An old new notation for elementary probability theory

Carroll Morgan

University of New South Wales, NSW 2052 Australia
carrollm@cse.unsw.edu.au *

Abstract. The Eindhoven approach to quantifier notation is 40 years old. We extend it by adding "distribution comprehensions" systematically to its repertiore; we believe the resulting notation for elementary probability theory is new.
After a step-by-step explanation of the proposed notational innovations, with small examples, we give as our exemplary case study the probabilistic reasoning associated with a quantitative noninterference semantics based on Hidden Markov Models of computation. Although that example was the motivation for this work, we believe the proposal here will be more generally applicable: and so we also revisit a number of popular puzzles, to illustrate the notation's wider utility.

Finally, we review the connection between comprehension notations and (category-theoretic) monads, and show how the Haskell approach to monad comprehensions applies to the distribution comprehensions we have introduced.

1 Context and motivation

Conventional notations for elementary probability theory are more descriptive than calculational. They communicate ideas, but they are not algebraic (as a rule) in the sense of helping to proceed reliably from one idea to the next one: and truly effective notations are those that we can reason *with* rather than simply *about*. In our recent work on security, the conventional notations for probability became so burdensome that we felt that it was worth investigating alternative, more systematic notations for their own sake.

The Eindhoven notation was designed in the 1970's to control complexity in reasoning about programs and their associated logics: the forty years since then have shown how effective it is. But as far as we know it has not been used for probability. We have done so by working "backwards," from an application in computer security (Sec. 9.2), with the Eindhoven style as a target (Sec. 2). That is the opposite, incidentally, of reconstructing elementary probability "forwards" from first principles — also a worthwhile goal, but a different one.

We judge our proposal's success by whether it simplifies reasoning about intricate probabilistic structures in computer science and elsewhere. For that

* This article extends an earlier publication XXX. We are grateful for the support of the Dutch NWO (Grant 040.11.303) and the Australian ARC (Grant DP1092464).

we give a small case study, based on noninterference-security semantics, both in the novel notation and in the conventional notation; and we compare them with each other (Sec. 9). We have also used the new notation more extensively [18].

Although the notation was developed retroactively, the account we give here is forwards, that is from the basics towards more advanced constructions. Along the way we use a number of popular puzzles as more general examples.

2 The Eindhoven quantifier notation, and our extension

In the 1970's, researchers at *THE* in Eindhoven led by E.W. Dijkstra proposed a uniform notation for quantifications in first-order logic, elementary set theory and related areas [5]. By $(\mathcal{Q}\, x{:}\,T \mid \mathit{rng} \bullet \mathit{exp})$ [1] they meant that *quantifier* $\mathcal{Q}$ *binds variable* x *of type* T *within textual scope* $(\cdots)$*, that* x *is constrained to satisfy formula rng, that expression exp is evaluated for each such* x *and that those values then are combined via an associative and commutative operator related to quantifier* $\mathcal{Q}$. These examples make the uniformity evident:

$(\forall x{:}\,T \mid \mathit{rng} \bullet \mathit{exp})$	means	for all x in T satisfying *rng* we have *exp*,
$(\exists x{:}\,T \mid \mathit{rng} \bullet \mathit{exp})$	means	for some x in T satisfying *rng* we have *exp*,
$(\Sigma x{:}\,T \mid \mathit{rng} \bullet \mathit{exp})$	means	the sum of all *exp* for x in T satisfying *rng*,
$\{x{:}\,T \mid \mathit{rng} \bullet \mathit{exp}\}$	means	the set of all *exp* for x in T satisfying *rng*.

A general shorthand applying to them all is that an omitted range $|\mathit{rng}$ defaults to $|\mathit{true}$, and an omitted $\bullet \mathit{exp}$ defaults to the bound variable $\bullet x$ itself.

These (once) novel notations are not very different from the conventional ones: they contain the same ingredients because they must. Mainly they are a reordering, an imposition of consistency, and finally a making explicit of what is often implicit: bound variables, and their scope. Instead of writing $\{n{\in}\mathbb{N} \mid n{>}0\}$ for the positive natural numbers we write $\{n{:}\,\mathbb{N} \mid n{>}0\}$, omitting the "$\bullet n$" via the shorthand above; the only difference is the explicit declaration of n via a colon (as in a programming language) rather than via $n{\in}\mathbb{N}$ which, properly speaking, is a formula (with both n and $\mathbb{N}$ free) and doesn't declare anything. And instead of $\{n^2 \mid n{\in}\mathbb{N}\}$ for the square numbers, we write $\{n{:}\,\mathbb{N} \bullet n^2\}$, keeping the declaration in first position (always) and avoiding ambiguous use of the vertical bar.

In program semantics one can find general structures such as

sets of distributions for probability and nondeterminism [24, 17],

distributions of distributions, for probabilistic noninterference security [18, 19], and even

sets of distributions of distributions to combine the two [20].

[1] The original Eindhoven style uses colons as separators; the syntax here with | and • is more in the *Z* style [11], and is one of many subsequent notational variations based on their innovation.

All of these are impeded by the conventional use of "Pr" to refer to probability with respect to some unnamed distribution "of interest" at the time: we need to refer to the whole distribution itself.

And when we turn to particular instances, the semantics of individual programs, we need to build functions corresponding to specific program components. The conventional "random variables" are inconvenient for this, since we must invent a name for every single one: we would rather use the expressions and variables occurring in the programs themselves. In the small –but nontrivial– example of information flow (Sec. 9), borrowed from our probabilistic security work [18–20], we compare the novel notation (Sec. 9.2) to the conventional (Sec. 9.3) in those respects.

Our essential extension of the Eindhoven quantifiers was to postulate a "distribution comprehension" notation $\{\!\{ s{:}\,\delta \mid rng \bullet exp \}\!\}$, intending it to mean "for all elements s in the distribution δ, conditioned by rng, make a new distribution based on evaluating exp." Thus we refer to a distribution itself (the whole comprehension), and we access random variables as expressions (the exp within). From there we worked backwards, towards primitives, to arrange that indeed the comprehension would have that meaning.

This report presents our results, but working forwards and giving simple examples as we go. Only at Def. 13 do we finally recover our conceptual starting point, a definition of the comprehension that agrees with the guesswork just above (Sec. 8.5).

3 Discrete distributions as enumerations

We begin with distributions written out explicitly: this is by analogy with the enumerations of sets which list their elements. The notation $f.x$ for application of function f to argument x is used from here on, except for type constructors where a distinct font allows us to reduce clutter by omitting the dot.

3.1 Finite discrete distributions as a type

A finite discrete distribution δ on a set S is a function assigning to each element s in S a (non-negative) probability $\delta.s$, where the sum of all such probabilities on S is one. The fair distribution on coin-flip outcomes $\{\mathsf{H}, \mathsf{T}\}$ takes both H, T to $1/2$; the distribution on die-roll outcomes $\{1{\cdot\cdot}6\}$ for a fair die gives $1/6$ for each integer n with $1{\leq}n{\leq}6$. In general we have

Definition 1. *The constructor $\mathbb{D}$ for finite discrete distributions*

The set $\mathbb{D}S$ of discrete distributions over a finite set S is the functions from S into $[0, 1]$ that sum to one, that is $\{\delta{:}\, S{\to}[0, 1] \mid (\Sigma s{:}\, S \bullet \delta.s) = 1\}$. The set S is called the *basis* (type) of δ. □

In Def. 1 the omitted $|rng$ of $\sum$ is $|true$, and the omitted $\bullet exp$ of $\{\cdots\}$ is $\bullet\delta$. One reason for using distinct symbols | and • is that in the default cases those symbols can be omitted as well, with no risk of ambiguity.

3.2 The support of a distribution

The support of a distribution is that subset of its basis to each of whose elements it assigns nonzero probability; it is in an informal sense the "relevant" or "interesting" elements in the distribution. We define

Definition 2. *Support of a distribution* For distribution $\delta: \mathbb{D}S$ with basis S, the support is the subset $\lceil\delta\rceil := \{s: S \mid \delta.s \neq 0\}$ of S. □

The "ceiling" notation $\lceil\cdot\rceil$ suggests the pointwise ceiling of a distribution which, as a function (Def. 1), is the characteristic function of its support.

3.3 Specialised notation for uniform distributions

By analogy with set enumerations like $\{\mathsf{H}, \mathsf{T}\}$, we define uniform-distribution enumerations that assign the same probability to every element in their support:

Definition 3. *Uniform-distribution enumeration* The uniform distribution over an enumerated set $\{a, b, \cdots, z\}$ is written $\{\!\{a, b, \cdots, z\}\!\}$. It is assumed that the values $a, b, \cdots, z$ are distinct and that there is at least one. □

Thus for example the fair-coin distribution is $\{\!\{\mathsf{H}, \mathsf{T}\}\!\}$ and the fair-die distribution is $\{\!\{1{\cdot\cdot}6\}\!\}$.

As a special case of uniform distribution we have the point distribution $\{\!\{a\}\!\}$ on some element a, assigning probability 1 to it: this is analogous to the singleton set $\{a\}$ that contains only a.

3.4 General notation for distribution enumerations

For distributions that are not uniform, we attach a probability explicitly to each element. Thus we have $\{\!\{\mathsf{H}^{@\frac{2}{3}}, \mathsf{T}^{@\frac{1}{3}}\}\!\}$ for the coin that is twice as likely to give heads H as tails T, and $\{\!\{1^{@\frac{2}{9}}, 2^{@\frac{1}{9}}, 3^{@\frac{2}{9}}, 4^{@\frac{1}{9}}, 5^{@\frac{2}{9}}, 6^{@\frac{1}{9}}\}\!\}$ for the die that is twice as likely to roll odd as even (but is uniform otherwise). In general we have

Definition 4. *Distribution enumeration* We write $\{\!\{a^{@p_a}, b^{@p_b}, \cdots, z^{@p_z}\}\!\}$ for the distribution over set $\{a, b, \cdots, z\}$ that assigns probability p_a to element a etc. For well-formedness we require that $p_a + p_b + \cdots + p_z = 1$. □

3.5 The support of a distribution is a subset of its basis

Strictly speaking one can't tell, just by drawing samples, whether $\{\!\{\mathsf{H}, \mathsf{T}\}\!\}$ represents the distribution of a fair two-sided coin, or instead represents the distribution of a three-sided coin with outcomes $\{\mathsf{H}, \mathsf{T}, \mathsf{E}\}$ that never lands on its edge E. Similarly we might not know whether $\{\!\{6\}\!\}$ describes a die that has the numeral 6 written on every face or a loaded die that always rolls 6.

Saying that δ is uniform *over* S means it is uniform and its support is S.

3.6 Specialised infix notations for making distributions

For distributions of support no more than two we have the special notation

Definition 5. *Doubleton distribution* For any elements a, b and $0 \leq p \leq 1$ we write $a \,{}_p\!\oplus\, b$ for the distribution $\{\!\{a^{@p}, b^{@1-p}\}\!\}$. □

Thus the fair-coin distribution $\{\!\{\mathsf{H}, \mathsf{T}\}\!\}$ can be written $\mathsf{H} \,{}_{1/2}\!\oplus\, \mathsf{T}$. For the weighted sum of two distributions we have

Definition 6. *Weighted sum* For two numbers x, y and $0 \leq p \leq 1$ we define the sum $x \,{}_p\!+\, y := px + (1-p)y$; more generally x, y can be elements of a vector space.

In particular, for two distributions $\delta, \delta' : \mathbb{D}S$ we define their weighted sum $\delta \,{}_p\!+\, \delta'$ by $(\delta \,{}_p\!+\, \delta').s := p(\delta.s) + (1-p)(\delta'.s)$ for all s in S. □

Thus the biased die from Sec. 3.4 can be written as $\{\!\{1,3,5\}\!\} \,{}_{2/3}\!+\, \{\!\{2,4,6\}\!\}$, showing at a glance that its odds and evens are uniform on their own, but that collectively the odds are twice as likely as the evens.

As simple examples of algebra we have first $x \,{}_p\!\oplus\, y = \{\!\{x\}\!\} \,{}_p\!+\, \{\!\{y\}\!\}$, and then

$$\delta \,{}_0\!+\, \delta' = \delta' \qquad \text{and} \quad \delta \,{}_1\!+\, \delta' = \delta$$
$$\text{and} \quad \lceil \delta \,{}_p\!+\, \delta' \rceil = \lceil \delta \rceil \cup \lceil \delta' \rceil \text{ when } 0 < p < 1.$$

3.7 Comparison with conventional notation [2]

Conventionally a distribution is over a *sample space* S, which we have called the basis (Def. 1). Subsets of the sample space are *events*, and a distribution assigns a number to every event, the probability that an observation "sampled" from the sample space will be an occurrence of that event. That is, a distribution is of type $\mathbb{P}S \to [0,1]$ from subsets of S rather than from its elements.

