := \mathsf{b} \wedge \mathsf{c};\quad \mathsf{b}_0, \mathsf{c}_0 := \mathsf{b}'_0 \nabla \mathsf{h}, \mathsf{c}'_0 \nabla \mathsf{h}\ , & \text{“remove atomicity”}
\end{array}
$$

and similarly for the disjunction. Removing the auxiliaries, and then applying a trivial renaming to get rid of the primes, we end up with

$$
\begin{array}{lll}
\ldots = & |||[\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; & \text{“Consolidating the above”} \\
& \quad (\mathsf{b}_0 \nabla \mathsf{c}_0) := \mathsf{b} \wedge \mathsf{c}; & \leftarrow \text{Two-Party Conjunction (contains Oblivious Transfer).} \\
& \quad (\mathsf{b}_1 \nabla \mathsf{c}_1) := \mathsf{b} \vee \mathsf{c}; & \leftarrow \text{Two-Party Disjunction (contains Oblivious Transfer).} \\
& \quad \mathsf{a}_B := \mathsf{b}_a; & \leftarrow \text{Oblivious Transfer.} \\
& \quad \mathsf{a}_C := \mathsf{c}_a; & \leftarrow \text{Oblivious Transfer.} \\
& \quad \mathbf{reveal}\ \mathsf{a}_B \nabla \mathsf{a}_C\ ||| &
\end{array}
$$

which is precisely what we sought.

In Fig. 2 we give the code with the (two) two-party computations instantiated. In Fig. 3 we instantiate one of the (four) oblivious transfers.

9 Conclusion: a challenge and an open problem

We have investigated the foundations for probabilistic non-interference security by proposing a semantics, and a refinement order between its programs, which we have demonstrated has connections with existing entropy-based measures. Especially it is related to Bayes Risk and we have given a soundness and completeness result that establishes compositional closure.

Our approach has a general goal: to justify practical methods which support accurate analysis of programs operating in a context of probabilistic uncertainty.

Starting from Fig. 2, we replace the specification of the first of its four oblivious transfers $\mathsf{c}_0{:=}(\mathsf{b}\nabla\mathsf{b}_0$ **if** c **else** $\mathsf{b}_0)$ by an implementation in elementary terms [23]:

$$
\begin{array}{l}
|[\ \mathbf{vis}_A\ \mathsf{a}_B, \mathsf{a}_C;\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \\
\quad \mathsf{b}_0{:=}\,\mathsf{true} \oplus \mathsf{false}; \\
\quad \mathsf{b}_1{:=}\,\mathsf{true} \oplus \mathsf{false}; \\
\\
\quad |[\ \mathbf{vis}_B\ \mathsf{m}'_0, \mathsf{m}'_1; \mathbf{vis}_C\ \mathsf{c}', \mathsf{m}'; \\
\quad\quad \mathsf{c}'{:=}\,\mathsf{true} \oplus \mathsf{false}; \\
\quad\quad \mathsf{m}'_0{:=}\,\mathsf{true} \oplus \mathsf{false};\ \mathsf{m}'_1{:=}\,\mathsf{true} \oplus \mathsf{false}; \\
\quad\quad \mathsf{m}'{:=}\,\mathsf{m}'_{\mathsf{c}'}; \qquad \leftarrow \text{Done in advance by trusted third party. } \dagger \\
\\
\quad\quad \mathbf{vis}_{ABC}\ \mathsf{x}, \mathsf{y}_0, \mathsf{y}_1; \quad \leftarrow \text{Note these are visible to all three agents.} \\
\quad\quad \mathsf{x}{:=}\,\mathsf{c}\nabla\mathsf{c}'; \\
\quad\quad \mathsf{y}_0{:=}\,\mathsf{b}_0\nabla\mathsf{m}'_{\mathsf{x}}; \\
\quad\quad \mathsf{y}_1{:=}\,\mathsf{b}\nabla\mathsf{b}_0\nabla\mathsf{m}'_{\neg\mathsf{x}}; \\
\quad\quad \mathsf{c}_0{:=}\,\mathsf{y}_c\nabla\mathsf{m}' \qquad \leftarrow \text{Although } \mathsf{y}_\mathsf{c} \text{ is public, only Agent } C \text{ knows } \mathsf{m}'. \ \ddagger \\
\quad]|; \\
\\
\quad \left.\begin{array}{l} \mathsf{c}_1{:=}\ (\neg\mathsf{b}_1 \textbf{ if } \mathsf{c} \textbf{ else } \mathsf{b}\nabla\mathsf{b}_1); \\ \mathsf{a}_B{:=}\ (\mathsf{b}_1 \textbf{ if } \mathsf{a} \textbf{ else } \mathsf{b}_0); \\ \mathsf{a}_C{:=}\ (\mathsf{c}_1 \textbf{ if } \mathsf{a} \textbf{ else } \mathsf{c}_0); \end{array}\right\} \begin{array}{l} \textit{Three more oblivious transfers,} \\ \text{each one to be} \\ \text{expanded as above.} \end{array} \\
\\
\quad \mathbf{reveal}\ \mathsf{a}_B\nabla\mathsf{a}_C\ ]|
\end{array}
$$

Each of the other three transfers would expand to a similar block of code, making about 40 lines of code in all. The Oblivious Transfer is formally derived elsewhere [27]; an informal explanation is given in §F.

The preservation of correctness, under expansion, is again guaranteed by the compositionality of the security semantics.

Note that aside from the statement marked † (and its three other instances within the three other, unexpanded oblivious transfers), all messages are wholly public because of the declarations **vis** $\mathsf{x}, \mathsf{y}_0, \mathsf{y}_1$; that is, all the privacy needed is provided already by the exclusive-or'ing with hidden Booleans, as ‡ shows.
The only private communications ($\dagger \times 4$) are done with the aid of a trusted third party. As explained by Rivest [32, 27] this party's involvement occurs only *before* the protocol begins, and it is trusted not to observe any data exchanged subsequently between the agents; alternatively, the subsequent transfers can themselves be encrypted without affecting the protocol's correctness. (A trusted third party without these limitations would implement trivially any protocol of this kind, simply by collecting the secret data, processing it and then distributing the result.)

Fig. 3. Three-Judges Protocol in elementary terms

Abstraction underlies tractable analysis, but the results of such analyses become relevant *only if* the method of abstraction aptly preserves the properties intended for examination. The impact of this research is to show firstly that our refinement order aptly characterises Bayes Risk, and secondly that the former discrepancies

between Bayes Risk and other information orders can be rationalised by taking contexts into account.

By taking a fresh point of view, we have related entropies that were formerly thought to be inconsistent. Furthermore, we highlight the *similarities* between non-interference (as defence against an adversary) and large-scale structuring techniques (such as stepwise refinement and its associated information hiding [29]) for probabilistic systems. Both require a careful distinction between what data can be observed and what data must be protected; by observing that distinction in the definition of abstraction, we allow the tractable analysis of properties which rely on "secrecy" (on the one hand) or "probabilistic local state" (on the other). This unified semantic foundation opens up the possibility for a uniform approach to the specification of security properties, along with other safety-critical features, during system design [20].

These positive results now present a challenge and an open problem. The challenge is to find a model where all three features, probability, nondeterminism and hidden state, can reside together, and then an equivalence between semantic objects which respects an appropriate definition of testing. The presence of nondeterminism would then include a treatment of distributed systems with schedulers having a restricted view of the state [4]; that is because nondeterminism can be interpreted either as underspecification, or as a range of decisions presented to a scheduler. Within such a model we would be increasing the power of the adversary to harvest information about the hidden state by increasing the expressivity of the contexts she can create. It is an open problem whether that increased power is sufficient to make the various information-theoretic orders (Bayes Risk, Shannon Entropy, Marginal Guesswork etc.) equivalent or whether they remain truly distinct.

Related techniques

The use of information orders, such as those summarised in §1.2, to determine the extent to which programs leak their secrets is widespread. Early work that took this approach includes [25, 38, 13], and more recently it has been employed in [34, 15, 1, 7, 18]. One of the contributions of this paper is to show how those evaluations can be related by taking a refinement-oriented perspective. Compositionality plays a major role in our definition of refinement and we note that other orders between probability distributions such as the "peakedness" introduced by Dubois and Hüllermeir [9] appear not to be compositional when generalised to our hyperdistributions.

More significant than the particular information order is the way that it is used in the analysis of programs. Our approach uses specifications to characterise permitted leaks, and a refinement order which ensures that for our chosen information order (i.e. Bayes Risk), the implementation is at least as secure as its specification. An alternative mode is taken by Braun et al. [1]. Rather than restricting the elementary *testing-relation* ($\preceq$) to a compositional subset ($\sqsubseteq$), they identify the *safe* contexts $\mathcal{C}_{\text{safe}}$ such that I "$\preceq$" $\mathcal{C}_{\text{safe}}(I)$. With our emphasis

on implementations I *and* their specifications S, by analogy we would be looking for $S \preceq I$ implies $\mathcal{C}_{\text{safe}}(S) \preceq \mathcal{C}_{\text{safe}}(I)$.

Building on the theoretical approaches, others have investigated the use of automation to evaluate the quantitative weaknesses in programs. Heusser and Malacaria [12], for example, have automated a technique based on Shannon entropy. Andrés et al. [17] similarly consider efficient calculation of information leakage, which can provide diagnostic feedback to the designer.

In some ways our semantics is related in structure to *Hidden Markov Models* [14] suggesting that, in the future, the algorithmic methods developed in that field might apply to the special concerns of program development. A Hidden Markov Model considers a system partitioned into hidden states (our h) and observable outputs (similar to our v). The h-state evolves according to a Markov Chain, in our terms repeated execution of a fragment $\mathsf{h}{:\in} D.\mathsf{h}$ in which the probability of the next state h' is given by a fixed "matrix" D as $D.h.h'$ where h is the current state. Associated with each transition is an observation, in our terms execution of a fragment $\mathsf{v}{:\in} E.\mathsf{h}$. Put together, therefore, the *HMM* evolves according to repeated executions of the fragment

$$\mathsf{h}{:\in} D.\mathsf{h};\ \mathsf{v}{:\in} E.\mathsf{h}\ , \tag{19}$$

which fragment is a special case of our probabilistic-choice statements since the distributions on the right in (19) do not depend on v, whereas in Fig. 1 they can.

The canonical problems associated with *HMM*'s are (in the terms above)

1. Given the source code (that is, the matrices D, E), compute the probability of observing a given sequence of values assigned to v.
2. Given a sequence of output values, determine the most likely values of D, E.
3. Given the source code and a particular sequence of values assigned to v, calculate the sequence of values assigned to h that was most likely to have occurred.

The first of those is basically the classical semantics [16, 21], but projected onto v since we are not interested in h's values. The second we do not treat at all — it is tantamount to trying to guess a program's source code (in a limited repertoire) given the outputs it produces. The third is closest to our security concerns, since it is in a sense trying to guess h from observation of v.

But in fact we address none of the three problems directly, since even in the third case we have a different concern: in *HMM* terms we are comparing *two* systems D_S, E_S and D_I, E_I, asking whether –according to certain entropy measures– the entropy of the a-posteriori distribution of the final value of h is at least as secure in system D_I, E_I as it is in D_S, E_S. Furthermore, our concern with compositionality would in *HMM* terms relate to the question of embedding each of D_S, E_S and D_I, E_I "inside" another system D, E.

The application of *HMM* techniques to our work would in the first instance probably be in the efficient calculation of whether D_S, E_S, the specification, was secure enough for our purposes: once that was done, the refinement relation would ensure that the implementation D_I, E_I was also secure enough, without

requiring a second calculation. The advantage of this is that the first calculation, over a smaller and more abstract system, is likely to be much simpler than the second would have been.