With our odd-biased die in Sec. 3.4 the sample space is $S = \{1 \cdot\cdot 6\}$ and the probability $2/3$ of "rolled odd," that is of the event $\{1,3,5\} \subset S$, is twice the probability $1/3$ of "rolled even," that is of the event $\{2,4,6\} \subset S$.

There are "additivity" conditions placed on general distributions, among which are that the probability assigned to the union of two disjoint events should be the sum of the probabilities assigned to the events separately, that the probability assigned to all of S should be one, and that the probability assigned to the empty event should be zero.

When S is finite, the general approach specialises so that a *discrete* distribution δ acts on separate points, instead of on sets of them. The probability of any event $S' \subset S$ is then just $\sum_{s \in S'} \delta(s)$ from additivity.

[2] When presenting examples in conventional notation, in this and later comparison sections, we will write $f(x)$ instead of $f.x$ and $\{exp \mid x \in S\}$ instead of $\{x : S \bullet exp\}$.

4 Expected values over discrete distributions

4.1 Definition of expected value as average

If the basis S of a distribution $\delta: \mathbb{D}S$ comprises numbers or, more generally, is a vector space, then the "weighted average" of the distribution is the sum of the values in S multiplied by the probability that δ assigns to each, that is $(\Sigma s: S \bullet \delta.s \times s)$. For the fair die that becomes $(1+2+3+4+5+6)/6 = 3\frac{1}{2}$; for the odd-biased die the average is $4\frac{2}{3}$.

For the fair coin $\{\!|\mathsf{H},\mathsf{T}|\!\}$ however we have no average, since $\{\mathsf{H},\mathsf{T}\}$ has no arithmetic. We must work indirectly via a function on the basis, using

Definition 7. *Expected value* By $(\mathcal{E}s: \delta \bullet \mathit{exp})$ we mean the expected value of function $(\boldsymbol{\lambda}\, s \bullet \mathit{exp})$ over distribution δ; it is

$$(\mathcal{E}s: \delta \bullet \mathit{exp}) \quad := \quad (\Sigma s: \lceil \delta \rceil \bullet \delta.s \times \mathit{exp}) \ . \quad [3]$$

Note that *exp* is an expression in which bound variable s probably appears (though it need not). We call *exp* the *constructor*. □

For example, the expected value of the *square* of the value rolled on a fair die is $(\mathcal{E}s: \{\!|1\mathinner{..}6|\!\} \bullet s^2) = (1^2 + \cdots + 6^2)/6 = 15\frac{1}{6}$.

For further examples, we name a particular distribution $\hat{\delta} := \{\!|0,1,2|\!\}$ and describe a notation for converting Booleans to numbers:

Definition 8. *Booleans converted to numbers* The function $[\cdot]$ takes Booleans T,F to numbers 0,1 so that $[\mathsf{T}] := 1$ and $[\mathsf{F}] := 0$. [4] □

Then we have

$$\begin{array}{rrcl} & (\mathcal{E}s: \hat{\delta} \bullet s \ \mathbf{mod}\ 2) & = & 1/3\times 0 + 1/3\times 1 + 1/3\times 0 \ = \ 1/3 \\ \text{and} & (\mathcal{E}s: \hat{\delta} \bullet [s \neq 0]) & = & 1/3\times 0 + 1/3\times 1 + 1/3\times 1 \ = \ 2/3 \ , \end{array}$$

where in the second case we have used Def. 8 to convert the Boolean $s \neq 0$ to a number. Now we can formulate the average proportion of heads shown by a fair coin as $(\mathcal{E}s: \{\!|\mathsf{H},\mathsf{T}|\!\} \bullet [s = \mathsf{H}]) = \frac{1}{2}$.

4.2 The probability of a subset rather than of a single element

We can use the expected value quantifier to give the aggregate probability assigned to a (sub)set of outcomes, provided we have a formula describing that set. [5] When *exp* is Boolean, we have that $(\mathcal{E}s: \delta \bullet [\mathit{exp}])$ is the probability assigned by δ to the whole of the set $\{s: \lceil \delta \rceil \mid \mathit{exp}\}$. This is because the expected value of the characteristic function of a set is equal to the probability of that set as a whole. An example of this occurs just after Def. 8; another is given at 4.3(e) below.

[3] Here is an example of not needing to know the basis type: we simply sum over the support of δ, since the other summands will be zero anyway.

[4] We disambiguate T for *true* and T for *tails* by context.

[5] Note that those aggregate probabilities do not sum to one over all subsets of the basis, since the individual elements would be counted many times.

4.3 Abbreviation conventions

The following are five abbreviations that we use in the sequel.

(a) If several bound variables are drawn from the same distribution, we assume they are drawn independently from separate instances of it. Thus $(\mathcal{E}x, y{:}\,\delta \bullet \cdots)$ means $(\mathcal{E}x{:}\,\delta; y{:}\,\delta \bullet \cdots)$ or equivalently $(\mathcal{E}(x,y){:}\,\delta^2 \bullet \cdots)$.
(b) If in an expected-value quantification the *exp* is omitted, it is taken to be the bound variable standing alone (or a tuple of them, if there are several). Thus $(\mathcal{E}s{:}\,\delta)$ means $(\mathcal{E}s{:}\,\delta \bullet s)$, and more generally $(\mathcal{E}x, y{:}\,\delta)$ means $(\mathcal{E}x, y{:}\,\delta \bullet (x,y))$ with appropriate arithmetic induced on $\lceil\delta\rceil \times \lceil\delta\rceil$.
(c) By analogy with summation, where for a set S we abbreviate $(\Sigma s{:}\,S)$ by $\sum S$, we abbreviate $(\mathcal{E}s{:}\,\delta)$ by $\mathcal{E}\delta$. Thus $\mathcal{E}\hat{\delta} = \mathcal{E}\{\!|0,1,2|\!\} = (0+1+2)/3 = 1$.
(d) If a set is written where a distribution is expected, we assume implicitly that it is the uniform distribution over that set. Thus $\mathcal{E}\{0,1,2\} = \mathcal{E}\hat{\delta} = 1$.
(e) If a Boolean expression occurs where a number is expected, then we assume an implicit application of the conversion function $[\cdot]$ from Def. 8. Thus $(\mathcal{E}s{:}\,\{0,1,2\} \bullet s{\neq}0) = 2/3$ is the probability that a number chosen uniformly from $0, 1, 2$ will not be zero.

4.4 Example of expected value: dice at the fairground

Define the set D to be $\{1{\cdot\cdot}6\}$, the possible outcomes of a die roll.

At the fairground there is a tumbling cage with three fair dice inside, and a grid of six squares marked by numbers from D. You place \$1 on a square, and watch the dice tumble until they stop.

If your number appears exactly once among the dice, then you get your \$1 back, plus \$1 more; if it appears twice, you get \$2 more; if it appears thrice you get \$3 more. If it's not there at all, you lose your \$1.

Using our notation so far, your expected profit is written

$$-1 \;+\; (\mathcal{E}s_1, s_2, s_3{:}\,D \bullet (\bigvee i \bullet s_i{=}s) + (\Sigma i \bullet s_i{=}s)) \;, \tag{1}$$

where the initial -1 accounts for the dollar you paid to play, and the free variable s is the number of the square on which you placed it. The disjunction describes the event that you get your dollar back; and the summation describes the extra dollars you (might) get as well.

The D is converted to a uniform distribution by 4.3(d), then replicated three times by 4.3(a), independently for $s_{\{1,2,3\}}$; and the missing conversions from Boolean to 0,1 are supplied by 4.3(e).

Finally we abuse notation by writing s_i even though i is itself a (bound) variable: e.g. by $(\bigvee i \bullet s_i{=}s)$ we mean in fact $s_1{=}s \vee s_2{=}s \vee s_3{=}s$.[6]

[6] It is an abuse because in the scope of i we are using it as if it were an argument to some function $s_{(\cdot)}$ — but the s is is already used for something else. Moreover s_1, s_2, s_3 must themselves be names (not function applications) since we quantify over them with $\mathcal{E}$. Also we gave no type for i.

While the point of this example is the way in which (1) is written, for those tempted to play this game it's worth pointing out that its value is approximately $-.08$, independent of s. That is an expected loss of about eight cents in the dollar on every play and no matter which square is chosen.

4.5 Comparison with conventional notation

Conventionally, expected values are taken over *random variables* that are functions from the sample space into a set with arithmetic, usually the reals (but more generally a vector space). Standard usage is first to define the sample space, then to define a distribution over it, and finally to define a random variable over the sample space and give it a name, say X. Then one writes $\Pr(X{=}x)$ for the probability assigned by that distribution to the event that the (real-valued) random variable X takes some (real) value x; and $\mathbf{E}(X)$ is the notation for the expected value of random variable X over the same (implicit) distribution.

In Def. 7 our random variable is $(\boldsymbol{\lambda}\, s \bullet \mathit{exp})$, and we can write it without a name since its bound variable s is already declared. Furthermore, because we can give the distribution δ explicitly, we can write expressions in which the distributions are themselves expressions. As examples, we have

$$
\begin{array}{rll}
(\mathcal{E}s{:}\,\{\!|e|\!\} \bullet \mathit{exp}) = \mathit{exp}[s\backslash e] & & \text{– one-point rule} \\
(\mathcal{E}s{:}\,(\delta \;{}_p{+}\; \delta') \bullet \mathit{exp}) = (\mathcal{E}s{:}\,\delta \bullet \mathit{exp}) \;{}_p{+}\; (\mathcal{E}s{:}\,\delta' \bullet \mathit{exp}) & & \text{– using Def. 6} \\
(\mathcal{E}s{:}\,(x \;{}_p{\oplus}\; y) \bullet \mathit{exp}) = \mathit{exp}[s\backslash x] \;{}_p{+}\; \mathit{exp}[s\backslash y] & & \text{– from the two above,}
\end{array}
$$

where $\mathit{exp}[s\backslash e]$ is bound-variable-respecting replacement of s by e in exp.

For example, if δ is the distribution of random variable X, then the conventional $\Pr(X{=}x)$ becomes $\delta.x$, and $\mathbf{E}(X)$ becomes $\mathcal{E}\delta$.

5 Discrete distributions as comprehensions

5.1 Definition of distribution comprehensions

With a comprehension, a distribution is defined by properties rather than by enumeration. Just as the set comprehension $\{s{:}\,\lceil\hat{\delta}\rceil \bullet s^2\}$ gives the set $\{0,1,4\}$ having the property that its elements are precisely the squares of the elements of $\lceil\hat{\delta}\rceil = \{0,1,2\}$, we would expect $\{\!|s{:}\,\hat{\delta} \bullet s^2|\!\}$ to be $\{\!|0,1,4|\!\}$ where in this case the uniformity of the source $\hat{\delta}$ has induced uniformity in the target.

If however some of the target values "collide," because exp is not injective, then their probabilities add together: thus we have $\{\!|s{:}\,\hat{\delta} \bullet s \;\mathbf{mod}\; 2|\!\} =$

Although our purpose is to show how we achieve a concise presentation with precise notation, we are at the same time arguing that "to abuse, or not to abuse" should be decided on individual merits. There are times when a bit of flexibility is helpful: arguably the abuse here gains more in readability than it loses in informality.

A similar use is $(\exists i \bullet \cdots H_i \cdots)$ for the weakest precondition of a loop: this finesse avoided swamping a concise first-order presentation with (mostly unnecessary) higher-order logic throughout [4].

$\{\!\{0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}\}\!\} = 0\ {}_{2/3}\!\oplus 1$, where target element 0 has received 1/3 probability as $0 \bmod 2$ and another 1/3 as $2 \bmod 2$.

We define distribution comprehensions by giving the probability they assign to an arbitrary element; thus

Definition 9. *Distribution comprehension* For distribution δ and arbitrary value e of the type of *exp* we define

$$\{\!\{s{:}\,\delta \bullet exp\}\!\}.e \quad := \quad (\mathcal{E}s{:}\,\delta \bullet [exp{=}e])\ . \quad ^{7}$$

□

The construction is indeed a distribution on $\{s{:}\,\lceil\delta\rceil \bullet exp\}$ (Lem. 1 in App. C), and assigns to element e the probability given by δ that $exp{=}e$ as s ranges over the basis of δ. [8]

Note that $\{\!\{s{:}\,\delta\}\!\}$ is therefore just δ itself.

5.2 Examples of distribution comprehensions

We have from Def. 9 that the probability $\{\!\{s{:}\,\hat{\delta} \bullet s \bmod 2\}\!\}$ assigns to 0 is

$$\begin{array}{ll} & \{\!\{s{:}\,\hat{\delta} \bullet s \bmod 2\}\!\}.0 \\ = & (\mathcal{E}s{:}\,\hat{\delta} \bullet [s \bmod 2 = 0]) \\ = & 1/3{\times}[0{=}0] + 1/3{\times}[1{=}0] + 1/3{\times}[0{=}0] \\ = & 1/3{\times}1 + 1/3{\times}0 + 1/3{\times}1 \\ = & 2/3\ , \end{array}$$

and the probability $\{\!\{s{:}\,\hat{\delta} \bullet s \bmod 2\}\!\}$ assigns to 1 is

$$\begin{array}{ll} & \{\!\{s{:}\,\hat{\delta} \bullet s \bmod 2\}\!\}.1 \\ = & 1/3{\times}[0{=}1] + 1/3{\times}[1{=}1] + 1/3{\times}[0{=}1] \\ = & 1/3\ . \end{array}$$

Thus we have verified that $\{\!\{s{:}\,\hat{\delta} \bullet s \bmod 2\}\!\} = 0\ {}_{2/3}\!\oplus 1$ as stated in Sec. 5.1.

---

[7] Compare $\{x{:}\,X \bullet exp\}\ni e$ defined to be $(\exists x{:}\,X \bullet exp{=}e)$.

[8] A similar comprehension notation is used in cryptography, for example the

$$\{s \xleftarrow{R} S; s' \xleftarrow{R} S' : exp\}$$

that in this case takes bound variables (s, s') uniformly $(\xleftarrow{R})$ from sample spaces (S, S') and, with them, makes a new distribution via a constructor expression (exp) containing those variables. We would write that as $\{\!\{s{:}\,S; s'{:}\,S' \bullet exp\}\!\}$ with the S, S' converted to uniform distributions by 4.3(d).

5.3 Comparison with conventional notation

Conventionally one makes a target distribution from a source distribution by "lifting" some function that takes the source sample space into a target. We explain that here using the more general view of distributions as functions of *subsets* of the sample space (Sec. 3.7), rather than as functions of single elements.

If δ_X is a distribution over sample space X, and we have a function $f\colon X\to Y$, then distribution δ_Y over Y is defined $\delta_Y(Y') := \delta_X(f^{-1}(Y'))$ for any subset Y' of Y. We then write $\delta_Y = f_*(\delta_X)$, and function $f_*\colon \mathbb{D}X\to\mathbb{D}Y$ is called the *push-forward*; it makes the *image measure* wrt. $f\colon X\to Y$ [7, index]. We could write the target distribution directly as $\delta_Y = \{\!|s\colon \delta_X \bullet f.s|\!\}$.

In the distribution comprehension $\{\!|s\colon \delta \bullet exp|\!\}$ for $\delta\colon \mathbb{D}S$, the source distribution is δ and the function f between the sample spaces is $(\boldsymbol{\lambda}\, s\colon S \bullet exp)$. The induced push-forward f_* is then the function $(\boldsymbol{\lambda}\, \delta\colon \mathbb{D}S \bullet \{\!|s\colon \delta \bullet exp|\!\})$.

6 Conditional distributions

6.1 Definition of conditional distributions

Given a distribution and an event, the latter a subset of possible outcomes, a conditioning of that distribution by the event is a new distribution formed by restricting attention to that event and ignoring all other outcomes. For that we have

Definition 10. *Conditional distribution* Given a distribution δ and a "range" predicate *rng* in variable s ranging over the basis of δ, the *conditional distribution of δ given rng* is determined by

$$\{\!|s\colon \delta \mid rng|\!\}.s' \quad := \quad \frac{(\mathcal{E}s\colon \delta \bullet rng \times [s{=}s'])}{(\mathcal{E}s\colon \delta \bullet rng)}\ ,$$

for any s' in the basis of δ. We appeal to the abbreviation 4.3(e) to suppress the explicit conversion $[rng]$ on the right. [9]

The denominator must not be zero (Lem. 2 in App. C). □

In Def. 10 the distribution δ is initially restricted to the subset of the sample space defined by *rng* (in the numerator), potentially making a subdistribution because it no longer sums to one. It it restored to a full distribution by normalisation, the effect of dividing by its weight (the denominator).

6.2 Example of conditional distributions

A simple case of conditional distribution is illustrated by the uniform distribution $\hat{\delta} = \{\!|0,1,2|\!\}$ we defined earlier. If we condition on the event "is not zero" we

[9] Leaving the $[\cdot]$ out enables a striking notational economy in Sec. 8.2.

find that $\{\!\{s\colon \hat{\delta} \mid s{\neq}0\}\!\} = \{\!\{1,2\}\!\}$, that when s is not zero it is equally likely to be 1 or 2. We verify this via Def. 10 and the calculation

$$\begin{array}{ll} & \{\!\{s\colon \hat{\delta} \mid s{\neq}0\}\!\}.1 \\ = & (\mathcal{E}s\colon \{\!\{0,1,2\}\!\} \bullet [s{\neq}0] \times [s{=}1]) \;/\; (\mathcal{E}s\colon \{\!\{0,1,2\}\!\} \bullet [s{\neq}0]) \\ = & \frac{1}{3}/\frac{2}{3} \\ = & 1/2\ . \end{array}$$

6.3 Comparison with conventional notation

Conventionally one refers to the conditional probability of an event A given some (other) event B, writing $\Pr(A|B)$ whose meaning is given by the Bayes formula $\Pr(A{\wedge}B)/\Pr(B)$. Usually A, B are names (occasionally expressions) referring to events or random variables defined in the surrounding text, and Pr refers, in the usual implicit way, to the probability distribution under consideration. Well-definedness requires that $\Pr(B)$ be nonzero.

Def. 10 with its conversions 4.3(e) explicit becomes

$$(\mathcal{E}s\colon \delta \bullet [s{=}s' \wedge \mathit{rng}]) \;/\; (\mathcal{E}s\colon \delta \bullet [\mathit{rng}])\ ,$$

with Event A corresponding to "is equal to s'" and Event B to "satisfies *rng*."

7 Conditional expectations

7.1 Definition of conditional expectations

We now put constructors *exp* and ranges *rng* together in a single definition of conditional expectation, generalising conditional distributions:

Definition 11. *Conditional expectation* Given a distribution δ, predicate *rng* and expression *exp* both in variable s ranging over the basis of δ, the *conditional expectation of exp over δ given rng* is

$$(\mathcal{E}s\colon \delta \mid \mathit{rng} \bullet \mathit{exp}) \quad := \quad \frac{(\mathcal{E}s\colon \delta \bullet \mathit{rng} \times \mathit{exp})}{(\mathcal{E}s\colon \delta \bullet \mathit{rng})}\ , \qquad [10]$$

in which the expected values on the right are in the simpler form to which Def. 7 applies, and *rng*,*exp* are converted if necessary according to 4.3(e).

The denominator must not be zero. □

7.2 Conventions for default range

If *rng* is omitted in $(\mathcal{E}s\colon \delta \mid \mathit{rng} \bullet \mathit{exp})$ then it defaults to T, that is *true* as a Boolean or 1 as a number: and this agrees with Def. 7. To show that, in this section only we use $\underline{\boldsymbol{\mathcal{E}}}$ for Def. 11 and reason

[10] From (14) in Sec. 12 we will see this equivalently as $(\mathcal{E}s\colon \{\!\{s\colon \delta \mid \mathit{rng}\}\!\} \bullet \mathit{exp})$.

$$
\begin{array}{lll}
 & (\underline{\boldsymbol{\mathcal{E}}}s{:}\,\delta \bullet \mathit{exp}) & \text{“as interpreted in Def. 11”} \\
= & (\underline{\boldsymbol{\mathcal{E}}}s{:}\,\delta \mid \mathsf{T} \bullet \mathit{exp}) & \text{“default } \mathit{rng} \text{ is } \mathsf{T}\text{”} \\
= & (\mathcal{E}s{:}\,\delta \bullet [\mathsf{T}] \times \mathit{exp}) \;/\; (\mathcal{E}s{:}\,\delta \bullet [\mathsf{T}]) & \text{“Def. 11 and 4.3(e)”} \\
= & (\mathcal{E}s{:}\,\delta \bullet \mathit{exp}) \;/\; (\mathcal{E}s{:}\,\delta \bullet 1) & \text{“}[\mathsf{T}]{=}1\text{”} \\
= & (\mathcal{E}s{:}\,\delta \bullet \mathit{exp})\ . & \text{“}(\mathcal{E}s{:}\,\delta \bullet 1) = (\Sigma s{:}\,S \bullet \delta.s) = 1\text{”}
\end{array}
$$

More generally we observe that a nonzero range *rng* can be omitted whenever it contains no free s, of which “being equal to the default value T” is a special case. That is because it can be distributed out through the $(\mathcal{E}s)$ and then cancelled.

7.3 Examples of conditional expectations

In our first example we ask for the probability that a value chosen according to distribution $\hat{\delta}$ will be less than two, given that it is not zero.

Using the technique of Sec. 4.2 we write $(\mathcal{E}s{:}\,\hat{\delta} \mid s{\neq}0 \bullet s{<}2)$ which, via Def. 11, is equal to $1/2$. Our earlier example at Sec. 6.2 also gives $1/2$, the probability of being less than two in the uniform distribution $\{\!|1, 2|\!\}$.

Our second example is the expected value of a fair die roll, given that the outcome is odd. That is written $(\mathcal{E}s{:}\,D \mid s \,\mathbf{mod}\, 2 = 1)$, using the abbreviation of 4.3(b) to omit the constructor s. Via Def. 11 it evaluates to $(1{+}3{+}5)/3 = 3$.

7.4 Comparison with conventional notation

Conventionally one refers to the expected value of some random variable X given that some other random variable Y has a particular value y, writing $\mathbf{E}(X|Y{=}y)$. With X, Y and the distribution referred to by $\mathbf{E}$ having been fixed in the surrounding text, the expression's value is a function of y.

Our first example in Sec. 7.3 is more of conditional probability than of conditional expectation: we would state in the surrounding text that our distribution is $\hat{\delta}$, that event A is “is nonzero” and event B is “is less than two.” Then we would have $\Pr(A|B) = 1/2$.

In our second example, the random variable X is the identity on D, the random variable Y is the $\mathbf{mod}\,2$ function, the distribution is uniform on D and the particular value y is 1. Then we have $\mathbf{E}(X|Y{=}1) = 3$.

8 Belief revision: *a priori* and *a posteriori* reasoning

8.1 A-priori and a-posteriori distributions in conventional style: introduction and first example

A priori, i.e. “before” and *a posteriori*, i.e. “after” distributions refer to situations in which a distribution is known (or believed) and then an observation is made that changes one's knowledge (or belief) in retrospect. This is sometimes known as *Bayesian belief revision*. A typical real-life example is the following.

> In a given population the incidence of a disease is believed to be one person in a thousand. There is a test for the disease that is 99% accurate. A patient who arrives at the doctor is therefore *a priori* believed to have only a 1/1,000 chance of having the disease; but then his test returns positive. What is his *a posteriori* belief that he has the disease?

The patient probably thinks the chance is now 99%. But the accepted Bayesian analysis is that one compares the probability of having the disease, and testing positive, with the probability of testing positive on its own (i.e. including false positives). That gives for the *a posteriori* belief

$$\begin{aligned}
&\Pr(\text{has disease} \wedge \text{test positive}) \;/\; \Pr(\text{test positive}) \\
&= (1/1000) \times (99/100) \;/\; ((1/1000) \times (99/100) + (999/1000) \times (1/100)) \\
&= 99 \;/\; (99 + 999) \\
&\approx 9\% \;,
\end{aligned}$$

that is less than one chance in ten, and not 99% at all. Although he is believed one hundred times more likely than before to have the disease, still it is ten times less likely than he feared.

8.2 Definition of *a posteriori* expectation

We begin with expectation rather than distribution, and define

Definition 12. A posteriori *expectation* Given a distribution δ, an experimental outcome *rng* and expression *exp* both possibly containing variable s ranging over the basis set of δ, the a posteriori *conditional expectation of exp over δ given rng* is $(\mathcal{E}s{:}\,\delta \mid rng \bullet exp)$, as in Def. 11 but without requiring *rng* to be Boolean. □

This economical reuse of the earlier definition, hinted at in Sec. 6.1, comes from interpreting *rng* not as a predicate but rather as the probability, depending on s, of observing some result. Note that since it varies with s it is not (necessarily) based on any *single* probability distribution, as we now illustrate.