References

1. C. Braun, K. Chatzikokolakis, and C. Palamidessi. Compositional methods for information-hiding. In *Proc. FOSSACS'08*, volume 4962 of *LNCS*, pages 443–57. Springer, 2008.
2. C. Braun, K. Chatzikokolakis, and C. Palamidessi. Quantitative notions of leakage for one-try attacks. In *Proc. MFPS*, volume 249 of *ENTCS*, pages 75–91. Elsevier Science Publishers B. V., 2009.
3. Christian Cachin. *Entropy measures and unconditional security in cryptography*. PhD thesis, ETH, Zürich, Switzerland, 1997.
4. K. Chatzikokolakis and C. Palamidessi. Making random choices invisible to the scheduler. *Information and Computation*, 208(6):694–715, 2010.
5. K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Probability of error in information-hiding protocols. In *Proc. CSF*, pages 341–354. IEEE Computer Society, 2007.
6. D. Chaum. The Dining Cryptographers problem: Unconditional sender and recipient untraceability. *Jnl Cryptol.*, 1(1):65–75, 1988.
7. M.R. Clarkson, A.C. Myers, and F.B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.
8. M. de Nicola and M. Hennessy. Testing equivalence for processes. *Theo Comp Sci*, 34:83–133, 1984.
9. D. Dubois and E. Hüllermeier. A notion of comparative probabilistic entropy based on the possibilistic specificity ordering. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 3571 of *LNCS*, pages 848–859. Springer, 2005.
10. J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp on Security and Privacy*, pages 75–86. IEEE Computer Society, 1984.
11. Probabilistic Systems Group. Collected publications. `www.cse.unsw.edu.au/~carrollm/probs`.
12. J. Heusser and P. Malacaria. Formal aspects in security and trust. In *Applied Quantitative Information Flow and Statistical Databases*, volume 5983 of *LNCS*, pages 96–110. Springer, 2010.
13. J.W. Gray III. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy*, pages 21–35. IEEE Computer Society, 1991.
14. D. Jurafsky and J.H. Martin. *Speech and Language Processing*. Prentice Hall International, 2000.
15. B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proc. 14th ACM Conf. Comp. Comm. Security*, 2007.
16. D. Kozen. A probabilistic PDL. *Jnl Comp Sys Sci*, 30(2):162–78, 1985.
17. Miguel M. Andrés, C. Palamidessi, P. Van Rossum, and G.Smith. Computing the leakage of information-hiding systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *LNCS*, pages 373–389. Springer, 2010.

18. P. Malacaria. Risk assessment of security threats for looping constructs. *Journal of Computer Security*, 18(2):191–228, 2010.
19. J.L. Massey. Guessing and entropy. In *Proc. IEEE International Symposium on Information Theory*, page 204, 1994.
20. A.K. McIver, , L.A. Meinicke, and C.C. Morgan. Security, probability and nearly fair coins in the cryptographers' café. In A. Cavalcanti and D. Dams, editors, *Proc FM '09*, volume 5850 of *LNCS*. Springer, 2009. Invited presentation.
21. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Tech Mono Comp Sci. Springer, New York, 2005.
22. A.K. McIver and C.C. Morgan. A calculus of revelations. Presented at VSTTE Theories Workshop `www.cs.york.ac.uk/vstte08/`, October 2008.
23. 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*, volume 5850 of *LNCS*. Springer, 2009.
24. A.K. McIver and C.C. Morgan. *The Thousand-and-One Cryptographers*. Prentice Hall International, 2010.
25. J.K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66. IEEE Computer Society, 1987.
26. 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*.
27. C.C. Morgan. *The Shadow Knows:* Refinement of ignorance in sequential programs. *Science of Computer Programming*, 74(8):629–653, 2009. Treats *Oblivious Transfer*.
28. 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`.
29. D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
30. J.O. Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *Progress in Cryptology (INDOCRYPT 2000)*, volume 1977 of *LNCS*, pages 67–79. Springer, 2000.
31. M.O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard University, 1981. Available at `eprint.iacr.org/2005/187`.
32. R. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initialiser. Technical report, M.I.T., 1999. `//theory.lcs.mit.edu/~rivest/Rivest-commitment.pdf`.
33. C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.
34. G. Smith. Adversaries and information leaks (Tutorial). In G. Barthe and C. Fournet, editors, *Proc. 3rd Symp. Trustworthy Global Computing*, volume 4912 of *LNCS*, pages 383–400. Springer, 2007.
35. K. Trustrum. *Linear Programming*. Library of Mathematics. Routledge and Kegan Paul, London, 1971.
36. D. Welsh. *Codes and Cryptography*. Oxford Science Publications, 1988.
37. N. Wirth. Program development by stepwise refinement. *Comm ACM*, 14(4):221–7, 1971.
38. J.T. Wittbold and D.M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161. IEEE Computer Society, 1990.

39. A.C-C. Yao. Protocols for secure computations (extended abstract). In *Annual Symposium on Foundations of Computer Science (FOCS 1982)*, pages 160–164. IEEE Computer Society, 1982.

A Proofs for partition-based matrix representations

We give here the proofs for properties we relied on in §7.

Property 9 (in §7.1): Convex closure of refinement matrices To show that the set of $N\times N$ refinement matrices is the convex closure of the transpose of the set of $N\times N$ strategy matrices, i.e. that

$$\mathcal{R}_N \quad = \quad \mathsf{ccl.}(\mathcal{M}_N) \ ,$$

we first observe ($\supseteq$) that every element in $\mathsf{ccl.}(\mathcal{M}_N)$ is trivially non-negative and column-one-summing (that is, it is a refinement matrix). It remains to show ($\subseteq$) that any refinement matrix R in $\mathcal{R}_N$ can be expressed as an interpolation of matrices from $\mathcal{M}_N$.

We argue as follows. Fix R, and identify a non-zero minimum element in each of its columns; let c be the minimum of those column-minima; select the M in $\mathcal{M}_N$ that has 1's in the column-minima positions exactly; and subtract cM from R to give some R'.

Now R' has at least one more 0 entry than R did, and yet the columns of R' still have equal sums, now $1-c$. Continue this process from R' onwards: it must stop, since the number of 0's increases each time; and when it does stop it must be because there is an all-0 column, in which case *all* columns must be all-0, since the column sums have remained the same all the way through.

The collection of M's and their associated c's is the interpolation we had to find: for example, in three steps the procedure generates the interpolation

$$\begin{pmatrix} 1/3 & 3/4 \\ 2/3 & 1/4 \end{pmatrix} \quad = \quad 1/4 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 1/12 \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + 2/3 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \ .$$

Property (11) (in §7.1): refinement matrices form a monoid Since matrix multiplication is associative and the identity $\mathbf{1}_N$ is an element of $\mathcal{R}_N$, we need only demonstrate that $\mathcal{R}_N$ is closed under multiplication. That can be checked by direct calculation.

B Secure semantics via matrices

In §7 we appealed to matrix representations of partitions to construct our proof that ($\sqsubseteq$) is the compositional closure of ($\preceq$). Here we we project the rest of our semantics into matrix algebra, giving matrix representations of split-states, hyper-distributions, programs, and refinement. These representations are used to verify both monotonicity (Thm. 2 from §6.5) in §C.2 and the Atomicity Lemma (Lem. 1 from §8.1) in §E.

B.1 Notation

For i taken from some ordered index set I we will write $(\ddagger i{:}\, I \mid R \bullet M_i)$ for the vertical concatenation of matrices M_i for those i satisfying R, taken in I-order: for this to be well defined, the column-count must be the same for all M's; but their row-counts may differ. In the same way we will write $(+\!\!+ i{:}\, I \mid R \bullet M_i)$ for horizontal matrix concatenation (in which case the row-counts must agree).

For a given dimension N and expression $E.\mathsf{n}$ we write $\backslash\!\backslash E.\mathsf{n} \backslash\!\backslash$ for the $N{\times}N$ diagonal matrix whose value at the element (doubly) indexed n is $E.n$. Thus for example we have $\mathsf{1} = \backslash\!\backslash 1 \backslash\!\backslash$.

B.2 Split-states as single-column matrices

Let N be the cardinality of $\mathcal{V}{\times}\mathcal{H}$. A split-state has type $\mathcal{V}{\times}\mathsf{D}\mathcal{H}$ and can be written as an $1{\times}N$ matrix, a row of probabilities in some agreed-upon index order of $\mathcal{V}{\times}\mathcal{H}$ where the element at (column) index (v, h) gives the probability $\delta.h$ associated with that pair.

Naturally the row sums to 1 but –more than that– each such representation of a split-state will have nonzero entries only in columns whose first index-component is the v appearing in (v, δ). We say that such a row is $\mathcal{V}$-*unique* and that it has *characteristic* v.

Write $\mathsf{1}_v$ for the $N{\times}N$ diagonal matrix $\backslash\!\backslash 1$ **if** $\mathsf{v}{=}v$ **else** $0 \backslash\!\backslash$ having ones only at positions whose row- (or equivalently column-) index has that v as its first component; elsewhere in the diagonal (and everywhere off the diagonal) the entries are zero. The row-matrix representation $(\!(v, \delta)\!)$ of split-state (v, δ) then satisfies $(\!(v, \delta)\!) = (\!(v, \delta)\!){\times}\mathsf{1}_v$ because it has characteristic v, so that the multiplication by $\mathsf{1}_v$ sets to zero only elements that were zero already.

B.3 Hyper-distributions as matrices

In §7.1 we interpreted whole partitions as matrices, with each row (fraction) giving a possible distribution over $\mathcal{H}$ for some fixed v. Here we proceed similarly, but we do not fix v, so that a hyper-distribution Δ whose support has cardinality F is represented as an $F{\times}N$ matrix $(\!(\Delta)\!)$ each of whose rows is $\mathcal{V}$-unique, as above, thus independently representing some split-state (v, δ). Extending the matrix representation of individual split-states, we can represent whole hyper-distributions according to

$$(\!(\Delta)\!) \quad := \quad (\ddagger (v, \delta){:}\, \lceil \Delta \rceil \bullet \Delta.(v, \delta) * (\!(v, \delta)\!)) \tag{20}$$

where, as in §7.1, with the multiplier $\Delta.(v, \delta)$ we are scaling the rows so that the total weight of each gives the probability of that split-state in the hyper-distribution overall; the distribution the split-state actually contains (the δ in the (v, δ) that the row represents) is as usual recoverable by normalising. Because each of the rows is $\mathcal{V}$-unique we say that the matrix as a whole, also, is $\mathcal{V}$-unique; but note that it is possible to have several rows with the same characteristic v.

$\mathcal{V}$-uniqueness means that no two distinct v's appear with non-zero probability in the *same* row.

As for partitions, in such matrices we define *similarity* between rows and say that a hyper-distribution is in *reduced* matrix representation if all its similar rows have been added together, and all its all-zero rows have been removed. We say that two hyper-distribution matrices are *similar* ($\approx$) if their reductions are equivalent up to a reordering of rows. Similarity is a congruence for matrix multiplication on the right, but not on the left; vertical concatenation ($\pm$) respects similarity on both sides.

While the column-order of $(\!(\Delta)\!)$ is fixed by our (arbitrary) ordering of $\mathcal{V}\times\mathcal{H}$, the row-order might vary since there is no intrinsic order on fractions. We therefore regard $(\!(\Delta)\!)$ as determined only up to similarity, and our reasoning below will be restricted to operations for which similarity is a congruence. In particular we have that $(\!(\Delta_1)\!)\approx(\!(\Delta_2)\!)$ implies $\Delta_1{=}\Delta_2$, i.e. that $(\!(\cdot)\!)$ is injective up to similarity.