8.3 Second example of belief revision: Bertrand's Boxes

Suppose we have three boxes, identical in appearance and named Box 0, Box 1 and Box 2. Each one has two balls inside: Box 0 has two black balls, Box 1 has one white- and one black ball; and Box 2 has two white balls.

A box is chosen at random, and a ball is drawn randomly from it. *Given that the ball was white*, what is the chance the other ball is white as well?

Using Def. 12 we describe this probability as

$$(\mathcal{E}b{:}\,\hat{\delta} \mid b/2 \bullet b{=}2) \;, \tag{2}$$

exploiting the box-numbering convention to write $b/2$ for the probability of observing the event "ball is white" if drawing randomly from Box b. Since

$(\Sigma b{:}\{0,1,2\} \bullet b/2)$ is $3/2 \neq 1$, it's clear that $b/2$ is not based on some single distribution, even though it is a probability. Direct calculation based on Def. 12 gives

$$\begin{array}{ll} & (\mathcal{E}b{:}\,\hat{\delta} \mid b/2 \bullet b{=}2) \\ = & (\mathcal{E}b{:}\,\{\!\{0,1,2\}\!\} \bullet b/2 \times [b{=}2]) \;/\; (\mathcal{E}b{:}\,\{\!\{0,1,2\}\!\} \bullet b/2) \\ = & \frac{1}{3}{\times}\frac{2}{2} \;/\; (\frac{1}{3}{\times}\frac{0}{2} + \frac{1}{3}{\times}\frac{1}{2} + \frac{1}{3}{\times}\frac{2}{2}) \\ = & \frac{1}{3} \;/\; \frac{1}{2} \\ = & 2/3\ . \end{array}$$

The other ball is white with probability $2/3$.

8.4 Third example of belief revision: The Reign in Spain

In Spain the rule of succession is currently that the next monarch is the eldest son of the current monarch, if there is a son at all: thus an elder daughter is passed over in favour of a younger son. We suppose that the current king had one sibling at the time he succeeded to the throne. What is the probability that his sibling was a brother? [11]

The answer to this puzzle will be given by an expression of the form

$$(\mathcal{E}\,\mathsf{two\ siblings} \mid \mathsf{one\ is\ king} \bullet \mathsf{the\ other\ is\ male})\ ,$$

and we deal with the three phrases one by one.

For $\mathsf{two\ siblings}$ we introduce two Boolean variables $c_{\{0,1\}}$, that is c for "child" and with the larger subscript 1 denoting the child with the larger age (i.e. the older one). Value T means "is male," and each Boolean will be chosen uniformly, reflecting the an assumption that births are fairly distributed between the two genders.

For $\mathsf{the\ other\ is\ male}$ we write $c_0 \wedge c_1$ since the king himself is male, and therefore his sibling is male just when they both are. We have now reached

$$(\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid \mathsf{one\ is\ king} \bullet c_0 \wedge c_1)\ . \tag{3}$$

In the Spanish system, there will be a king (as opposed to a queen) just when *either* sibling is male: we conclude our "requirements analysis" with the formula

$$(\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid c_0 \vee c_1 \bullet c_0 \wedge c_1)\ . \tag{4}$$

It evaluates to $1/3$ via Def. 12: in Spain, kings are more likely to have sisters.

Proceeding step-by-step as we did above allows us easily to investigate alternative situations. What would the answer be in Britain, where the eldest sibling becomes monarch regardless of gender? In that case we would start from (3) but reach the final formulation $(\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid c_1 \bullet c_0 \wedge c_1)$ instead of the Spanish formulation (4) we had before. We could evaluate this directly from Def. 12; but more interesting is to illustrate the algebraic possibilities for simplifying it:

[11] We see this as belief revision if we start by assuming the monarch's only sibling is as likely to be male as female; when we learn that the monarch is a Spanish king, we revise our belief.

$$
\begin{array}{lll}
 & (\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid c_1 \bullet c_0 \wedge c_1) & \text{“British succession”} \\
= & (\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid c_1 \bullet c_0 \wedge \mathsf{T}) & \text{“}c_1 \text{ is } \mathsf{T}\text{, from the range”} \\
= & (\mathcal{E}c_0, c_1{:}\,\mathsf{Bool} \mid c_1 \bullet c_0) & \text{“Boolean identity”} \\
= & (\mathcal{E}c_0{:}\,\mathsf{Bool} \mid (\mathcal{E}c_1{:}\,\mathsf{Bool} \bullet c_1) \bullet c_0) & \text{“}c_1 \text{ not free in constructor } (\bullet c_0)\text{: see below”} \\
= & (\mathcal{E}c_0{:}\,\mathsf{Bool} \mid 1/2 \bullet c_0) & \text{“Def. 7”} \\
= & (\mathcal{E}c_0{:}\,\mathsf{Bool} \bullet c_0) & \text{“remove constant range: recall Sec. 7.2”} \\
= & 1/2\ . & \text{“Def. 7”}
\end{array}
$$

We set the above out in unusually small steps simply in order to illustrate its (intentional) similarity with normal quantifier-based calculations. The only non-trivial step was the one labelled “see below”: it is by analogy with the set equality $\{s{:}\,S; s'{:}\,S' \mid \mathit{rng} \bullet \mathit{exp}\} = \{s{:}\,S \mid (\exists s'{:}\,S' \bullet \mathit{rng}) \bullet \mathit{exp}\}$ that applies when s' is not free in exp. We return to it in Sec. 12.

8.5 General distribution comprehensions

Comparison of Def. 10 and Def. 12 suggests a general form for distribution comprehensions, comprising both a range and a constructor. It is

Definition 13. *General distribution comprehensions* Given a distribution δ, an experimental outcome rng in variable s that ranges over the basis set of δ and a constructor exp, the general *a posteriori* distribution formed via that constructor is determined by

$$\{\!|s{:}\,\delta \mid \mathit{rng} \bullet \mathit{exp}|\!\}.e \quad := \quad (\mathcal{E}s{:}\,\delta \mid \mathit{rng} \bullet [\mathit{exp}{=}e])\ ,$$

for arbitrary e of the type of exp. (See Sec. 10.8 for an alternative definition.) □

Thus $\{\!|c_0, c_1{:}\,\mathsf{Bool} \mid c_0 \vee c_1 \bullet c_0 \wedge c_1|\!\} = \mathsf{T}\,_{1/3}\!\oplus \mathsf{F}$, giving the distribution of kings' siblings in Spain.

8.6 Comparison with conventional notation

Conventional notation for belief revision is similar to the conventional notation for conditional reasoning once we take the step of introducing the *joint distribution*.

In the first example, from Sec. 8.1, we would consider the joint distribution over the product space, that is

Joint sample space (Cartesian product)	Joint distribution $\times 100,000$	
	⊞	⊟
{has disease ☹, doesn't have disease ☺}		
× {test positive ⊞, test negative ⊟}		
☹	1×99	1×1
☺	999×1	999×99

and then the column corresponding to ⊞, i.e. test positive, assigns weights 99 and 999 to ☹ and ☺ respectively. Normalising those weights gives the distribution $☹\,_{9\%}\!\oplus ☺$ for the *a posteriori* health of the patient.

Thus we would establish that joint distribution, in the surrounding text, as the referent of Pr, then define as random variables the two projection functions H (health) and T (test), and finally write for example $\Pr(H{=}☹|T{=}⊞) = 9\%$ for the *a posteriori* probability that a positive-testing patient has the disease.

9 Use in computer science for program semantics

9.1 "Elementary" can still be intricate

By *elementary* probability theory we mean discrete distributions, usually over finite sets. Non-elementary would then include measures, and the subtle issues of measurability as they apply to infinite sets. In Sec. 9.2 we illustrate how simple computer programs can require intricate probabilistic reasoning even when restricted to discrete distributions on small finite sets.

The same intricate-though-elementary effect led to the Eindhoven notation in the first place.

A particular example is assignment statements, which are mathematically elementary: functions from state to state. Yet for *specific* program texts those functions are determined by expressions in the program variables, and they leave most of those variables unchanged: working with syntactic substitutions is a better approach [4, 5], but that can lead to complex formulae in the program logic.

Careful control of variable binding, and quantifiers, reduces the risk of reasoning errors in the logic, and can lead to striking simplifications because of the algebra that a systematic notation induces. That is what we illustrate in the following probabilistic example.

9.2 Case study: quantitative noninterference security

In this example we treat noninterference security for a program fragment, based on the mathematical structure of Hidden Markov Models [10, 13, 19].

Suppose we have a "secret" program variable h of type H whose value could be partly revealed by an assignment statement $v{:=}\, exp$ to a visible variable v of type V, if expression exp contains h. Although an attacker cannot see h, he can see v's final value, and he knows the program code (i.e. he knows the text of exp).

Given some known initial distribution δ in $\mathbb{D}H$ of h, how do we express what the attacker learns by executing the assignment, and how might we quantify the resulting security vulnerability? As an example we define $\delta{=}\{\!|0,1,2|\!\}$ to be a distribution on h in $H{=}\{0,1,2\}$, with $v{:=}\, h \,\mathbf{mod}\, 2$ assigning its parity to v of type $V{=}\{0,1\}$.

The output distribution over V that the attacker observes in variable v is

$$\{\!|h{:}\,\delta \bullet exp|\!\}\ , \tag{5}$$

thus in our example $\{\!|h{:}\,\{\!|0,1,2|\!\} \bullet h \,\mathbf{mod}\, 2|\!\}$. It equals $0 \,{}_{2/3}\!\oplus 1$, showing that the attacker will observe $v{=}0$ twice as often as $v{=}1$.

The attacker is however not interested in v itself: he is interested in h. When he observes v=1 what he learns, and remembers, is that definitely h=1. But when v=0 he learns "less" because the (*a posteriori*) distribution of h in that case is $\{\!\{0,2\}\!\}$. In that case he is still not completely sure of h's value.

In our style, for the first case v=1 the *a posteriori* distribution of h is given by the conditional distribution $\{\!\{h\colon \{\!\{0,1,2\}\!\} \mid h \bmod 2 = 1\}\!\} = \{\!\{1\}\!\}$; in the second case it is however $\{\!\{h\colon \{\!\{0,1,2\}\!\} \mid h \bmod 2 = 0\}\!\} = \{\!\{0,2\}\!\}$; and in general it would be $\{\!\{h\colon \{\!\{0,1,2\}\!\} \mid h \bmod 2 = v\}\!\}$ where v is the observed value, either 0 or 1.

If in the example the attacker forgets v but remembers what he learned about h, then 2/3 of the time he remembers that h has distribution $\{\!\{0,2\}\!\}$, i.e. is equally likely to be 0 or 2; and 1/3 of the time he remembers that h has distribution $\{\!\{1\}\!\}$, i.e. is certainly 1. Thus what he remembers about h is

$$\{\!\{0,2\}\!\} \quad {}_{2/3}\!\oplus \quad \{\!\{1\}\!\} \;, \tag{6}$$

which is a distribution of distributions. [12] In general, what he remembers about h is the distribution of distributions Δ given by

$$\Delta \quad := \quad \{\!\{v\colon \{\!\{h\colon \delta \bullet \mathit{exp}\}\!\} \bullet \{\!\{h\colon \delta \mid \mathit{exp}{=}v\}\!\}\}\!\} \;, \tag{7}$$

because v itself has a distribution, as we noted at (5) above; and then the *a posteriori* distribution $\{\!\{h\colon \delta \mid \mathit{exp}{=}v\}\!\}$ of h is determined by that v. The attacker's lack of interest in v's actual value is reflected in v's not being free in (7).

We now show what the attacker can do with (7), his analysis Δ of the program's meaning: if he guesses optimally for h's value, with what probability will he be right? For v=0 he will be right only half the time; but for v=1 he will be certain. So overall his attack will succeed with probability $\frac{1}{2}\ {}_{2/3}\!\oplus\ 1 = \frac{2}{3}\times\frac{1}{2} + \frac{1}{3}\times 1 = 2/3$, obtained from (6) by replacing the two distributions with the attacker's "best guess probability" for each, the maximum of the probabilities in those distributions. We say that the "vulnerability" in this example is 2/3.

For *vulnerability* in general take (7), apply the "best guess" strategy and then average over the cases: it becomes $(\mathcal{E}\eta\colon \Delta \bullet (\mathbf{max}\, h\colon H \bullet \eta.h))$, that is the maximum probability in each of the "inner" distributions η of Δ, averaged according to the "outer" probability Δ itself assigns to each. [13]

It is true that (7) appears complex if all you want is the information-theoretic vulnerability of a single assignment statement. But a more direct expression for that vulnerability is not compositional for programs generally; we need Δ-like semantics from which the vulnerability can subsequently be calculated, because they contain enough additional information for composition of meanings. We show elsewhere that (7) is necessary and sufficient for compositionality [18].

[12] In other work, we call this a *hyperdistribution* [18–20].
[13] This is the *Bayes Vulnerability* of Δ [27].

9.3 Comparison with conventional notation

Given the assignment statement $v{:=}\, exp$ as above, define random variables F for the function exp in terms of h, and I for h itself (again as a function of h, i.e. the identity).

Then we determine the observed output distribution of v from the input distribution δ of h by the push-forward of $F_*(\delta)$, from Sec. 5.3, of F over δ.

Then define function g^δ, depending on h's initial distribution δ, that gives for any value of v the conditioning of δ by the event $F{=}v$. That is $g^\delta(v){:=}\, \Pr(I|F{=}v)$ where the Pr on the right refers to δ.

Finally, the output hyperdistribution (7) of the attacker's resulting knowledge of h is given by the push-forward $g^\delta_*(F_*(\delta))$ of g^δ over $F_*(\delta)$ which, because composition distributes through push-forward, we can rewrite as $(g^\delta \mathbin{\circ} F)_*(\delta)$.

An analogous treatment of (7) is given at (9) below, where superscript δ in g^δ here reflects the fact that δ is free in the inner comprehension there.

9.4 Comparison with *qualitative* noninterference security

In a qualitative approach [22, 23] we would suppose a *set* $H{:=}\,\{0,1,2\}$ of hidden initial possibilities for h, not a distribution of them; and then we would execute the assignment $v{:=}\, h \;\mathbf{mod}\; 2$ as before. An observer's deductions are described by the set of sets $\{\{0,2\},\{1\}\}$, a demonic choice between knowing $h{\in}\{0,2\}$ and knowing $h{=}1$. The general $v{:=}\, exp$ gives $\{v{:}\,\{h{:}\,H \bullet exp\} \bullet \{h{:}\,H \mid exp{=}v\}\}$, which is a qualitative analogue of (7). [14]

With the (extant) Eindhoven algebra of *set* comprehensions, and some calculation, that can be rewritten

$$\{h{:}\,H \bullet \{h'{:}\,H \mid exp{=}exp'\}\}\;, \tag{8}$$

where exp' is $exp[h\backslash h']$. It is the partition of H by the function $(\boldsymbol{\lambda} h{:}\,H \bullet exp)$. Analogously, with the algebra of *distribution* comprehensions (see (14) below) we can rewrite (7) to

$$\{\!\{h{:}\,\delta \bullet \{\!\{h'{:}\,H \mid exp{=}exp'\}\!\}\}\!\} \tag{9}$$

The occurrence of (8) and others similar, in our earlier qualitative security work [21, App. A], convinced us that there should be a *probabilistic* notational analogue (9) reflecting those analogies of meaning. This paper has described how that was made to happen; in retrospect, however, the conclusion was forgone — the more abstract view of these notations is via monads, which we now explore.

[14] Written conventionally that becomes $\{\{h{\in}H \mid exp{=}v\} \mid v{\in}\{exp \mid h{\in}H\}\}$, where the left- and right occurrences of "|" now have different meanings. And then what does the middle one mean?

10 Synthesis of the comprehension notation

10.1 Monads, and Haskell's comprehensions

Giry's, and Lawvere's seminal work [16, 9] defined a probability functor on the category of measurable spaces and measurable maps between them: it took any object to the probability measures on it (and perforce made that a measurable space itself).

Wadler [28], inspired by earlier work of Moggi, introduced monads into functional programming as a way of mimicking so-called "impure" features within a pure functional style; and he suggested that for every monad there is a corresponding comprehension notation, one that gives the already known list comprehensions as an instance. In order to handle filters he introduced the idea of *zero* for a monad [ibid., Sec. 5], named by analogy with *unit*, noting however that not all monads have zeroes.

We put these together by specialising Giry's work to discrete distributions, as used in earlier sections, and then following Wadler's construction to synthesise a definition of distribution comprehension within a functional language: we find that it agrees with what we did above. A first problem along the way, however, is that the probability monad is one of those noted by Wadler to have no zero; but that is solved by moving to the *sub*-distributions that sum to *no more than* one (and indeed can sum even to zero exactly). [15]

A second problem arises if we want to mimic our generalisation of conditional- to a-posteriori constructions, since then (as we saw in Sec. 8) the type of "zero" must be more general than Boolean; we solve that by moving from subdistributions to vector spaces where coefficients can sum to any real value at all.

We begin by reviewing and setting the notation for vector spaces (Sec. 10.2), at the same time showing their straightforward relation to distributions, our actual topic. Then we review Wadler's work, introducing Haskell-like notation as necessary (Sec. 10.3ff).

Because our aim is conceptual (i.e. demonstrating the systhesis of a mathematical definition), we do not insist on executable code — indeed executable probabilistic monads have been proposed some time ago [25, 6, 14], recently [8] and even very recently [29]. Rather we use Haskell-like notation to make it easy to compare what we do here with what Wadler did before; in particular, we use Haskell comprehensions (of course) rather than their modern replacement (`do`-notation), we use an actual real-number type $\mathbb{R}$ (rather than e.g. `Rational`, `Float` or `Real`) and we define monads by `unit` and `join` rather than (equivalently) by `return` and `>>=` (bind).

10.2 The monad of vector spaces

For a fixed field K the category K-Vect has vector spaces over K as objects and K-linear transformations as morphisms. If we take K to be the reals $\mathbb{R}$, then

[15] We thank Jeremy Gibbons for this observation. In our other work [24, 17, 3] we use subdistributions to describe non-termination: but we must forgo that here.

we can form the category $\mathbb{R}$-Vect that represents real-valued multisets. We call that category Vec. Each of its objects is the ($\mathbb{R}$-) vector space over some basis B, which we denote by $\mathbb{V}B$. We refer to elements of (a particular) $\mathbb{V}B$ as its *vecs*. For any vec $v\colon \mathbb{V}B$ the co-ordinates it assigns to elements of b are v's *weights*, and we write $v.b$ for that, by analogy with function application. The overall *weight* of a vec, i.e. as a whole, is the sum of the weights it assigns individually, when that is finite.

Obvious specialisations of vecs are

- to *sets*, where a vec's weights are in $\{0,1\}$,
- to *subdistributions*, where weights are in $[0,1]$ and sum to no more than 1,
- to *(total) distributions*, where weights are in $[0,1]$ and sum to exactly 1 and
- to (conventional) *multisets*, where weights are always (non-negative) integers.

In each case we are taking the full subcategory whose objects are constrained by the specialisation imposed.

For sets we usually write $\mathbb{P}$ rather than $\mathbb{V}$ for the object constructor; similarly, earlier in this article we have been using $\mathbb{D}$ to construct discrete total distributions. For sets there is the specialised Boolean notation $(b{\in})$ for testing b's membership of a set s; but we can also use the vec-style notation $(.b)$, so that $s.b{:=}[b{\in}s]$. [16] In all cases we continue with the notation $\lceil v \rceil$, the *support* of $v\colon \mathbb{V}B$, meaning those elements of its basis to which it assigns a nonzero value, that is $\{b\colon B \mid v.b{\neq}0\}$. [17]

In category Set we know that $\mathbb{P}$ is a functor: for morphism $f\colon B{\to}A$ we define $\mathbb{P}f\colon \mathbb{P}B \to \mathbb{P}A$ by $b \in \mathbb{P}f.B' \;:=\; (\exists b\colon B' \mid f.b{=}a)$. We know also that $\mathbb{P}$ forms a monad when equipped with the two natural transformations *unit* η and *multiply* μ as follows:

$$\begin{array}{lll} \eta\colon B{\to}\mathbb{P}B & & \text{— unit} \\ b' \in \eta.b & := & b{=}b' \end{array}$$

$$\begin{array}{lll} \mu\colon \mathbb{P}^2B{\to}\mathbb{P}B & & \text{— multiply} \\ b' \in \mu.\mathcal{B} & := & (\exists B'\colon \mathbb{P}B \mid b'{\in}B' \wedge B'{\in}\mathcal{B})\ . \end{array}$$

It is because these functions satisfy the following conditions that we can say that $(\mathbb{P}, \eta, \mu)$ is a monad:

1. $\mathbb{P}\mathbf{1} \;=\; \mathbf{1}$	$\mathbf{1}$ is the identity
2. $\mathbb{P}(g \circ f) \;=\; \mathbb{P}g \circ \mathbb{P}f$	$\circ$ is functional composition
3. $\mathbb{P}f \circ \eta \;=\; \eta \circ f$	
4. $\mathbb{P}f \circ \mu \;=\; \mu \circ \mathbb{P}^2 f$	
5. $\mu \circ \eta \;=\; \mathbf{1}$	
6. $\mu \circ \mathbb{P}\eta \;=\; \mathbf{1}$	
7. $\mu \circ \mu \;=\; \mu \circ \mathbb{P}\mu$	

[16] Recall that the square brackets $[\cdot]$ convert a Boolean to 1,0.
[17] On sets the support function is therefore the identity.

Laws 1–2 hold because $\mathbb{P}$ is a functor; Laws 3–4 hold because η and μ are natural transformations; and the remaining Laws 5–7, the *Coherence Conditions*, are specific to monads.

The same structure obtains for category Vec, where $(\mathbb{V}, \eta, \mu)$ is a monad when we define

$$\begin{array}{lcll} \mathbb{V}f.B'.a & := & (\Sigma b\colon B \mid f.b{=}a \bullet B'.b) & \text{— functor}\,[18] \\ & & & \\ \eta\colon B{\to}\mathbb{V}B & & & \text{— unit} \\ \eta.b.b' & := & [b{=}b'] & \\ & & & \\ \mu\colon \mathbb{V}^2B{\to}\mathbb{V}B & & & \text{— multiply} \\ \mu.\mathcal{B}.b' & := & (\Sigma B'\colon \mathbb{V}B \mid \mathcal{B}.B' \times B'.b')\ . & \end{array} \tag{10}$$

These definitions also satisfy Conditions 1–7 above, once we replace $\mathbb{P}$ by $\mathbb{V}$; and they apply equally well to (sub)distributions, in the sense e.g. that if B' is a subdistribution then so is $\mathbb{V}f.B'$ etc. They carry through to sets as well, if we replace summation by disjunction.

10.3 Haskell-style comprehensions for monads

We now review Wadler's introduction of monads into functional programming, specifically into Haskell [26], as it would apply to $\mathbb{V}$; we then review his derivation of comprehensions from them. As mentioned in Sec. 10.1, we must solve two small problems: that the probability monad has no zero with which to make filters [8]; and that anyway we want filters that are more general than Boolean (Sec. 8). [19]

We begin by defining a constructor in the Haskell style, but in the manner of an abstract datatype we give its specification rather than a specific implementation: it is `type Vec b = b->`$\mathbb{R}$, to represent the $\mathbb{V}$ functor applied to basis b. However `Vec` might actually be implemented, we assume there is a function

```
supp :: Vec b -> [b]
```

implementing the support function $\lceil\cdot\rceil$ described above, returning its result as a (finite) list. [20] As a convenient way of making specific distributions, we introduce

[18] It needs to be verified that $\mathbb{V}f$ is not just any function but a linear transformation, and that the functoriality requirements $\mathbb{V}f.\mathbf{1} = \mathbf{1}$ and $\mathbb{V}(f \circ g) = \mathbb{V}f \circ \mathbb{V}g$ are met.

[19] Furthermore, monads are required to be *endofunctors* which, in our case, means that our vector-space bases themselves have to be (other) vector spaces: thus they cannot be finite sets (like die-roll outcomes), since our field $\mathbb{R}$ is infinite. This seems however not to bother Haskell programmers, since their use of monads exploits, primarily, the algebra of their unit, multiply and associated derived operations. A more thorough treatment of the endo-issue is given by Altenkirch et al. [1].