The operation $(\!(\cdot)\!)$ on (sub-)hyper-distributions is linear in the sense that

$$\begin{array}{rrcl} & (\!(p{*}\Delta)\!) & = & p * (\!(\Delta)\!) \\ \text{and} & (\!(\Delta_1{+}\Delta_2)\!) & \approx & (\!(\Delta_1)\!) \pm (\!(\Delta_2)\!)\ . \end{array} \tag{21}$$

B.4 Classical commands as matrices

We recall from §2.3 that the classical "relational" semantics $[\![P]\!]_C$ of a program P is a function $\mathcal{V}\times\mathcal{H} \to \mathsf{D}(\mathcal{V}\times\mathcal{H})$ and may hence be treated (just as $D.\mathsf{v.h}$ from §7.1 was) as an $N\times N$ matrix written $(\!|P|\!)$ whose value in row (v,h) and column (v',h') is just $[\![P]\!]_C.(v,h).(v',h')$. [30]

Sequential composition between classical commands is then represented by matrix multiplication, in the usual Markov style, so that we have

$$(\!|P_1; P_2|\!) \quad = \quad (\!|P_1|\!) \times (\!|P_2|\!)\ . \tag{22}$$

B.5 Secure commands as matrices

We will establish that for any secure program P there is an I-indexed set of $N\times N$ matrices such that

$$(\!(\, [\![P]\!].(v,\delta)\,)\!) \quad \approx \quad (\pm i{:}\, I \bullet (\!(v,\delta)\!) \times M_i) \tag{23}$$

for any split-state (v,δ). We think of the matrices as giving a *normal form* for P. Using the normal form, we will be able to represent the lifting of P's secure semantics using matrix operations, since then

$$(\!(\, (\odot\,(v,\delta){:}\,\Delta \bullet [\![P]\!].(v,\delta))\,)\!) \quad \approx \quad (\pm i{:}\, I \bullet (\!(\Delta)\!) \times M_i) \tag{24}$$

can be established by the calculation

[30] Note that operation $(\!|\cdot|\!)$ applies to texts, i.e. syntax but $(\!(\cdot)\!)$ applies to hyper-distributions, i.e. semantics.

$$
\begin{array}{lll}
 & (\!(\, (\odot\,(v,\delta)\colon \Delta \bullet [\![P]\!].(v,\delta))\,)\!) & \\
= & (\!(\, (\sum(v,\delta)\colon \lceil\Delta\rceil \bullet \Delta.(v,\delta) * [\![P]\!].(v,\delta)\,)\,)\!) & \text{“definition expected value §2.1”} \\
\approx & (\ddagger(v,\delta)\colon \lceil\Delta\rceil \bullet \Delta.(v,\delta) * ((\![P]\!].(v,\delta)))) & \text{“from (21)”} \\
\approx & (\ddagger(v,\delta)\colon \lceil\Delta\rceil \bullet \Delta.(v,\delta) * (\ddagger i\colon I \bullet (\!(v,\delta)\!) \times M_i)\,) & \text{“normal form (23)”} \\
= & (\ddagger(v,\delta)\colon \lceil\Delta\rceil; i\colon I \bullet (\Delta.(v,\delta) * (\!(v,\delta)\!)) \times M_i) & \text{“distribute multiplication”} \\
\approx & (\ddagger i\colon I \bullet (\ddagger(v,\delta)\colon \lceil\Delta\rceil \bullet \Delta.(v,\delta) * (\!(v,\delta)\!)) \times M_i) & \text{“rearrange rows; distribute post-multiplication”} \\
\approx & (\ddagger i\colon I \bullet (\!(\Delta)\!) \times M_i)\ . & \text{“from (20) defining } (\!(\Delta)\!)\text{”}
\end{array}
$$

We now show by structural induction how embedded classical commands, general choice, sequential composition (and hence all of our secure commands) can be translated into this normal form.

Embedded classical commands In Def. 3 from §8.1 we gave the semantics $[\![\langle\!\langle P\rangle\!\rangle]\!]$ of a program P considered as an atomic unit; we now do the same here in matrix style.

If we were to execute an atomic program $\langle\!\langle P\rangle\!\rangle$ from a split-state (v,δ), in the matrix style we would begin by calculating $(\!(v,\delta)\!)\times(\!|P|\!)$, giving again a single row; but that row might not be $\mathcal{V}$-unique, in which case a further step would be needed. We'd split its possibly non-unique rows into (maximally) $\mathcal{V}$-unique portions, an operation that corresponds roughly to the embed funtion used in Def. 1.

Given a row matrix R that is $\mathcal{V}\times\mathcal{H}$-indexed by column (such as the output $(\!(v,\delta)\!)\times(\!|P|\!)$ from just above) the splitting of its possibly non-unique row is achieved via

$$\mathsf{embed}.R \quad := \quad (\ddagger v'\colon \mathcal{V} \bullet R \times 1_{v'})\ , \tag{25}$$

in which each of the values v' in $\mathcal{V}$ is used, in turn, to construct a row matrix of characteristic v' projected from R by zeroing all other entries: those characteristic-v' projections are then stacked on top of each other with $(\ddagger)$ to make a single (possibly quite tall!) matrix that is derived from R but now is $\mathcal{V}$-unique. [31] With that apparatus, we have

$$(\!(\, [\![\langle\!\langle P\rangle\!\rangle.(v,\delta)]\!]\,)\!) \quad \approx \quad (\ddagger v'\colon \mathcal{V} \bullet (\!(v,\delta)\!) \times (\!|P|\!) \times 1_{v'})\ , \tag{26}$$

thereby giving the $\mathcal{V}$-unique matrix representation (up to similarity) of the hyper-distribution output by $\langle\!\langle P\rangle\!\rangle$ if executed from incoming split-state (v,δ). [32]

General choice For both general choice and sequential composition we assume inductively that the semantics of subprograms P_1 and P_2 can be written in

[31] For example, if the row R is $\mathcal{V}$-unique already then $R'{:=}\,\mathsf{embed}.R$ will stack up a great many all-zero rows. But still we will have $R{\approx}R'$, so no damage is done.

[32] Note the algebra of similarity here: if we have $R{\approx}(\!(v,\delta)\!)$ for some R, then $(\ddagger v'\colon \mathcal{V} \bullet R \times (\!|P|\!) \times 1_{v'})$ is similar to the right-hand side above.

matrix normal form so that for each split-state (v,δ) we have

$$(\!(\, [\![P_i]\!].(v,\delta)\,)\!) \quad \approx \quad (\ddagger j_i\colon J_i \bullet (\!(v,\delta)\!) \times M_{i,j_i})\ .$$

To show that general choice can be expressed in matrix normal form, we use the following identity which expresses the conditioning of a split-state (v,δ) by expression $E.\mathsf{v.h}$ in terms of matrix operations:

$$(\odot h\colon \delta \bullet E.v.h) * (\!(v, \{\!|h\colon \delta \mid E.v.h|\!\}\!)) \quad = \quad (\!(v,\delta)\!) \times \backslash\!\backslash E.\mathsf{v.h} \backslash\!\backslash\ . \qquad (27)$$

We then have

$(\!(\, [\![P_1 \ {}_{q.\mathsf{v.h}}\!\oplus P_2]\!].(v,\delta)\,)\!)$

$=$ "general choice from Fig. 1; $p{:=}(\odot h\colon \delta \bullet q.v.h)$"
$(\!(\, p * [\![P_1]\!].(v, \{\!|h\colon \delta \mid q.v.h|\!\}) + (1{-}p) * [\![P_2]\!].(v, \{\!|h\colon \delta \mid 1{-}q.v.h|\!\})\,)\!)$

$\approx$ "from (21)"
$p * (\!(\, [\![P_1]\!].(v, \{\!|h\colon \delta \mid q.v.h|\!\})\,)\!) \ \ddagger\ (1{-}p) * (\!(\, [\![P_2]\!].(v, \{\!|h\colon \delta \mid 1{-}q.v.h|\!\})\,)\!)$

$\approx$ "inductive assumption: matrix normal form of P_1 and P_2"
$p * \ (\ddagger j_1\colon J_1 \bullet (\!(v, \{\!|h\colon \delta \mid q.v.h|\!\})\!) \times M_{1,j_1})$
$\ddagger\ (1{-}p) * \ (\ddagger j_2\colon J_2 \bullet (\!(v, \{\!|h\colon \delta \mid 1{-}q.v.h|\!\})\!) \times M_{2,j_2})$

$=$ "distribute scalar multiplications"
$(\ddagger j_1\colon J_1 \bullet (p * (\!(v, \{\!|h\colon \delta \mid q.v.h|\!\})\!))) \times M_{1,j_1})$
$\ddagger\ (\ddagger j_2\colon J_2 \bullet ((1{-}p) * (\!(v, \{\!|h\colon \delta \mid 1{-}q.v.h|\!\})\!))) \times M_{2,j_2})$

$=$ "recall $p{:=}(\odot h\colon \delta \bullet q.v.h)$; from (27)"
$(\ddagger j_1\colon J_1 \bullet (\!(v,\delta)\!) \times \backslash\!\backslash q.\mathsf{v.h} \backslash\!\backslash \times M_{1,j_1})$
$\ddagger\ (\ddagger j_2\colon J_2 \bullet (\!(v,\delta)\!) \times \backslash\!\backslash 1{-}q.\mathsf{v.h} \backslash\!\backslash \times M_{2,j_2})$

$=$ "Let $p_1.\mathsf{v.h}{:=}q.\mathsf{v.h}$ and $p_2.\mathsf{v.h}{:=}1{-}q.\mathsf{v.h}$"
$(\ddagger i\colon \{1,2\}; j\colon J_i \bullet (\!(v,\delta)\!) \times \backslash\!\backslash p_i.\mathsf{v.h} \backslash\!\backslash \times M_{i,j})\ .$

Sequential composition For sequential composition of P_1 and P_2 we have

$(\!(\, [\![P_1; P_2]\!].(v,\delta)\,)\!)$
$=$ $(\!|\, (\odot (v',\delta')\colon [\![P_1]\!].(v,\delta) \bullet [\![P_2]\!].(v',\delta'))\,|\!)$ "Composition from Fig. 1"
$\approx$ $(\ddagger j_2\colon J_2 \bullet ((\!|[\![P_1]\!].(v,\delta)|\!)) \times M_{2,j_2})$ "(24); matrix normal form of P_2"
$\approx$ $(\ddagger j_1\colon J_1; j_2\colon J_2 \bullet (\!(v,\delta)\!) \times M_{1,j_1} \times M_{2,j_2})\ .$ "matrix normal form of P_1"

B.6 Refinement as matrix multiplication

In §7.1 we showed how refinement between partitions could be defined using matrix multiplication. We can promote this to hyper-distributions by dealing

with each v separately: we have that hyper-distribution Δ_S is refined by Δ_I just when for each $v{\in}\mathcal{V}$ there exists a refinement matrix (i.e. a non-negative, column one-summing matrix) R_v such that

$$R_v \times (\!|\Delta_S|\!) \times \mathbf{1}_v \quad \approx \quad (\!|\Delta_I|\!) \times \mathbf{1}_v \ . \tag{28}$$

The effect of requiring similarity for each v separately is to prevent rows with differing v's from being added together.

C Proofs for the refinement relation

C.1 Secure programs are partially ordered by ($\sqsubseteq$)

We show (Thm. 1 in §6.5) that the refinement relation ($\sqsubseteq$) defines a partial order over hyper-distributions. It follows by extension that it is a partial order over secure programs.

Reflexivity For any hyper-distribution Δ reflexivity holds trivially since, for each v, the intermediate partition $\mathsf{fracs}.\Delta.v$ is both similar to and as fine as itself.