[20] Even if we replaced the ideal $\mathbb{R}$ by the implementable `Rational`, say, the function `supp` cannot be implemented in general. We do it this way to concentrate on the mathematical properties of distributions rather than how they might be represented (say as lists of (value,probability) pairs).

also the function

```
fromList :: [b] -> Vec b
fromList bs b' = sum [w | (b,w)<-bs, b==b'] .
```

The functor/monad (class-instance) structure (10) is then supplied by these definitions:

$$\begin{array}{lcl}
\texttt{map f v b'} & = & \texttt{sum [v b | b<- supp v, f b = b']} \\
 & & \quad \text{for } \texttt{f :: B->A} \text{ and } \texttt{v :: Vec B} \text{ and } \texttt{b' :: B} \\
\texttt{unit b b'} & = & \texttt{if b==b' then 1 else 0} \\
\texttt{join vv b'} & = & \texttt{sum [vv v * v b'| v<- supp vv]} \\
 & & \quad \text{for } \texttt{vv :: Vec(Vec B)} \text{ and } \texttt{b' :: B} \,.
\end{array} \tag{11}$$

We have used the names `map` for application of a functor to a morphism, and similarly `join` for multiply μ.[21] As Wadler showed, these functions enable a comprehension notation for monadic values to be defined by a syntactic de-sugaring.

In Haskell, a comprehension has the form `[tTerm|qQuals]`, where `tTerm` is a *term* and `qQuals` is a comma-separated list of *qualifiers*, possibly empty. (If the qualifier list is empty, we can omit the separator "`|`".) The terms have some monadic type, the qualifiers are either bindings or Boolean expressions and the de-sugaring is done as follows:[22] we define

1. `[tTerm]` = `unit tTerm`
2. `[tTerm| x<-uTerm]` = `map (\x-> tTerm) uTerm`
3. `[tTerm | qQuals,rQuals]` = `join [[tTerm|rQuals] | qQuals]` ,

where `x` is a variable and `tTerm`, `uTerm` are terms and `qQuals`, `rQuals` are qualifiers or comma-separated lists of them. The functions `unit`, `map` and `join` are the ones in the style of (11) that are associated with the monad on which the comprehension is based.

10.4 Distribution comprehensions seen monadically

To start with, we concentrate on distributions only (i.e. not the more general vecs); but we use the definitions (11) regardless, because they specialise correctly.

Assume we have a function `uniform :: [b] -> Vec b` defined

```
uniform bs b' = if b' elem bs then 1/length bs else 0 ,
```

[23]

and define `delta` to be `uniform [0,1,2]` for our use below. We note that the composition `supp.uniform` is the identity `id` (or a permutation).

[21] Modern Haskell uses `fmap`; but Wadler at that time used `map`.

[22] In Haskell's presentation of monads the term `tTerm` could be of any type; but mathematically it should be of monadic type because monads are endofunctors.

[23] If we are worried about repeating elements in `bs`, add `let bs' = nub bs in`... But if `b'` is empty or infinite, it is simply an error.

From the comprehension de-sugaring rules above, we see that a Haskell-style distribution comprehension would render our earlier, introductory example from Sec. 5.2 as follows: we rewrite our mathematical expression $\{\!\{s\colon \Delta \bullet s \,\mathbf{mod}\, 2\}\!\}$ as `[s mod 2 | s<- delta]`, and the calculations become

```
    [s mod 2 | s<- delta] 0                              "apply to element 0"
=   map (\s->s mod 2) delta 0                                      "Sugar 2"
=   sum [delta s | s<- supp delta, s mod 2 = 0]              "(11) for map"
=   sum [delta s | s<- [0,1,2], s mod 2 = 0]             "supp.uniform=id"
=   sum [delta 0, delta 2]                                "list comprehension"
=   sum [1/3,1/3]                                          "definition delta"
=   2/3 .
```

and similarly

```
    [s mod 2 | s<- delta] 1                              "apply to element 1"
=     ⋮
=   1/3
```

so that we obtain the distribution `fromList [(0,2/3),(1,1/3)]` as expected.

10.5 Conditional distributions seen monadically

In Sec. 6 we introduced conditioning within distribution comprehensions, basing the notation for it on filters within set comprehensions. Wadler remarks that a monad-based approach to filters is possible in the case that the monad has a useful analogue of the empty list `[]` in the list monad [28, Sec. 5]. For this he adds a fourth "zero" function that we call ζ, in our case of type $A \to \mathbb{V}B$, with the additional conditions

8. $\mathbb{V}f \circ \zeta \;=\; \zeta \circ g$
9. $\mu \circ \zeta \;=\; \zeta$
10. $\mu \circ \mathbb{V}\zeta \;=\; \zeta$

Note that the source type A of ζ is arbitrary, turning what is essentially a zero *element* into a zero-returning function. For example in the list monad, the (constant) function ζ returns the empty list, and for sets it returns the empty set. In all cases, the crucial properties of ζ are

- that it is "empty," so that mapping any function over it returns empty (8.)
- and, since there is "nothing" in it, taking the join of everything in it again returns empty (9.) and
- finally that can be ignored in joins (10.) just as for example 0 can be ignored in additions.

We represent $\zeta\colon A \to \mathbb{V}B$ as `zero :: a->Vec b` and, following Wadler, give the additional comprehension de-sugaring [24]

[24] Wadler does not give this definition exactly: he notates it differently.

4. `[tTerm | bBool]` = `if bBool then unit tTerm else zero tTerm` ,

where `bBool` is a Boolean-typed term.

Unfortunately, we cannot specialise this to (full) distributions because for vecs it should return the everywhere-zero element that has empty support, and the all-zero element of a vector space (the origin, in effect) does not sum to one as a distribution requires.

Fortunately, we can find a zero element if we consider *sub*-distributions, those summing to *no more than* one. We therefore define

$$\texttt{zero a b' = 0}\,, \tag{12}$$

so that for any element `b' :: b` the "probability" (more generally, weight) of `b'` in `zero a` is zero.

For our first example we use the definition in Sec. 6.2 of conditional-distribution comprehensions; translated, this is

```
    [s | s<- delta, s/=0] s'                                  "translation"
=   join vv s' where vv = [[s|s/=0] | s<- delta]              "Sugar 3 "
=   sum [vv v * v s' | v<- supp vv] where...          "defn. (11) of join"

=                          "supp [tTerm| x<-uTerm] = [tTerm| x<-supp uTerm]"
    sum [vv v * v s' | v<- [[0|0/=0],[1|1/=0],[2|2/=0]]]

=   1/3 * ([0|False] s' + [1|True] s' + [2|True] s')     "defn. vv, sum"
=   1/3 * (zero 0 s' + unit 1 s' + unit 2 s')                   "Sugar 4"
=   if s'==1 || s'==2 then 1/3 else 0 .          "(11) unit and (12) zero"
```

Once we abstract from `s'`, we have the subdistribution over $\{0,1,2\}$ that returns $1/3$ for values $1,2$ and 0 for everything else, that is

```
fromList [(1,1/3),(2,1/3)] .
```

This is of course not exactly what we hoped for, since our earlier result of this conditioning was $\{\!|s{:}\,\Delta \mid s{\neq}0|\!\} = \{\!|1,2|\!\}$, the uniform distribution over $\{1,2\}$ — we might have preferred therefore to obtain `uniform [1,2]`.

We have lost the re-normalisation: we see that this general approach to conditioning, as filters, has indeed forced us out of the distributions into the more general subdistributions.

10.6 To normalise, or not to normalise?

We decide *not* to normalise, at this stage, and so to make a virtue of the necessity revealed at the end of the previous section. [25] That is, we concentrate on vecs in their own right, thinking of them as *samples* such as "in 106 out of 206 live human births, the child is male." A (full, one-summing) probability distribution we see as an abstraction from a sample, one in which the arithmetic of normalisation has been done to make subsequent comparisons and calculations easier. We return to normalisation in Sec. 10.8.

[25] Translation: it's a *feature*.

10.7 Belief-revision revised

In Sec. 8 we discovered that the notation for conditional-distribution comprehensions easily extended to belief revision, vis. the conversion of an a-priori distribution into an a-posteriori distribution on the basis of an experiment. This was done by allowing the filter (as we call it above, but in Sec. 8 it was called the "range") to take numeric- rather than only Boolean values. That done, we found that Def. 12, given originally for conditional distributions, calculated the a-posteriori value automatically as a generalisation of conditioning.

We can to some extent replicate this generalisation in the monad-based approach we are now following.

Rather than introduce a zero-function ζ, as we did for conditionals, we introduce a scaling function $\sigma\colon \mathbb{R}\to\mathbb{V}B\to\mathbb{V}B$ that generalises the identity. For $b\colon B$ and $v\colon \mathbb{V}B$ and $r\colon \mathbb{R}$ we define $\sigma.r.v.b := r\times v.b$; since the identity $\mathbf{1}$ is then $\sigma.1$, by analogy we will write $\sigma.r$ as (boldface) $\boldsymbol{r}$ where it is not ambiguous: the scaling functions have the properties

11. $\mathbb{V}f\circ\boldsymbol{r} \;=\; \boldsymbol{r}\circ\mathbb{V}f$
12. $\mu\circ\boldsymbol{r} \;=\; \boldsymbol{r}\circ\mu$
13. $\mu\circ\mathbb{V}\boldsymbol{r} \;=\; \boldsymbol{r}\circ\mu$,

and ζ (`zero`) can be defined `unit` followed by 0-scaling, that is $\zeta := \mathbf{0}\circ\eta$.

With the monad conditions 1.–7. from Sec. 10.2, we can use the above to prove (rather than postulate) the properties 8.–10. introduced in Sec. 10.5 for ζ.

We now translate function $\sigma.r$, that is $\boldsymbol{r}$, into `scale r` defined

```
scale r v b' = r * v b' ,                                   (13)
```

so that we can use `scale` in our construction of comprehensions: we redefine the unsugaring of filter for this more general case to give

4'. `[tTerm| rReal]` = `scale rReal (unit tTerm)` ,

so that the original Definition 4 for `[tTerm | bBool]` becomes the special case where the real-valued expression `rReal` is restricted to 1 (true) or 0 (false), i.e. is effectively Boolean.

The belief-revision example from Sec. 8.3 becomes

```
    [b | b<- delta, b/2] b'                    "translation {{b: Δ | b/2}}"
=   join vv b' where vv = [[b|b/2] | b<- delta]          "Sugar 3 "
=   sum [vv v * v b' | v<- supp vv] where...     "defn. (11) of join"

=                          "supp [tTerm| x<-uTerm] = [tTerm| x<-supp uTerm]"
    sum [vv v * v b' | v<- [[0|0/2],[1|1/2],[2|2/2]]]

=   1/3 * ([0|0] b' + [1|1/2] b' + [2|1] b')          "defn. vv, sum"
```

```
=  1/3 * (  scale 0 (unit 0 b')                          "Sugar 4'"
         + scale (1/2) (unit 1 b')
         + scale 1 (unit 2 b'))

=  case b' of { 1-> 1/6; 2-> 1/3; otherwise 0 } .        "(11) unit
                                                          and (13) scale"
```

Again we have a proper subdistribution, thus needing normalisation to agree with Sec. 8.3. But even without normalisation, we can see that the probability the other ball is white (1/3 unnormalised, when `b'` is 2) is twice the probability that it is black (1/6).

10.8 Normalisation and synthesis

In Sec. 10.6 we remarked that the use of filters in comprehensions can produce subdistributions, and that the the conventional (full) distributions can be recovered by normalisation:

```
normalise v        =  scale (1/weight v) v
  where weight v   =  sum [v b | b<- supp v]
```

An explicit use of normalisation is advocated by Hehner, as noted in Sec. 11.

We therefore conclude by using the above systematic procedure, and subsequent normalisation, to synthesise our earlier Def. 13 in Sec. 8.5. With the obvious re-ordering of terms, it is simply

$$\{\!|\, s{:}\delta \mid rng \bullet exp \,|\!\} \quad = \quad \texttt{normalise [exp | s<- delta, rng]} \ .$$

11 Related work

As the previous section remarked, the structure of the Eindhoven notation is monadic: for distributions it is the Giry monad Π on a category Mes of measurable spaces, with measurable maps as its morphisms [9]; for sets, it is the powerset monad $\mathbb{P}$ on Set. That accounts for many similarities, among which is the resemblance between (8) and (9).

Among the presentations of probabilistic monads in functional programming, there are a number of packages that have been put together on that basis [25, 6, 14, 8, 29]. The goal of those is mainly to enable probabilistic functional programming, with some emphasis also on a notation for reasoning; our goal here was not to repeat that work, but rather to suggest a comprehension notation compatible with it, one moreover especially suited to expressing the denotational semantics of security-based programs.