Transitivity Assume that $\Delta_1{\sqsubseteq}\Delta_2$ and $\Delta_2{\sqsubseteq}\Delta_3$. It is enough to show that for each v we have $\mathsf{fracs}.\Delta_1.v \sqsubseteq \mathsf{fracs}.\Delta_3.v$. For each i let Π_i be the $N{\times}N$ matrix representation (§7.1) of $\mathsf{fracs}.\Delta_i.v$ for some N. To prove $\Pi_1{\sqsubseteq}\Pi_3$, we need to find a refinement matrix R_{31} such that Π_3 is $R_{31}{\times}\Pi_1$.

From above there are refinement matrices R_{32}, R_{21} with $\Pi_3 = R_{32}{\times}\Pi_2$ and $\Pi_2 = R_{21}{\times}\Pi_1$. Thus R_{31} defined $R_{32}{\times}R_{21}$ satisfies $\Pi_3 = R_{31}{\times}\Pi_1$, and it is a refinement matrix by Property (11) from §7.1.

Antisymmetry Assume that both $\Delta_1{\sqsubseteq}\Delta_2$ and $\Delta_2{\sqsubseteq}\Delta_1$ but $\Delta_1 \neq \Delta_2$. From the first and third assumptions, with Lem. 6 (§G.1) we have that the Shannon Entropy of Δ_1 is strictly less than that of Δ_2; from the second and third, we have the opposite — thus a contradiction.

C.2 Monotonicity of secure programs w.r.t. ($\sqsubseteq$)

We use the following technical results to verify that ($\sqsubseteq$) is monotonic with respect to secure program contexts (Thm. 2 from §6.5). They are verified using the matrix algebra from §B above.

Lemma 3. For any indexed set of matrices $\{i{:}\,I \bullet M_i\}$ each of dimension $F{\times}N$ and corresponding refinement matrices R_i each having F_i rows and F columns, there exists a refinement matrix R such that

$$(\ddagger i{:}\,I \bullet R_i \times M_i) \quad = \quad R \times (\ddagger i{:}\,I \bullet M_i) \ . \tag{29}$$

Proof: Refinement matrix R can be given directly as

$$(\ddagger i{:}\,I \bullet (+\!\!+ i'{:}\,I \bullet (\ R_i \textbf{ if } i = i' \textbf{ else } \mathbf{0}_{F_i \times F}\ ))\ )\ .$$

That R is a refinement matrix (i.e. it has non-negative entries and is column-one-summing) follows from its definition and the fact that each R_i is a refinement matrix. It can be established by matrix multiplication that (29) holds. [33] □

Lemma 4. *Additive monotonicity of hyper-distributions* For probability $p{:}\,[0,1]$, and hyper-distributions $\Delta_{S_1}, \Delta_{S_2}, \Delta_{I_1}, \Delta_{I_2}$ we have that $\Delta_{S_i} \sqsubseteq \Delta_{I_i}$ implies

$$\Delta_{S_1}\ {}_p\!\oplus \Delta_{S_2} \quad \sqsubseteq \quad \Delta_{I_1}\ {}_p\!\oplus \Delta_{I_2}\ .$$

Proof: From (28) it is enough for each v to find a refinement matrix R such that

$$R \times (\!(\Delta_{S_1}\ {}_p\!\oplus \Delta_{S_2})\!) \times \mathbb{1}_v \quad \approx \quad (\!(\Delta_{I_1}\ {}_p\!\oplus \Delta_{I_2})\!) \times \mathbb{1}_v\ .$$

We have:

$$\begin{array}{ll}
& (\!(\ \Delta_{I_1}\ {}_p\!\oplus \Delta_{I_2}\ )\!) \times \mathbb{1}_v \\
\approx & (\ p{*}(\!(\Delta_{I_1})\!)\ \ddagger\ (1{-}p){*}(\!(\Delta_{I_2})\!)\ ) \times \mathbb{1}_v \qquad \text{“from (21)”} \\
= & p{*}(\!(\Delta_{I_1})\!) \times \mathbb{1}_v\ \ddagger\ (1{-}p){*}(\!(\Delta_{I_2})\!) \times \mathbb{1}_v \qquad \text{“distribute post-multiplication”} \\
\\
\approx & \text{“}\Delta_{S_i} \sqsubseteq \Delta_{I_i} \text{ implies } (\!(\Delta_{I_i})\!) \times \mathbb{1}_v \approx R_i \times (\!(\Delta_{S_i})\!) \times \mathbb{1}_v \text{ for some refinement} \\
& p * R_1 \times (\!(\Delta_{S_1})\!) \times \mathbb{1}_v\ \ddagger\ (1{-}p) * R_2 \times (\!(\Delta_{S_2})\!) \times \mathbb{1}_v \qquad \text{matrix } R_i\text{”} \\
\\
= & \text{“commute scalar multiplication; distribute post-multiplication”} \\
& (\ R_1 \times p{*}(\!(\Delta_{S_1})\!)\ \ddagger\ R_2 \times (1{-}p){*}(\!(\Delta_{S_2})\!)\ ) \times \mathbb{1}_v \\
\\
= & \text{“Lem. 3 for some refinement matrix } R\text{”} \\
& R \times (\ p * (\!(\Delta_{S_1})\!)\ \ddagger\ (1{-}p) * (\!(\Delta_{S_2})\!)\ ) \times \mathbb{1}_v \\
\approx & R \times (\!(\ \Delta_{S_1}\ {}_p\!\oplus \Delta_{S_2}\ )\!) \times \mathbb{1}_v\ . \qquad \text{“from (21)”}
\end{array}$$

□

Lemma 5. *Pointwise monotonicity* For all program texts P and hyper-distributions Δ_S and Δ_I such that $\Delta_S \sqsubseteq \Delta_I$, we have

$$(\odot\,(v,\delta){:}\,\Delta_S \bullet [\![P]\!].(v,\delta)) \quad \sqsubseteq \quad (\odot\,(v,\delta){:}\,\Delta_I \bullet [\![P]\!].(v,\delta))\ .$$

Proof: Let $\{i{:}\,I \bullet M_i\}$ be a set of $N{\times}N$ matrices giving the normal form for $[\![P]\!]$ as at (23) above, so that for any (v,δ) we have

$$(\!(\ [\![P]\!].(v,\delta)\ )\!) \quad \approx \quad (\ddagger i{:}\,I \bullet (\!(v,\delta)\!) \times M_i)\ .$$

From (28) and (24), it is enough to show that for each $v' \in \mathcal{V}$ there exists a refinement matrix R such that $R \times (\ddagger i{:}\,I \bullet (\!(\Delta_S)\!) \times M_i) \times \mathbb{1}_{v'} \approx (\ddagger i{:}\,I \bullet M_i \times (\!(\Delta_I)\!)) \times \mathbb{1}_{v'}$. We have

[33] A sketch of the block matrices helps to see the pattern.

$$
\begin{array}{lll}
 & (\fatsemi i{:}\, I \bullet (\!(\Delta_I)\!) \times M_i) \times \mathbf{1}_{v'} & \\
\approx & (\fatsemi i{:}\, I \bullet (\fatsemi v{:}\, \mathcal{V} \bullet (\!(\Delta_I)\!) \times \mathbf{1}_v) \times M_i) \times \mathbf{1}_{v'} & \text{“}(\!(\Delta_I)\!) \text{ is } \mathcal{V}\text{-unique”} \\
= & (\fatsemi i{:}\, I; v{:}\, \mathcal{V} \bullet (\!(\Delta_I)\!) \times \mathbf{1}_v \times M_i) \times \mathbf{1}_{v'} & \text{“distribute post-multiplication”} \\
\\
\approx & & \text{“}\Delta_S \sqsubseteq \Delta_I \text{ implies } (\!(\Delta_I)\!) \times \mathbf{1}_v \approx R_v \times (\!(\Delta_S)\!) \times \mathbf{1}_v \text{ for some} \\
 & (\fatsemi i{:}\, I; v{:}\, \mathcal{V} \bullet R_v \times (\!(\Delta_S)\!) \times \mathbf{1}_v \times M_i) \times \mathbf{1}_{v'} & \text{refinement matrix } R_v\text{”} \\
\\
= & R \times (\fatsemi i{:}\, I; v{:}\, \mathcal{V} \bullet (\!(\Delta_S)\!) \times \mathbf{1}_v \times M_i) \times \mathbf{1}_{v'} & \text{“Lem. 3} \\
 & & \text{for some refinement matrix } R\text{”} \\
\\
\approx & R \times (\fatsemi i{:}\, I \bullet (\!(\Delta_S)\!) \times M_i) \times \mathbf{1}_{v'}\ . & \text{“distribute post-multiplication;} \\
 & & (\!(\Delta_S)\!) \text{ is } \mathcal{V}\text{-unique”}
\end{array}
$$

□

Monotonicity of secure programs w.r.t. ($\sqsubseteq$) Using Lem. 4 and Lem. 5 we now prove Thm. 2 from §6.5. We must show that if $S \sqsubseteq I$ then $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$ for all contexts $\mathcal{C}$ built from programs as defined in Fig. 1.

We use structural induction. For the base case, context $\mathcal{C}(S) := S$ is trivially monotonic.

General probabilistic choice (and hence probabilistic and conditional choice) is trivially monotonic in either argument from monotonicity of addition over hyper-distributions (Lem. 4). For example, for monotonicity in the first argument we have

$$
\begin{array}{lll}
 & [\![S \;{}_{q.\mathsf{v.h}}\!\oplus R]\!].(v,\delta) & \\
= & & \text{“Let } q_\delta := (\odot\, h{:}\, \delta \bullet q.v.h)\text{; General choice from Fig. 1”} \\
 & ([\![S]\!].(v, \{\!\{h{:}\, \delta \mid q.v.h\}\!\})\ {}_{q_\delta}\!\oplus [\![R]\!].(v, \{\!\{h{:}\, \delta \mid 1{-}q.v.h\}\!\})) & \\
\\
\sqsubseteq & ([\![I]\!].(v, \{\!\{h{:}\, \delta \mid q.v.h\}\!\})\ {}_{q_\delta}\!\oplus [\![R]\!].(v, \{\!\{h{:}\, \delta \mid 1{-}q.v.h\}\!\})) & \text{“}S \sqsubseteq I\text{; Lem. 4”} \\
= & [\![I \;{}_{q.\mathsf{v.h}}\!\oplus R]\!].(v,\delta)\ . & \text{“General choice from Fig. 1”}
\end{array}
$$

To show monotonicity of sequential composition in its right-hand argument we have for any programs R and $S \sqsubseteq I$ and initial state (v, δ) that

$$
\begin{array}{lll}
 & [\![R; S]\!].(v,\delta) & \\
= & (\odot\, (v',\delta'){:}\, [\![R]\!].(v,\delta) \bullet [\![S]\!].(v',\delta')) & \text{“Composition from Fig. 1”} \\
\sqsubseteq & (\odot\, (v',\delta'){:}\, [\![R]\!].(v,\delta) \bullet [\![I]\!].(v',\delta')) & \text{“}S \sqsubseteq I\text{; Lem. 4”} \\
= & [\![R; I]\!].(v,\delta)\ . & \text{“Composition from Fig. 1”}
\end{array}
$$

For monotonicity in the first argument we have

$$
\begin{array}{lll}
 & [\![S; R]\!].(v,\delta) & \\
= & (\odot\, (v',\delta'){:}\, [\![S]\!].(v,\delta) \bullet [\![R]\!].(v',\delta')) & \text{“Composition from Fig. 1”} \\
\sqsubseteq & (\odot\, (v',\delta'){:}\, [\![I]\!].(v,\delta) \bullet [\![R]\!].(v',\delta')) & \text{“}S \sqsubseteq I \text{ and Lem. 5”} \\
= & [\![I; R]\!].(v,\delta)\ . & \text{“Composition from Fig. 1”}
\end{array}
$$

D Example of completeness construction

Here we illustrate the completeness proof set out in §7.3 by applying it to the example of §6, where we claimed that $P_4 \not\sqsubseteq P_2$. We use §7.3 to find a C such that indeed $P_4;C \not\preceq P_2;C$.