A notable example of other related work, but with a different background, is Hehner's *Probabilistic Perspective* [12]. A distribution there is an expression whose free variables range over a separately declared sample space: for each assignment of values to the free variables, the expression gives the probability

of that assignment as an observation: thus for $n{:}\,\mathbb{N}^+$ the expression 2^{-n} is an example of a geometric distribution on the positive integers.

With a single new operator $\updownarrow$, for normalisation, and existing programming-like notations, Hehner reconstructs many familiar artefacts of probability theory (including conditional distributions and *a posteriori* analyses), and convincingly demystifies a number of probability puzzles, including some of those treated here. In Sec. 10.8, also we treated normalisation explicitly.

A strategic difference between our two approaches is (we believe) that Hehner's aim is in part to put elementary probability theory on a simpler, more rational footing; we believe he succeeds. In the sense of our comments in Sec. 1, he is working "forwards." As we hope Sec. 9 demonstrated, we started instead with existing probabilistic constructions (essentially Hidden Markov Models as we explain elsewhere [19]), as a program semantics for noninterference, and then worked backwards towards the Eindhoven quantifier notation. One of the senses in which we met Hehner "in the middle" is that we both identify discrete distributions as first-class objects, for Hehner a real-valued expression over free variables of a type and for us a function from that type into the reals.

In conventional approaches to probability theory that explicit treatment of distributions, i.e. giving them names and manipulating them, does not occur until one reaches either proper measures or Markov chains. For us it is (in spirit) the former; we believe part of Hehner's approach can be explained in terms of the latter.

A technical difference is our explicit treatment of free- and bound variables, a principal feature of the Eindhoven notation and one reason we chose it.

12 Summary and prospects

We have argued that Eindhoven-style quantifier notation simplifies many of the constructions appearing in elementary probability. As evidence for this we invite comparison of the single expression (9) with the paragraphs of Sec. 9.3. We have also shown how the notation arises naturally from a monadic viewpoint.

There is no space here to give a comprehensive list of calculational identities; but we mention two of them as examples of how the underlying structure mentioned above (Sec. 10) generates equalities similar to those already familiar from the Eindhoven notation applied to sets.

One identity is the trading rule

$$\begin{array}{ll} & (\mathcal{E}s{:}\,\delta \mid rng' \bullet \{\!|\, exp' \,|\!\} \mid rng \bullet exp) \\ = & (\mathcal{E}s{:}\,\delta \mid rng' \times rng[s\backslash exp'] \bullet exp[s\backslash exp']) \ , \end{array} \tag{14}$$

so-called because it "trades" components of an inner quantification into an outer one. Specialised to defaults for *true* for *rng* and s for exp', it gives an alternative to Def. 11. An identity similar to this took us from (7) to (9).

A second identity is the one used in Sec. 8.4, that $(\mathcal{E}s{:}\,\delta; s'{:}\,\delta' \mid rng \bullet exp)$ equals $(\mathcal{E}s{:}\,\delta \mid (\mathcal{E}s'{:}\,\delta' \bullet rng) \bullet exp)$ when s' is not free in exp. As noted there, this corresponds to a similar trading rule between set comprehension and existential

quantification: both are notationally possible only because variable bindings are explicitly given *even when those variables are not used*. This is just what the Eindhoven style mandates.

The notations here generalise to (non-discrete) probability measures, i.e. even to non-elementary probability theory, again because of the monadic structure. For example the integral of a measurable function given as expression *exp* in a variable s on a sample space S, with respect to a measure μ, could conventionally be written $\int exp\, \mu(\mathrm{d}s)$.[26] We write it however as$(\mathcal{E}s{:}\,\mu \bullet exp)$, and have access to (14)-like identities such as

$$(\mathcal{E}s{:}\,[\![s'{:}\,\mu \bullet exp']\!] \bullet exp) \quad = \quad (\mathcal{E}s'{:}\,\mu \bullet exp[s\backslash exp])\ .$$

(See App. A for how this would be written conventionally for measures.)

We ended in Sec. 9 with an example of how the notation improves the treatment of probabilistic computer programs, particularly those presented in a denotational-semantic style and based on *Hidden Markov Models* for quantitative noninterference security [13, 19]. Although the example concludes this report, it was the starting point for the work.

Acknowledgements Jeremy Gibbons identified functional-programming activity in this area, and shared his own recent work with us. Frits Vaandrager generously hosted our six-month stay at Radboud University in Nijmegen during 2011. The use of this notation for security (Sec. 9.2) was in collaboration with Annabelle McIver and Larissa Meinicke [18, 19, and others].

Roland Backhouse, Eric Hehner, Bart Jacobs and David Jansen made many helpful suggestions on the original paper; in particular Jansen suggested looking at continuous distributions (i.e. those given as a density function).

For this revision, I thank members of IFIP WG 2.1, in particular the members of the "problem-solving group" at the Ottawa meeting (Lambert Meertens, Jeremy Gibbons, Henk Boom, Fritz Henglein), for ideas on how to introduce filters into distribution comprehensions in a monadic way. Separately, Phil Wadler made many suggestions for Sec. 10, proposing an implementation of the monad with which he was able to code many of the paper's examples [29].

Finally, Lambert Meertens wrote extensive and detailed comments on the whole paper: they were received just the day before the world was due to end. If you are reading this, it did not.

References

1. Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors, 2012. Available at `citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.156.7931`.

[26] Or not? We say "could" because "[there] are a number of different notations for the integral in the literature; for instance, one may find any of the following: $\int_Y s\,\mathrm{d}\mu$, $\int_Y s(x)\,\mathrm{d}\mu$, $\int_Y s(x)\mu$, $\int_Y s(x)\mu(\,\mathrm{d}x)$, or even $\int_Y s(x)\,\mathrm{d}x\ldots$" [2].

2. Steve Cheng. A crash course on the Lebesgue integral and measure theory. `www.gold-saucer.org/math/lebesgue/lebesgue.pdf`, downloaded Dec. 2011.
3. Y. Deng, R. van Glabbeek, M. Hennessy, C.C. Morgan, and C. Zhang. Characterising testing preorders for finite probabilistic processes. In *Proc LiCS 07*, 2007.
4. E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
5. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer, 1990.
6. Martin Erwig and Steve Kollmansberger. Probabilistic functional programming in Haskell. *Journal of Functional Programming*, 16:21–34, 2006.
7. D.H. Fremlin. *Measure Theory.* Torres Fremlin, 2000.
8. Jeremy Gibbons and Ralf Hinze. Just do it: simple monadic equational reasoning. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *ICFP*, pages 2–14. ACM, 2011.
9. M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85. Springer, 1981.
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. Ian Hayes. *Specification Case Studies.* Prentice-Hall, 1987. `http://www.itee.uq.edu.au/~ianh/Papers/SCS2.pdf`.
12. Eric C. R. Hehner. A probability perspective. *Form. Asp. Comput.*, 23:391–419, July 2011.
13. D. Jurafsky and J.H. Martin. *Speech and Language Processing.* Prentice Hall International, 2000.
14. Oleg Kiselyov and Chung-Chieh Shan. Embedded probabilistic programming. In Walid Taha, editor, *Domain-Specific Languages*, volume 5658 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2009.
15. E Kowalski. Measure and integral. `www.math.ethz.ch/~kowalski/measure-integral.pdf`, downloaded Dec. 2011.
16. F.W. Lawvere. The category of probabilistic mappings, 1962. (Preprint).
17. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Tech Mono Comp Sci. Springer, New York, 2005.
18. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Compositional closure for Bayes Risk in probabilistic noninterference. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming: Part II*, volume 6199 of *ICALP'10*, pages 223–235, Berlin, Heidelberg, 2010. Springer.
19. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Hidden-Markov program algebra with iteration. At arXiv:1102.0333v1; to appear in Mathematical Structures in Computer Science in 2012, 2011.
20. Annabelle McIver, Larissa Meinicke, and Carroll Morgan. A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In *Proc. LiCS 2012*, 2012.
21. Carroll Morgan. Compositional noninterference from first principles. *Formal Aspects of Computing*, pages 1–24, 2010. 10.1007/s00165-010-0167-y.
22. 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.*
23. C.C. Morgan. *The Shadow Knows:* Refinement of ignorance in sequential programs. *Science of Computer Programming*, 74(8):629–653, 2009. Treats *Oblivious Transfer.*

24. 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.
25. Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. *SIGPLAN Not.*, 37:154–165, January 2002.
26. Haskell Report. www.haskell.org/onlinereport/haskell2010/, 2010.
27. 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.
28. Philip Wadler. Comprehending monads. In *Mathematical Structures in Computer Science*, pages 61–78, 1992.
29. Philip Wadler. Private communication. 2013.

Appendices

A Measure spaces

More general than discrete distributions, *measures* are used for probability over infinite sample spaces, where expected value becomes integration [7]. Here we sketch how "measure comprehensions" might appear; continuous distributions would be a special case of those.

In Riemann integration we write $\int_a^b x^2 \,\mathrm{d}x$ for the integral of the real-valued squaring-function $sqr := (\boldsymbol{\lambda} x \bullet x^2)$ over the interval $[a, b]$, and in that notation the x in x^2 is bound by the quantifier $\mathrm{d}x$. The scope of the binding is from $\int$ to $\mathrm{d}x$.

In Lebesgue integration however we write $\int sqr \,\mathrm{d}\mu$ for the integral of that same function over a measure μ.

The startling difference between those two notations is the use of of the concrete syntax "d" that in Riemann integration's $\mathrm{d}x$ binds x, while for measures the μ in $\mathrm{d}\mu$ is free. To integrate the expression form of the squaring-function over μ we have to bind its x in another way: two typical approaches are $\int x^2 \,\mu(\mathrm{d}x)$ and $\int x^2 \,\mathrm{d}\mu(x)$ [2]. [27]

An alternative is to achieve some uniformity by using $\mathrm{d}(\cdot)$ in the same way for both kinds of integrals [17]. We use $\int exp \,\mathrm{d}x$ for $\int (\boldsymbol{\lambda} x \bullet exp)$ in all cases; and the measure, or the bounds, are always found *outside* the expression, next to the integral sign $\int$. Thus we write $\int_\mu(\cdot)$ for integration over general measure μ, and then the familiar $\int_a^b(\cdot)$ is simply a typographically more attractive presentation of the special case $\int_{[a,b]}(\cdot)$ over the uniform measure on the real interval $[a, b]$. [28]

Then with $f := (\boldsymbol{\lambda} x \bullet F)$ we would have the equalities

$$\int_{\{\!\{c\}\!\}} F \,\mathrm{d}x \quad = \quad \int_{\{\!\{c\}\!\}} f \quad = \quad f.c \qquad \text{one-point rule}$$

and

$$\int_{g_*.\mu} F \,\mathrm{d}x \quad = \quad \int_{g_*.\mu} f \quad = \quad \int_\mu f \circ g \qquad \begin{array}{l}\text{recall push-forward}\\ \text{from Sec. 5.3.}\end{array}$$

In the second case we equate the integral of f, over an (unnamed) measure formed by pushing function g forward over measure μ, with the integral of the functional composition $f \circ g$ over measure μ directly.

For complicated measures, unsuitable as subscripts, an alternative for the integral notation $\int_\mu exp \,\mathrm{d}x$ is the expected value $(\mathcal{E}x{:}\,\mu \bullet exp)$. The one-point rule is then written $(\mathcal{E}x{:}\,\{\!\{exp\}\!\} \bullet F) = F[x \backslash exp]$. In the second case we have

$$(\mathcal{E}x{:}\,\{\!\{y{:}\,\mu \bullet G\}\!\} \bullet F) \quad = \quad (\mathcal{E}y{:}\,\mu \bullet F[x \backslash G]) \;, \tag{15}$$

[27] And there are more, since "[if] we want to display the argument of the integrand function, alternate notations for the integral include $\int_{x \in X} f(x) \,\mathrm{d}\mu \ldots$" [15].

[28] This is more general than probabiity measures, since the (e.g. Lebesgue) measure $b - a$ of the whole interval $[a, b]$ can exceed one.

where function g has become the lambda abstraction $(\boldsymbol{\lambda} y \bullet G)$. In Lem. 3 below we prove (15) for the discrete case.