Our v' is 1, since that is where we find the difference between P_2 and P_4 in the residual uncertainties of h; with that, we extract the fractions

$$\text{when } v'{=}1 \quad \left\{ \begin{array}{lcl} \Pi_{P_4} & := & \langle \{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{4}} \}\!\} \rangle \\ \Pi_{P_2} & := & \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle \end{array} \right.$$

and find that there are two values of h, two fractions in Π_{P_4} and three fractions in Π_{P_2}. Accordingly we set N to 3 and include an extra column for $\mathsf{h}{=}2$ and an extra, zero fraction in Π_{P_4}. Note that $\sum \Pi_{P_2} = \sum \Pi_{P_4}$ and that the total weight (of each) is 2/3.

The $N{\times}N$ matrix corresponding to Π_{P_2} is then as at right with its columns corresponding to values $1,2,3$ of h and the rows to Π_{P_2}'s three fractions. The point obtained by concatenating the rows is $(1/6,0,0,1/6,0,1/6,0,0,1/6)$, and is in 9-dimensional space; but to avoid a proliferation of fractions, we scale everything up from now on by a factor of 12, and so take x_{P_2} to be the point $(2,0,0,2,0,2,0,0,2)$.

$$\begin{pmatrix} 1/6 & 0 & 0 \\ 1/6 & 0 & 1/6 \\ 0 & 0 & 1/6 \end{pmatrix}$$

Now the scaled-up (and extended) matrix corresponding to Π_{P_4}, with a selection of refinement-forming matrices M in $\mathcal{M}_N$, is given by

$$\begin{pmatrix} 3&0&1 \\ 1&0&3 \\ 0&0&0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1&1&1 \\ 0&0&0 \\ 0&0&0 \end{pmatrix}, \begin{pmatrix} 1&0&1 \\ 0&1&0 \\ 0&0&0 \end{pmatrix}, \begin{pmatrix} 1&0&1 \\ 0&0&0 \\ 0&1&0 \end{pmatrix}, \begin{pmatrix} 0&1&1 \\ 1&0&0 \\ 0&0&0 \end{pmatrix} \cdots$$

Carrying out the matrix multiplications gives us these four possible refinements of Π_{P_4}:

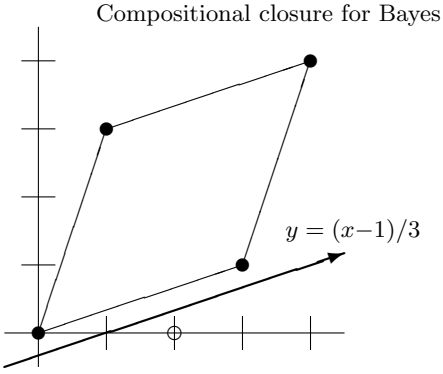
$$\begin{pmatrix} 4&0&4 \\ 0&0&0 \\ 0&0&0 \end{pmatrix}, \begin{pmatrix} 3&0&1 \\ 1&0&3 \\ 0&0&0 \end{pmatrix}, \begin{pmatrix} 3&0&1 \\ 0&0&0 \\ 1&0&3 \end{pmatrix}, \begin{pmatrix} 1&0&3 \\ 3&0&1 \\ 0&0&0 \end{pmatrix} \cdots \ .$$

Doing all of them for M in $\mathcal{M}_N$, and concatenating their rows to make points in 9-dimensional space, gives us this collection of refinements altogether:

$$\begin{array}{l} \{\ (4,0,4,0,0,0,0,0,0)\ ,\ (3,0,1,1,0,3,0,0,0)\ ,\ (3,0,1,0,0,0,1,0,3)\ , \\ \ \ (1,0,3,3,0,1,0,0,0)\ ,\ (0,0,0,4,0,4,0,0,0)\ ,\ (0,0,0,3,0,1,1,0,3)\ , \\ \ \ (1,0,3,0,0,0,3,0,1)\ ,\ (0,0,0,1,0,3,3,0,1)\ ,\ (0,0,0,0,0,0,4,0,4)\ \ \}\ . \end{array} \tag{30}$$

Our claim that $P_4 \not\sqsubseteq P_2$ suggests that the point x_{P_2} (corresponding to the matrix derived from Π_{P_2}) should not lie in the convex closure of the points (30) above.

We can see this easily by concentrating on the first and third dimensions only: for Π_2 we get $(2,0)$; and for Π_{P_4}, that is (30), after removing duplicates we get $(4,4)$, $(3,1)$, $(1,3)$ and $(0,0)$. The Π_{P_2}-point x_{P_2} is not in the convex closure of the other four because all of them have their two coordinates both positive or both zero, a property preserved by any convex combination but not shared by $(2,0)$.



We insert a hyperplane (just a line, in 2-space) midway between the separated point and the convex shape, parallel to the boundary of the latter.

Fig. 4. Finding a separating hyperplane $y = (x-1)/3$ in 2-space.

Now that we can concentrate on just two dimensions, it's easy to find a separating hyperplane with a picture. Fig. 4 shows the Π_{P_2}-point as an open circle at $(2,0)$, while the filled circles give the vertices of the diamond-shaped convex closure corresponding to refinements of Π_{P_4}. Clearly the (degenerate) hyperplane $y = (x-1)/3$ separates the point from the diamond. The normal of that hyperplane (up and left, perpendicular to the line) has direction $(-1,3)$, and when we fill in the other seven dimensions as zero –since they're not needed for separation– that gives us a candidate normal of $X{:=}(-1,0,3,0,0,0,0,0,0)$ in 9-space. By translating X to matrix form and transposing it, we can then give a tentatitive definition of D as shown at right. However this is not quite our final value for it.

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}$$

The dot-product of X with Π_{P_2}, that is $\mathsf{tr}.(\Pi_{P_2} \times D)$, turns out to be -2; and with the refinements of Π_{P_4} shown at (30) we get the dot-products 8 and 0 (multiple times), showing indeed the separation we expect, but in the wrong direction: the values 0 and 8 for P_4 are both strictly greater than the value -2 for P_2, and we want them to be strictly less. Accordingly we multiply the tentative D above by -1 [34] and then add 3 to all its elements to make them non-negative; finally we divide everything by 10 to make its rows sum to no more than 1. To get the final D from this we must then add a new "zero-th column" to make each row one-summing exactly. That gives

$$D := \begin{pmatrix} \overset{\downarrow \text{Zero'th column, for one-summing}}{0} & 0.4 & 0.3 & 0.3 \\ 0.1 & 0.3 & 0.3 & 0.3 \\ 0.4 & 0 & 0.3 & 0.3 \end{pmatrix} .$$

The distinguishing context $(-\,;C)$, say, must then overwrite h according to the distribution given by a row of D, the one selected by the value $\hat{h}$ of h incoming to C; thus we construct C to be

$$\begin{array}{l} \textbf{if } \mathsf{v}{=}1 \textbf{ then } \ \mathsf{h}{:\in}\, (\{\!|1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}|\!\} \textbf{ if } \mathsf{h}{=}1 \textbf{ else } \{\!|0^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}|\!\}) \\ \qquad \textbf{else } \ \mathsf{h}{:=}\, 0 \textbf{ fi} \ , \end{array}$$

[34] The fact we can simply multiply by -1 to reverse the sense of the comparison does not mean we can just as easily construct a context to show $P_2 \sqsubseteq P_4$ — which would indeed be a worry. In fact for $P_2 \not\sqsubseteq P_4$ we'd need a shape X_{P_2} and a point x_{P_4}; and then we would find x_{P_4} *inside* the shape X_{P_2}, thus unable to be separated from it.

with the outer **if** effectively restricting our attack to occur only when $v'{=}1$. (That allows us to ignore the $\mathsf{h}{=}2$ case in D, as well.) Thus we have our context $(-\,;C)$. Let us now check that it actually works.

We begin with $P_4;C$. Its output hyper-distribution is (after some calculation) given by

$$\begin{array}{l}\{\!\!\{\ (0, \{\!\!\{0\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{0^{@\,0.1}, 1^{@\,0.3}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{0^{@\,0.3}, 1^{@\,0.1}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{3}}\ \ \}\!\!\}\ ,\end{array}$$

whose Bayes Vulnerability is $1/3{*}1 + 1/3{*}0.3 + 1/3{*}0.3 \approx 0.53$. On the other hand, for $P_2;C$ the output hyper-distribution is

$$\begin{array}{l}\{\!\!\{\ (0, \{\!\!\{0\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{6}}\ ,\\ \quad (1, \{\!\!\{0^{@\,0.2}, 1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{0^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{6}}\ \ \}\!\!\}\ ,\end{array}$$

and here the vulnerability is $1/3{*}1 + 1/6{*}0.4 + 1/3{*}0.3 + 1/6{*}0.3 \approx 0.55$. Note that in the third summand we took 0.3 rather than the larger 0.4 associated with 0, since as part of our construction we exclude guesses that h is 0.[35]

Thus we have established that $P_4;C \not\simeq P_2;C$ (for the adjusted C — see Footnote 35), because the vulnerability of the former is 0.53 but for the latter the vulnerability is the greater 0.55. Hence when our refinement relation insists that $P_4 \not\sqsubseteq P_2$ –as we argued earlier above– in fact it is not being too severe, but rather it is acting just as a compositional closure should. It protects us not only against the context C we just made, but all other contexts too — in spite of the fact that in isolation P_4 and P_2 are not distinguished by elementary testing.

Finally, in this example there are many hyperplanes with distinct normals that achieve the separation we need, and each of these may be used to construct different distinguishing contexts. For example, since there exists a separating hyperplane with normal $(0,0,2,1,0,0,1,0,0)$ we can use it to define another

[35] Dealing with this detail would split the 0-case in half, uniformly distributed over $-1,-2$, since the resulting probability $0.4/2$ for each would then be small enough that a Bayes-Vulnerability attack would never choose it. The adjusted context C' would contain

$$\mathsf{h}{:\in}\,(\{\!\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\}\ \textbf{if}\ \mathsf{h}{=}1\ \textbf{else}\ \{\!\!\{-2^{@\,0.2}, -1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})$$

and the resulting output for $P_2;C'$ would be

$$\begin{array}{l}\{\!\!\{\ (0, \{\!\!\{0\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{6}}\ ,\\ \quad (1, \{\!\!\{-2^{@\,0.1}, -1^{@\,0.1}, 1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\!\{-2^{@\,0.2}, -1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\!\})^{@\frac{1}{6}}\ \ \}\!\!\}\ ,\end{array}$$

in which neither -2 nor -1 would ever be chosen for a Bayes-Vulnerability attack.

distribution matrix

$$D := \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0.5 \end{pmatrix}$$

from which we can specify the distinguishing context $(-\,;C)$, where C is

$$\begin{array}{l} \textbf{if } \mathsf{v}{=}1 \textbf{ then} \\ \quad \mathsf{h}{:\in}\ (\,\{\!\{1^{@\frac{1}{2}}, 2^{@\frac{1}{4}}, 3^{@\frac{1}{4}}\}\!\}\ \textbf{ if } \mathsf{h}{=}1 \textbf{ else }\ \{\!\{2^{@\frac{1}{2}}, 3^{@\frac{1}{2}}\}\!\}\,) \\ \textbf{else } \mathsf{h}{:=}\,1 \textbf{ fi}\ , \end{array} \tag{31}$$

which requires no $\mathsf{h}{:=}\,0$ case for D since the rows of its defining normal just happen to have the same sum. (That's not true for the middle row; but as before we can ignore it since, in the $v'{=}1$ case we are considering, that row is never used.) [36]

Finding the normal that generates (31), however, is harder if done geometrically: it turns out that we would have had to specialise to three coordinate indices 3, 4 and 7 rather than just 1 and 3. The resulting inspection –to see just where to slip the hyperplane in between– would then have had to be done in three- rather than two dimensions, as Fig. 5 illustrates (in a side view). In general such hyperplanes can of course be found, without drawing pictures, by using constraint solvers to deal with the linear inequalities symbolically.

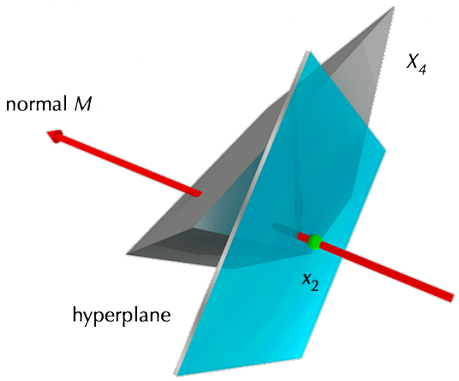


Fig. 5. Finding a separating hyperplane, with (2,1,1) as normal, in 3 of the 9 dimensions.

E Proof of the Atomicity Lemma

To prove the atomicity distribution lemma (Lem. 1 from §8.1) we use the matrix algebra from §B.

[36] It can be shown that this is a legitimate counter-example by using (4) and (31) to calculate the partitions

$$v'{=}1 \left\{ \begin{array}{ll} \Pi_{P_2;C}\colon & \langle\{\!\{1^{@\frac{1}{12}}, 2^{@\frac{1}{24}}, 3^{@\frac{1}{24}}\}\!\}, \{\!\{1^{@\frac{1}{12}}, 2^{@\frac{1}{8}}, 3^{@\frac{1}{8}}\}\!\}, \{\!\{2^{@\frac{1}{12}}, 3^{@\frac{1}{12}}\}\!\}\rangle \\ \Pi_{P_4;C}\colon & \langle\{\!\{1^{@\frac{1}{8}}, 2^{@\frac{5}{48}}, 3^{@\frac{5}{48}}\}\!\}, \{\!\{1^{@\frac{1}{24}}, 2^{@\frac{7}{48}}, 1^{@\frac{7}{48}}\}\!\}\rangle\ . \end{array} \right.$$

and observing that the vulnerability of $\Pi_{P_4;C}$ is 1/8+7/48 = 13/48, which is just smaller than the vulnerability of $\Pi_{P_2;C}$ at 1/12+1/8+1/12 = 7/24.

Suppose we have matrix representations $(\!|P_{\{1,2\}}|\!)$ for the classical program texts P_1 and P_2 and a row-matrix representation $((v,\delta))$ of an incoming split-state.

If from every initial and final v-state of $P_1;P_2$ it is possible to determine the intermediate value of v (after P_1 and before P_2) then there must exist a total function $f\colon \mathcal{V}\to\mathcal{V}\to\mathcal{V}$ such that for all v, v' we have

$$1_v \times (\!|P_1|\!) \times (\!|P_2|\!) \times 1_{v'} \quad = \quad 1_v \times (\!|P_1|\!) \times 1_{f.v.v'} \times (\!|P_2|\!) \times 1_{v'} \tag{32}$$

from which we have for all $\hat{v} \neq f.v.v'$ that

$$1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'} \quad = \quad \mathbf{0}_N \ , \tag{33}$$

where $\mathbf{0}_N$ is the $N\times N$ matrix of zeros.

Assuming such an f with properties (32) and (33), we can calculate

$$\begin{array}{lll}
& \big(\!\big|\, [\![\langle\!\langle P_1; P_2 \rangle\!\rangle]\!].(v,\delta) \,\big|\!\big) & \\
\approx & (\pm v'\colon \mathcal{V} \bullet ((v,\delta)) \times (\!|P_1; P_2|\!) \times 1_{v'}) & \text{“embedding”} \\
= & (\pm v'\colon \mathcal{V} \bullet ((v,\delta)) \times (\!|P_1|\!) \times (\!|P_2|\!) \times 1_{v'}) & \text{“classical composition”} \\
= & (\pm v'\colon \mathcal{V} \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times (\!|P_2|\!) \times 1_{v'}) & \text{“}((v,\delta))\times 1_v = ((v,\delta))\text{”} \\
= & (\pm v'\colon \mathcal{V} \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{f.v.v'} \times (\!|P_2|\!) \times 1_{v'}) & \text{“(32)”} \\
\\
= & (\pm v', \hat{v}\colon \mathcal{V} \mid \hat{v}{=}f.v.v' \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \text{“one-point rule for } (\pm)\text{”} \\
\\
\approx & & \text{“}\mathbf{0}_{1\times N} \text{ is unit of concatenation, up to similarity”} \\
& (\pm v', \hat{v}\colon \mathcal{V} \mid \hat{v}{=}f.v.v' \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \\
\pm & (\pm v', \hat{v}\colon \mathcal{V} \mid \hat{v}{\neq}f.v.v' \bullet \mathbf{0}_{1\times N}) & \\
\\
= & & \text{“}\mathbf{0}_{1\times N} = ((v,\delta))\times \mathbf{0}_N,\ (33)\text{”} \\
& (\pm v', \hat{v}\colon \mathcal{V} \mid \hat{v}{=}f.v.v' \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \\
\pm & (\pm v', \hat{v}\colon \mathcal{V} \mid \hat{v}{\neq}f.v.v' \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \\
\\
\approx & & \text{“}\hat{v}{=}f.v.v' \text{ and } \hat{v}{\neq}f.v.v' \text{ are disjoint and exhaustive;} \\
& & (\pm) \text{ is commutative and associative up to similarity”} \\
& (\pm v', \hat{v}\colon \mathcal{V} \bullet ((v,\delta)) \times 1_v \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \\
\\
= & (\pm v', \hat{v}\colon \mathcal{V} \bullet ((v,\delta)) \times (\!|P_1|\!) \times 1_{\hat{v}} \times (\!|P_2|\!) \times 1_{v'}) & \text{“}((v,\delta))\times 1_v = ((v,\delta))\text{”} \\
= & (\pm v'\colon \mathcal{V} \bullet (\pm \hat{v} \bullet ((v,\delta)) \times (\!|P_1|\!) \times 1_{\hat{v}}) \times (\!|P_2|\!) \times 1_{v'}) & \text{“distribute } \pm\text{”} \\
\approx & \big(\!\big|\, [\![\langle\!\langle P_1 \rangle\!\rangle ; \langle\!\langle P_2 \rangle\!\rangle]\!].(v,\delta) \,\big|\!\big) \ , & \text{“composition, embedding”}
\end{array}$$

whence our result follows because $((\cdot))$ is injective up to similarity and (v,δ) was arbitrary.

F Informal description of the Oblivious Transfer implementation [37]

Given are two agents B, C; Agent B has two messsages $\mathsf{m}_{\{0,1\}}$, bit-strings of the same length; Agent C has a message variable m and a choice $\mathsf{c}\colon\{0,1\}$ of which of $\mathsf{m}_{\{0,1\}}$ is to be assigned to m. The specification is thus

$$\begin{array}{c}\mathbf{vis}_B\ \mathsf{m}_0, \mathsf{m}_1;\ \mathbf{vis}_C\ \mathsf{m}, \mathsf{c};\\ \mathsf{m}{:=}\,\mathsf{m}_{\mathsf{c}}\ .\end{array}$$

Note that B does not discover c and that C does not discover $\mathsf{m}_{\neg\mathsf{c}}$.

The implementation is, informally, as follows:

This is the prelude of the protocol

1. Agent B chooses privately two random bit-strings $\mathsf{m}'_{\{0,1\}}$ to be used for ∇-encrypting $\mathsf{m}_{\{0,1\}}$ respectively.
2. Agent C chooses privately in $\mathsf{c}'\colon\{0,1\}$ which of the encrypting strings $\mathsf{m}'_{\{0,1\}}$ will be revealed to her.
3. A trusted third party collects both $\mathsf{m}'_{\{0,1\}}$ from B, collects c' from C and then reveals (only) $\mathsf{m}'_{c'}$ to C. She throws $\mathsf{m}'_{\neg c'}$ away, and then leaves.

From here is the main part of the protocol

4. Agent C then tells B to encrypt and send messages in the following way:

   (a) If Agent C wants m_0 and has m'_0, then she instructs B to send her both $\mathsf{m}_0 \nabla \mathsf{m}'_0$ and $\mathsf{m}_1 \nabla \mathsf{m}'_1$. Because she has m'_0 she can recover m_0 via $(\mathsf{m}_0 \nabla \mathsf{m}'_0) \nabla \mathsf{m}'_0$; but she cannot recover m_1.
   (b) Similarly, if Agent C wants m_0 but has m'_1 instead (of m'_0), then she simply instructs B to send her both $\mathsf{m}_0 \nabla \mathsf{m}'_1$ and $\mathsf{m}_1 \nabla \mathsf{m}'_0$, i.e. with the encryption the other way around.
   (c) If Agent C wants m_1 and has m'_0 — as for (4b).
   (d) If Agent C wants m_1 and has m'_1 — as for (4a).

   The four cases (4a–4d) can be described succinctly –if cryptically– simply by instructing B to send $\mathsf{m}_i \nabla \mathsf{m}'_{i \nabla \mathsf{c} \nabla \mathsf{c}'}$ for $i = 0, 1$.

[37] An even more informal description is this fairy tale. An Apprentice magician is about to graduate, and he must now choose between black- or white magic. His Sorcerer will allow him to read either the *Black Tome* or the *White Tome*, not both; and his choice must be his own, uncoerced, thus never revealed to the Academy.

The Sorcerer summons a *trusted third party* Djinn who gives him two locks, one black and one white; and the Djinn gives a single, golden key to the Apprentice. On the key is a small dot that only the Apprentice can see: it is the colour of the matching lock. The Djinn then returns to his own dimension.

The Apprentice tells the Sorcerer to match the lock colours to the Tomes, or to reverse them: it depends on whether his choice matches the colour of the dot. The Sorcerer then leaves; the Apprentice can unlock only the Tome of his choice; and –provided he locks it again– no one afterwards will know which one he read.

Note that only Step (3) involves private messages (first between B and the third party, and then between the third party and C), and that is only in the prelude, before any of the actual data $\mathsf{m}_{\{0,1\}}, \mathsf{c}$ has appeared. Steps (1) and (2) involve no messages at all; and the messages occurring in Step (4) are ∇-encrypted already. In effect the prelude has created a one-time pad.

A formal derivation of this implementation is given elsewhere [27].

G Alternative uncertainty measures

G.1 Shannon Entropy

The *Shannon Entropy* of a (full) distribution $\delta{:}\,\mathsf{D}\mathcal{X}$ is $\mathsf{H}.\delta := (\odot\, d{:}\,\delta \bullet -\lg(\delta.d))$, that is the weighted average of the negated base-2 logarithms of its constituent probabilities [33]. By extension, for any hyper-distribution Δ we define the *conditional* Shannon Entropy $\mathsf{H}.\Delta$ to be $(\odot\,(v,\delta){:}\,\Delta \bullet \mathsf{H}.\delta)$, the expected value of the entropies of its support [3].

Going further, if we split up our hyper-distribution by v into its partitions, we have an equivalent presentation of entropy as the sum of individual partition-entropies $\mathsf{H}.\Delta = \big(\sum v{:}\,\mathcal{V} \bullet \mathsf{H}.(\mathsf{fracs}.\Delta.v)\big)$, provided we define the entropy of a single partition, and of a single fraction, as follows:

$$\begin{array}{rl} \mathsf{H}.\Pi := & (\sum \pi{:}\,\Pi \bullet \mathsf{H}.\pi) \\ \mathsf{H}.\pi := & (\odot\, d{:}\,\pi \bullet \overline{\lg}([\pi].d)) \ , \end{array} \tag{34}$$

where we write $\overline{\lg}$ for $-\lg$ to avoid a proliferation of minus signs, and $[\pi]$ is normalisation of the fraction π, scaling it up (if necessary) to give a distribution again.

The ordering $(\preceq_{\mathsf{H}})$ based on hyper-distributions

$$\Delta_S \preceq_{\mathsf{H}} \Delta_I \quad := \quad (\mathsf{ft}.\Delta_S = \mathsf{ft}.\Delta_I) \wedge (\mathsf{H}.\Delta_S \leq \mathsf{H}.\Delta_I)$$

is then specified, as for the Bayes order $(\preceq)$, so that $\Delta_S \preceq_{\mathsf{H}} \Delta_I$ if they are functionally equivalent and the uncertainty (the Shannon Entropy in this case) of Δ_I is no less than that of Δ_S. It extends pointwise to secure programs. Furthermore we write that $S \prec_{\mathsf{H}} I$ when $S \preceq_{\mathsf{H}} I$ but $I \not\preceq_{\mathsf{H}} S$.

Non-compositionality Consider again two functionally-equivalent programs from our three-box puzzle example from §2 and §4:

$$\begin{array}{rl} S := & \mathsf{h}{:=}\,0 \oplus 1 \oplus 2;\ \mathsf{v}{:\in}\,\{\!\{ w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}} \}\!\};\qquad \mathsf{v}{:=}\,\bot \\ I_2 := & \mathsf{h}{:=}\,0 \oplus 1 \oplus 2;\ \mathsf{v}{:\in}\,\{\!\{ w^{@(\mathsf{h}\div 2)}, b^{@1-(\mathsf{h}\div 2)} \}\!\};\ \mathsf{v}{:=}\,\bot \end{array}$$

with final hyper-distributions

$$\{\!\{\ (\bot, \{\!\{ 1^{@\frac{1}{3}}, 2^{@\frac{2}{3}} \}\!\}),\ (\bot, \{\!\{ 0^{@\frac{2}{3}}, 1^{@\frac{1}{3}} \}\!\})\ \}\!\} \tag{Δ'_S}$$

and

$$\{\!\{\ (\bot, \{\!\{2\}\!\})^{@\frac{1}{3}}\ ,\ (\bot, \{\!\{0,1\}\!\})^{@\frac{2}{3}}\ \}\!\}\ . \tag{Δ'_{I_2}}$$

The Shannon entropy of Δ'_S, calculated $2 * \frac{1}{2}(\frac{1}{3}\overline{\lg}\frac{1}{3} + \frac{2}{3}\overline{\lg}\frac{2}{3})$, is slightly more than 0.918, exceeding the entropy Δ'_{I_2} that, by the simpler calculation given by $\frac{1}{3}(\overline{\lg}\,1) + \frac{2}{3}(2{*}\frac{1}{2}\overline{\lg}\frac{1}{2})$, turns out to be exactly $\frac{2}{3}$; and so $I_2 \preceq_{\mathsf{H}} S$.

However if we define context $\mathcal{C}$ to be $(-\,;\, \mathsf{h}{:=}(1\ \textbf{if}\ \mathsf{h}{=}2\ \textbf{else}\ \mathsf{h}))$ then the entropy of $\mathcal{C}(I_2)$ is the same as before at $\frac{2}{3}$; but the entropy of $\mathcal{C}(S)$ is now only a half of what it was, at $\approx$0.459. Hence $\mathcal{C}(I_2) \not\preceq_{\mathsf{H}} \mathcal{C}(S)$.

Soundness We follow initially the structure of the soundness proof for Bayes Risk. Fix an initial split-state and construct the output hyper-distributions $\Delta'_{\{S,I\}}$ that result from S, I respectively. Then since we assume $S \sqsubseteq I$ we must have $\Delta'_S \sqsubseteq \Delta'_I$. We now show that this implies $\Delta'_S \preceq_{\mathsf{H}} \Delta'_I$.

Since $S \sqsubseteq I$ trivially guarantees that $\mathsf{ft}.\Delta'_S{=}\mathsf{ft}.\Delta'_I$, we need to show that the Shannon Entropy condition in $(\preceq_{\mathsf{H}})$ is satisfied. Since we have that $\mathsf{H}.\Delta'$ is $(\sum v'{:}\mathcal{V} \bullet \mathsf{H}.(\mathsf{fracs}.\Delta.v'))$, it is enough to show that for each $v'{:}\mathcal{V}$ the entropy of $\Pi'_S{:=}\mathsf{fracs}.\Delta'_S.v'$ is no less than the entropy of $\Pi'_I{:=}\mathsf{fracs}.\Delta'_I.v'$, provided that $\Pi'_S \approx \Pi' \sqsubseteq\!\!\!\sim \Pi'_I$ for some partition Π' depending on v'.

For $\Pi'_S{\approx}\Pi'$ we consider the unique Π that is the reduction of both: it is formed in each case by adding together groups of similar fractions. From (34) and arithmetic, we obtain immediately that $\mathsf{H}.\Pi'_S = \mathsf{H}.\Pi = \mathsf{H}.\Pi'$. [38]

For $\Pi' \sqsubseteq\!\!\!\sim \Pi'_I$ we know that the fractions of Π'_I are sums of groups of not-necessarily-similar fractions in Π'. We consider the special case of just two fractions $\pi_{\{1,2\}}$ in Π' summing to a single fraction $\pi{:=}\pi_1{+}\pi_2$ in Π'_I, and look at their relative contributions to the sum (34); we have

$$
\begin{array}{lll}
 & \mathsf{H}.\pi & \\
= & \mathsf{H}.(\pi_1{+}\pi_2) & \\
= & (\odot\, d{:}\,(\pi_1{+}\pi_2) \bullet \overline{\lg}([\pi_1{+}\pi_2].d)) & \\
= & (\odot\, d{:}\,\pi_1 \bullet \overline{\lg}([\pi_1{+}\pi_2].d)) + (\odot\, d{:}\,\pi_2 \bullet \overline{\lg}([\pi_1{+}\pi_2].d)) & \\
\dagger\ \geq & (\odot\, d{:}\,\pi_1 \bullet \overline{\lg}([\pi_1].d)) + (\odot\, d{:}\,\pi_2 \bullet \overline{\lg}([\pi_2].d)) & \text{“see below”} \\
= & \mathsf{H}.\pi_1 + \mathsf{H}.\pi_2\ , &
\end{array}
$$

that is that the contribution to the conditional entropy of π on its own is at least as great as it was when was separated into $\pi_{\{1,2\}}$.

For “see below” we refer to the *Key Lemma* [36, p5] which states that for two total distributions δ, δ' of equal support, the weighted sum $(\odot\, d{:}\,\delta \bullet \overline{\lg}(\delta'.d))$ attains its minimum over δ' when $\delta{=}\delta'$.

Extending the argument similarly to multiple additions gives $\mathsf{H}.\Pi' \leq \mathsf{H}.\Pi'_I$ as required and thus we have $\mathsf{H}.\Pi'_S \leq \mathsf{H}.\Pi'_I$ overall. We note that the inequality at † is strict when $\pi_1 \not\approx \pi_2$, because then e.g. $[\pi_1]{\neq}[\pi_1{+}\pi_2]$. [39] We have established

Theorem 6. *Soundness of* $(\sqsubseteq)$ *w.r.t.* $(\preceq_{\mathsf{H}})$ For all secure programs S and I and contexts $\mathcal{C}$, we have that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq_{\mathsf{H}} \mathcal{C}(I)$. □

[38] If $\pi_1 \approx \pi_2$ then $[\pi_1]{=}[\pi_2]{=}[\pi_1{+}\pi_2]$ and so the line marked † below becomes an equality.

[39] This follows from a strengthening of the Key Lemma to “... *only* when $\delta{=}\delta'$,” which is implied by the proof of Thm. 1 [*op. cit.*] immediately before.

Finally, when $S \sqsubseteq I$ but $S \neq I$ so that $\Delta'_S \neq \Delta'_I$ for some initial split-state we must have $\Pi'_S \not\approx \Pi'_I$ for some final v', since both those partitions are in reduced form: that is, reduced partitions cannot be similar without actually being equal. Thus also $\Pi' \not\approx \Pi'_I$, and so we can find particular $\pi_1 \not\approx \pi_2$ to realise the strict inequality at †. That gives us

Lemma 6. *Strict soundness* For all hyper-distributions $\Delta_{\{1,2\}}$ we have that $\Delta_1 \sqsubseteq \Delta_2$ but $\Delta_1 \neq \Delta_2$ implies $\Delta_1 \prec_{\mathsf{H}} \Delta_2$. □

G.2 Marginal guesswork

The *Marginal guesswork* [30] of a distribution $\delta : \mathsf{D}\mathcal{X}$ is the least number of guesses an attacker requires to be sure that her chance of guessing some h chosen according to δ is at least a given probability α. We define it

$$\mathsf{W}_\alpha.\delta \quad := \quad (\sqcap i : 1..N \mid \sqcup^i \delta \geq \alpha)$$

where we write $\sqcup^i \delta$, or more generally $\sqcup^i \pi$ for fraction π to mean the sum of the i greatest probabilities in π, and N is the cardinality of $\mathcal{X}$. Note that by super-distribution of maximum over addition we have $\sqcup^i(\pi_1 + \pi_2) \leq \sqcup^i \pi_1 + \sqcup^i \pi_2$ for any i in the proper range. To avoid clutter, we will omit the range $1..N$ for i from here on.

For a hyper-distribution Δ we define

$$\begin{array}{rl} & \mathsf{W}_\alpha.\Delta := \ (\sqcap i \mid (\odot\,(v,\delta) : \Delta \bullet \sqcup^i \delta) \geq \alpha) \\ \text{or equivalently} & \mathsf{W}_\alpha.\Delta := \ (\sqcap i \mid (\sum \mathsf{v} : \mathcal{V}; \pi : \mathsf{fracs}.\Delta.v \bullet \sqcup^i \pi) \geq \alpha) \end{array} \tag{35}$$

which is the least value i such that if an attacker is allowed to make that many guesses then she can discover the value of h with probability at least α.

Observe that our definition of $\mathsf{W}_\alpha.\Delta$ is *not* the same as the *conditional marginal guesswork* $(\odot\,\delta : \Delta \bullet \mathsf{W}_\alpha.\delta)$ as conventionally defined [15]. We argue that conditional marginal guesswork *is not* a reasonable measure of the number of guesses required by an attacker to ensure that the probability of guessing h in Δ is greater than α. Consider for example the hyper-distributions

$$\Delta_S := \{\!\{(v, \{\!\{0\}\!\}), (v, \{\!\{1..4\}\!\})\}\!\} \quad \text{and} \quad \Delta_I := \{\!\{(v, \{\!\{0\}\!\} \oplus \{\!\{1..4\}\!\})\}\!\} \ .^{40}$$

Note that $\Delta_S \sqsubseteq \Delta_I$ since the latter is obtained by merging the two split-states of the former.

Now an attacker has more information about how h was chosen in Δ_S than in Δ_I: for Δ_S she knows not only that h is distributed according to the distribution $\{\!\{0\}\!\} \oplus \{\!\{1..4\}\!\}$ overall (as for Δ_I), but as well she knows when h was chosen from $\{\!\{0\}\!\}$ and when h was chosen from $\{\!\{1..4\}\!\}$. However, when we set $\alpha := 1/2$ the conditional marginal guesswork of Δ_S is $\frac{1}{2}(1) + \frac{1}{2}(4\alpha)$, that is $3/2$ — which is higher than for Δ_I, which gives only 1. This suggests that it is *harder* for an

[40] Alternatively we could write out $\{\!\{0\}\!\} \oplus \{\!\{1..4\}\!\}$ with its explicit probabilities as $\{\!\{0^{@\frac{1}{2}}, 1^{@\frac{1}{8}}, 2^{@\frac{1}{8}}, 3^{@\frac{1}{8}}, 4^{@\frac{1}{8}}\}\!\}$, but we prefer to avoid the superscripts.

attacker to guess h in Δ_S than in Δ_I, in spite of the fact that the attacker knows more about the final h-distribution in Δ_S when launching an attack.

Using our W_α we have $\mathsf{W}_{1/2}.\Delta_S = \mathsf{W}_{1/2}.\Delta_I$, that is 1 in both cases: with just one guess at her disposal an attacker is guaranteed to guess h at least half the time. Applying her one guess to Δ_S, half the time she can guess 0 and is sure to be right; in Δ_1 she guesses 0 and will be right half the time.

The ordering between hyper-distributions based on marginal guesswork is

$$\Delta_S \preceq_{\mathsf{W}_\alpha} \Delta_I \quad := \quad (\mathsf{ft}.\Delta_S = \mathsf{ft}.\Delta_I) \wedge (\mathsf{W}_\alpha.\Delta_S \leq \mathsf{W}_\alpha.\Delta_I)$$

which extends pointwise to programs.

Non-compositionality When α is not zero, marginal guesswork –like the other measures– is non-compositional for our subset of programs. For such an $\alpha{\neq}0$ take, for example, functionally equivalent programs

$$\begin{aligned} S :=\ & \mathsf{h}{:\in}\, \{\!\{0,1,2\}\!\}\ {}_\alpha\!\oplus \{\!\{-N..-1\}\!\}; \\ & \mathsf{v}{:\in}\, (\{\!\{w^{@\frac{\mathsf{h}}{2}}, b^{@(1-\frac{\mathsf{h}}{2})}\}\!\} \ \textbf{if } \mathsf{h}{\geq}0 \textbf{ else } \ \{\!\{w^{@\frac{1}{2}}, b^{@\frac{1}{2}}\}\!\}); \\ & \mathsf{v}{:=}\,\bot \end{aligned}$$

$$\begin{aligned} I :=\ & \mathsf{h}{:\in}\, \{\!\{0,1,2\}\!\}\ {}_\alpha\!\oplus \{\!\{-N..-1\}\!\}; \\ & \mathsf{v}{:\in}\, (\{\!\{w^{@\mathsf{h}\div 2}, b^{@1-\mathsf{h}\div 2}\}\!\} \ \textbf{if } \mathsf{h}{\geq}0 \textbf{ else } \ \{\!\{w^{@\frac{1}{3}}, b^{@\frac{2}{3}}\}\!\}); \\ & \mathsf{v}{:=}\,\bot \end{aligned}$$

such that if $\alpha{=}1$ then $N{=}0$ else $N \geq 3{\times}\frac{1-\alpha}{\alpha}$. These programs have the final output distributions

$$\{\!\{(\bot, \{\!\{1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}\}\!\}\ {}_\alpha\!\oplus \{\!\{-N...-1\}\!\}),\ (\bot, \{\!\{0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}\}\!\}\ {}_\alpha\!\oplus \{\!\{-N...-1\}\!\})\}\!\} \quad (\Delta'_S)$$

and $$\{\!\{(\bot, \{\!\{2\}\!\}\ {}_\alpha\!\oplus \{\!\{-N...-1\}\!\})^{@\frac{1}{3}},\ (\bot, \{\!\{0,1\}\!\}\ {}_\alpha\!\oplus \{\!\{-N...-1\}\!\})^{@\frac{2}{3}}\}\!\}\ . \quad (\Delta'_I)$$

We can calculate that both $\mathsf{W}_\alpha.\Delta'_S$ and $\mathsf{W}_\alpha.\Delta'_I$ are 2, and so $S \preceq_{\mathsf{W}_\alpha} I$, but that for context $\mathcal{C}$ defined as $(-\,;\, \mathsf{h}{:=}(\mathsf{h} \div 2 \textbf{ if } \mathsf{h}{\geq}0 \textbf{ else } \mathsf{h}))$ we have $\mathsf{W}_\alpha.\Delta'_I$ is only 1, while $\mathsf{W}_\alpha\Delta'_S$ remains at 2 — and so $\mathcal{C}(S) \not\preceq_{\mathsf{W}_\alpha} \mathcal{C}(I)$.

Soundness

Lemma 7. ($\sqsubseteq$) *implies* ($\preceq_{\mathsf{W}_\alpha}$) For all hyper-distributions Δ_S and Δ_I and probabilities α, if $\Delta_S{\sqsubseteq}\Delta_I$ then also $\Delta_S{\preceq_{\mathsf{W}_\alpha}}\Delta_I$; consequently $S{\sqsubseteq}I$ implies$S{\preceq_{\mathsf{W}_\alpha}}I$.

Proof: From (35) and the definition of refinement (Def. 2, §6.4) it is enough to show that for any partition Π and i in range that (i) if the fractions in Π are similar then $(\sum \pi{:}\,\Pi \bullet \sqcup^i\pi) = \sqcup^i(\sum \pi{:}\,\Pi)$ else (ii) $(\sum \pi{:}\,\Pi \bullet \sqcup^i\pi) \geq \sqcup^i(\sum \pi{:}\,\Pi)$. To show (ii) we have by generalising $\sqcup^i(\pi_1{+}\pi_2) \leq \sqcup^i\pi_1{+}\sqcup^i\pi_2$ that indeed

$$(\sum \pi{:}\,\Pi \bullet \sqcup^i\pi) \quad \geq \quad \sqcup^i(\sum \pi{:}\,\Pi)\ ,$$

and for (i) we can replace inequality by equality since $(\sqcup^i)$ distributes over summation in that case. □

Theorem 7. *Soundness of* $(\sqsubseteq)$ *w.r.t.* $(\preceq_{\mathsf{W}_\alpha})$ For all probabilities α, secure programs S, I and contexts $\mathcal{C}$ we have that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq_{\mathsf{W}_\alpha} \mathcal{C}(I)$.

Proof: Lem. 7 and monotonicity of $(\sqsubseteq)$ (Thm. 2 from §6.5). □

G.3 Guessing entropy

The *guessing entropy* [19] of a distribution δ is the (least) average number of guesses required to guess h in δ. It is equivalent to the average α-marginal guesswork over all values of α [30], and we define it

$$\mathsf{W}.\delta \quad := \quad (\sum i{:}\, 1..N \bullet \sqcap^i \delta) \ ,$$

where $\sqcap^i\delta$ is the sum of the i smallest probabilities in δ.[41] Note that by subdistribution of minimum over addition we have $\sqcap^i(\pi_1{+}\pi_2) \geq \sqcap^i\pi_1 + \sqcap^i\pi_2$ for any i in range. For hyper-distribution Δ we define the conditional guessing entropy, thus

$$\begin{array}{rrcl} & \mathsf{W}.\Delta & := & (\odot\,(v,\delta){:}\,\Delta \bullet \mathsf{W}.\delta) \\ \text{or equivalently} & \mathsf{W}.\Delta & := & (\sum v{:}\,\mathcal{V}; \pi{:}\,\mathsf{fracs}.\Delta.v \bullet \mathsf{W}.\pi) \ , \end{array}$$

where $\mathsf{W}.\pi$ is defined in the same way as $\mathsf{W}.\delta$. We define the ordering by

$$\Delta_S \preceq_{\mathsf{W}} \Delta_I \quad := \quad (\mathsf{ft}.\Delta_S = \mathsf{ft}.\Delta_I) \wedge (\mathsf{W}.\Delta_S \leq \mathsf{W}.\Delta_I) \ ,$$

which extends pointwise to secure programs.

Non-compositionality To show non-compositionality of ordering $(\preceq_{\mathsf{W}})$ we refer again (as we did for Shannon entropy in §G.1) to the functionally equivalent programs S and I_2. First we calculate that

$$\begin{array}{rrcll} & \mathsf{W}.\Delta'_S & = & 2{\times}\frac{1}{2}(\frac{1}{3} + (\frac{1}{3}{+}\frac{2}{3})) & = \ \frac{4}{3} \\ \text{and} & \mathsf{W}.\Delta'_{I_2} & = & \frac{1}{3}(1) + \frac{2}{3}(\frac{1}{2} + (\frac{1}{2}{+}\frac{1}{2})) & = \ \frac{4}{3} \ , \end{array}$$

so that we have $I_2 \preceq_{\mathsf{W}} S$. Again taking context $\mathcal{C}$ to be $(-\,;\, \mathsf{h}{:=}\,(1 \textbf{ if } \mathsf{h}{=}2 \textbf{ else } \mathsf{h}))$ we get that the guessing entropy of $\mathcal{C}(S)$ is reduced to $\frac{7}{6}$ while that of $\mathcal{C}(I_2)$ is still $\frac{4}{3}$, and hence $\mathcal{C}(I_2) \not\preceq_{\mathsf{W}} \mathcal{C}(S)$.

Soundness

Lemma 8. $(\sqsubseteq)$ *implies* $(\preceq_{\mathsf{W}})$ For all hyper-distributions Δ_S and Δ_I, we have that $\Delta_S{\sqsubseteq}\Delta_I$ implies $\Delta_S{\preceq_{\mathsf{W}}}\Delta_I$; and consequently $S{\sqsubseteq}I$ implies $S{\preceq_{\mathsf{W}}}I$.

Proof: As in the proof of soundness for marginal guesswork, it is enough to show that for any partition Π (i) if the fractions in Π are similar then $(\sum \pi{:}\,\Pi \bullet \mathsf{W}.\pi) = \mathsf{W}.(\sum \pi{:}\,\Pi)$ else (ii) $(\sum \pi{:}\,\Pi \bullet \mathsf{W}.\pi) \leq \mathsf{W}.(\sum \pi{:}\,\Pi)$. For (ii) we reason:

[41] If *wlog* the four probabilities a, b, c, d are ordered greatest to least, then the best strategy is to guess (the value associated with) a first, and then to go on to guess b, c, d in order as necessary. The average number of guesses needed overall is then $a + 2b + 3c + 4d$, that is $d + (d{+}c) + (d{+}c{+}b) + (d{+}c{+}b{+}a)$.

$$
\begin{array}{lll}
 & (\sum \pi\colon \Pi \bullet \mathsf{W}.\pi) & \\
= & (\sum \pi\colon \Pi \bullet (\sum i \bullet \sqcap^i \pi)\,) & \text{“definition } \mathsf{W} \text{ for a partition”} \\
= & (\sum i \bullet (\sum \pi\colon \Pi \bullet \sqcap^i \pi)\,) & \text{“swap summations”} \\
\leq & (\sum i \bullet \sqcap^i (\sum \pi\colon \Pi \bullet \pi)\,) & \text{“subdistribute minimisation”} \\
= & \mathsf{W}.(\sum \pi\colon \Pi)\ . & \text{“definition } \mathsf{W} \text{ for partition”}
\end{array}
$$

When all the fractions π in Π are similar, we can replace the inequality in the second-last step with equality, establishing (i). □

Theorem 8. *Soundness of* $(\sqsubseteq)$ *w.r.t.* $(\preceq_{\mathsf{W}})$ For all programs S and I and contexts $\mathcal{C}$ we have that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq_{\mathsf{W}} \mathcal{C}(I)$.

Proof: Immediate from Lem. 8 and monotonicity of $(\sqsubseteq)$ (Thm. 2 in §6.5). □