B Exploiting non-freeness in the constructor

Here we prove the nontrivial step referred forward from Sec. 8.4: the main assumption is that s' is not free in exp. But should δ itself be an expression, we require that s' not be free there either.

$$
\begin{array}{lll}
 & (\mathcal{E}s{:}\,\delta; s'{:}\,\delta' \mid rng \bullet exp) & \\
= & (\mathcal{E}s{:}\,\delta; s'{:}\,\delta' \bullet rng \times exp) \;/\; (\mathcal{E}s{:}\,\delta; s'{:}\,\delta' \bullet rng) & \text{“Def. 11”} \\
 & & \\
= & \dfrac{(\Sigma s{:}\,S; s'{:}\,S' \bullet \delta.s \times \delta'.s' \times rng \times exp)}{(\Sigma s{:}\,S; s'{:}\,S' \bullet \delta.s \times \delta'.s' \times rng)} & \text{“Def. 7”} \\
 & & \\
= & \dfrac{(\Sigma s{:}\,S \bullet \delta.s \times (\Sigma s'{:}\,S' \bullet \delta'.s' \times rng) \times exp)}{(\Sigma s{:}\,S \bullet \delta.s \times (\Sigma s'{:}\,S' \bullet \delta'.s' \times rng))} & \text{“}s'\text{ not free in }exp\text{”} \\
 & & \\
= & (\mathcal{E}s{:}\,\delta \bullet (\mathcal{E}s'{:}\,\delta' \bullet rng) \times exp) \;/\; (\mathcal{E}s{:}\,\delta \bullet (\mathcal{E}s'{:}\,\delta' \bullet rng)) & \text{“Def. 7”} \\
= & (\mathcal{E}s{:}\,\delta \mid (\mathcal{E}s'{:}\,\delta' \bullet rng) \bullet exp)\ . & \text{“Def. 11”}
\end{array}
$$

C Assorted proofs related to definitions [29]

Lemma 1. *$\{\!|s{:}\,\delta \bullet exp|\!\}$ is a distribution on $\{s{:}\,\lceil\delta\rceil \bullet exp\}$*

Proof: We omit the simple proof that $0 \leq \{\!|s{:}\,\delta \bullet exp|\!\}$; for the one-summing property, we write S for $\lceil\delta\rceil$ and calculate

$$
\begin{array}{lll}
 & (\Sigma e{:}\,\{s{:}\,S \bullet exp\} \bullet \{\!|s{:}\,\delta \bullet exp|\!\}.e) & \text{“let } e \text{ be fresh”} \\
= & (\Sigma e{:}\,\{s{:}\,S \bullet exp\} \bullet (\mathcal{E}s{:}\,\delta \bullet [exp{=}e])) & \text{“Def. 9”} \\
= & (\Sigma e{:}\,\{s{:}\,S \bullet exp\} \bullet (\Sigma s{:}\,S \bullet \delta.s{\times}[exp{=}e])) & \text{“Def. 7”} \\
= & (\Sigma s{:}\,S; e{:}\,\{s{:}\,S \bullet exp\} \bullet \delta.s{\times}[exp{=}e]) & \text{“merge and swap summations”} \\
= & (\Sigma s{:}\,S; e{:}\,\{s{:}\,S \bullet exp\} \mid exp{=}e \bullet \delta.s) & \text{“trading”} \\
= & (\Sigma s{:}\,S \bullet \delta.s) & \text{“one-point rule”} \\
= & 1\ . & \text{“}\delta \text{ is a distribution”}
\end{array}
$$

□

Lemma 2. *$\{\!|s{:}\,\delta \mid rng|\!\}$ is a distribution on $\lceil\delta\rceil$ if $(\mathcal{E}s{:}\,\delta \bullet rng){\neq}0$*

Proof: We omit the simple proof that $0 \leq \{\!|s{:}\,\delta \mid rng|\!\}$; for the one-summing property, we write S for $\lceil\delta\rceil$ and calculate

[29] We thank Roland Backhouse for the suggestion to include the first two of these.

$$
\begin{array}{lll}
 & (\Sigma s' \colon S \bullet \{\!|s \colon \delta \mid rng|\!\}.s') & \text{“let } s' \text{ be fresh”} \\
= & (\Sigma s' \colon S \bullet (\mathcal{E}s \colon \delta \bullet rng \times [s{=}s'])\ /\ (\mathcal{E}s \colon \delta \bullet rng)) & \text{“Def. 10”} \\
= & (\Sigma s' \colon S \bullet (\mathcal{E}s \colon \delta \bullet rng \times [s{=}s']))\ /\ (\mathcal{E}s \colon \delta \bullet rng) & \text{“}s' \text{ not free in denominator”} \\
= & (\mathcal{E}s \colon \delta \bullet rng)\ /\ (\mathcal{E}s \colon \delta \bullet rng) & \text{“one-point rule; Def. 7”} \\
= & 1\ . & \text{“}(\mathcal{E}s \colon \delta \bullet rng){\neq}0\text{”}
\end{array}
$$

□

Lemma 3. $(\mathcal{E}x \colon \{\!|y \colon \delta \bullet G|\!\} \bullet F) = (\mathcal{E}y \colon \delta \bullet F[x \backslash G])$
This is Equation (15) in the discrete case.

Proof: Let X be the support of $\{\!|y \colon \delta \bullet G|\!\}$, for which a more concise notation is given in App. D below, and let Y be the support of δ; we calculate

$$
\begin{array}{lll}
 & (\mathcal{E}x \colon \{\!|y \colon \delta \bullet G|\!\} \bullet F) & \\
= & (\Sigma x \colon X \bullet \{\!|y \colon \delta \bullet G|\!\}.x \times F) & \text{“Def. 7”} \\
= & (\Sigma x \colon X \bullet (\mathcal{E}y \colon \delta \bullet [G{=}x]) \times F) & \text{“Def. 9”} \\
= & (\Sigma x \colon X \bullet (\Sigma y \colon Y \bullet \delta.y \times [G{=}x]) \times F) & \text{“Def. 7”} \\
= & (\Sigma y \colon Y ; x \colon X \bullet \delta.y \times [G{=}x] \times F) & \text{“distribution of summations”} \\
= & (\Sigma y \colon Y \bullet \delta.y \times F[x \backslash G]) & \text{“one-point rule for summation”} \\
= & (\mathcal{E}y \colon \delta \bullet F[x \backslash G])\ . & \text{“Def. 7”}
\end{array}
$$

□

From Lem. 3 we have immediately an analogous equality for distributions, since distribution comprehensions are a special case of expected values: a more succinct, point-free alternative to Def. 9 and Def. 13 is given by the equality

$$\{\!|s \colon \delta \mid rng \bullet exp|\!\} \quad = \quad (\mathcal{E}s \colon \delta \mid rng \bullet \{\!|exp|\!\})\ , \quad {}^{30} \tag{16}$$

where the right-hand expected value is being taken in a vector space (of discrete distributions). This is how we simplified (7) to (9) in Sec. 9.

D Further identities

The identities below are motivated by the first one, i.e. Sec. D.1, justifying the idea that in a comprehension with both range and constructor one can think in terms of enforcing the range as a first step, and then the constructor to what results. The identities are listed in order of increasing generality.

For conciseness in this section we use E^{new}_{old} for substitution and letters R, E instead of words rng, exp for expressions and $\lceil s \colon \delta \mid rng \bullet exp \rceil$ for the support $\lceil \{\!|s \colon \delta \mid rng \bullet exp|\!\} \rceil$.

[30] from $(\mathcal{E}s \colon \delta \mid rng \bullet \{\!|exp|\!\}).e = (\mathcal{E}s \colon \delta \mid rng \bullet \{\!|exp|\!\}.e) = (\mathcal{E}s \colon \delta \mid rng \bullet [exp{=}e])$.

D.1 ——————————

$$
\begin{array}{lll}
 & (\mathcal{E}s{:}\,\{\!|\, s{:}\,\delta \mid R \,|\!\}\bullet E) & \\
= & (\mathcal{E}e{:}\,\{\!|\, s{:}\,\delta \mid R \,|\!\}\bullet E^e_s) & \text{“fresh variable } e\text{”} \\
= & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R\rceil \bullet \{\!|\, s{:}\,\delta \mid R \,|\!\}.e \times E^e_s) & \\
= & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R\rceil \bullet (\mathcal{E}s{:}\,\delta \mid R \bullet [s{=}e]) \times E^e_s) & \\
= & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R\rceil \bullet (\mathcal{E}s{:}\,\delta \bullet R \times [s{=}e]) \times E^e_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \\
= & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R\rceil \bullet \delta.e \times R^e_s \times E^e_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \text{“one-point rule”} \\
= & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R\rceil \bullet \delta.e \times R^e_s \times E^e_s)/(\mathcal{E}s{:}\,\delta \bullet R) & \text{“}e\text{ not free in }R\text{ or }\delta\text{”} \\
= & (\mathcal{E}e{:}\,\delta \bullet R^e_s \times E^e_s)/(\mathcal{E}s{:}\,\delta \bullet R) & \text{“definition }\mathcal{E}\text{ and }\lceil s{:}\,\delta \mid R\rceil\text{”} \\
= & (\mathcal{E}s{:}\,\delta \bullet R \times E)/(\mathcal{E}s{:}\,\delta \bullet R) & \text{“}e\text{ not free in }R, E\text{”} \\
= & (\mathcal{E}s{:}\,\delta \mid R \bullet E)\ . & \text{“Def. 11”}
\end{array}
$$

D.2 —————————— **D.1 for distributions**

$$
\begin{array}{lll}
 & \{\!|\, s{:}\,\{\!|\, s{:}\,\delta \mid R \,|\!\}\bullet E \,|\!\} & \\
= & \{\!|\, s{:}\,\delta \mid R \bullet E \,|\!\}\ . & \text{“from Sec. D.1 under the same conditions, using (16)”}
\end{array}
$$

D.3 ——————————

An elaboration of Sec. D.1 with constructor F, generalising Lem. 3.

$$
\begin{array}{lll}
 & (\mathcal{E}s{:}\,\{\!|\, s{:}\,\delta \mid R \bullet F \,|\!\}\bullet E) & \\
= & & \text{“as for Sec. D.1…”} \\
 & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R \bullet F\rceil \bullet (\mathcal{E}s{:}\,\delta \bullet R \times [F{=}e]) \times E^e_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \\
= & & \text{“…but cannot use one-point wrt. }F\text{”} \\
 & (\Sigma e{:}\,\lceil s{:}\,\delta \mid R \bullet F\rceil \bullet (\Sigma s{:}\,\lceil\delta\rceil \bullet \delta.s \times R \times [F{=}e]) \times E^e_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \\
= & (\Sigma s{:}\,\lceil\delta\rceil; e{:}\,\lceil s{:}\,\delta \mid R \bullet F\rceil \bullet \delta.s \times R \times [F{=}e] \times E^e_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \\
= & (\Sigma s{:}\,\lceil\delta\rceil \bullet \delta.s \times R \times E^F_s/(\mathcal{E}s{:}\,\delta \bullet R)) & \text{“if }\delta.s\text{ and }R\text{ both nonzero,} \\
 & & \text{then } F{\in}\lceil s{:}\,\delta \mid R \bullet F\rceil\text{;} \\
 & & e\text{ not free in }R\text{”} \\
= & (\Sigma s{:}\,\lceil\delta\rceil \bullet \delta.s \times R \times E^F_s)/(\mathcal{E}s{:}\,\delta \bullet R) & \\
= & (\mathcal{E}s{:}\,\delta \bullet R \times E^F_s)/(\mathcal{E}s{:}\,\delta \bullet R) & \\
= & (\mathcal{E}s{:}\,\delta \mid R \bullet E^F_s)\ . &
\end{array}
$$

D.4 —————————— **D.3 for distributions**

$$
\begin{array}{lll}
 & \{\!|\, s{:}\,\{\!|\, s{:}\,\delta \mid R \bullet F \,|\!\}\bullet E \,|\!\} & \\
= & \{\!|\, s{:}\,\delta \mid R \bullet E^F_s \,|\!\}\ . & \text{“from Sec. D.3, under the same conditions”}
\end{array}
$$

D.5 ——————————

An elaboration of Sec. D.3 with range G.

$$
\begin{aligned}
& (\mathcal{E}s\colon \{\!\{s\colon \delta \mid R \bullet F\}\!\} \mid G \bullet E) \\
= \quad & (\mathcal{E}e\colon \{\!\{s\colon \delta \mid R \bullet F\}\!\} \mid G_s^e \bullet E_s^e) \\
= \quad & (\mathcal{E}e\colon \{\!\{s\colon \delta \mid R \bullet F\}\!\} \bullet G_s^e \times E_s^e)/(\mathcal{E}e\colon \{\!\{s\colon \delta \mid R \bullet F\}\!\} \bullet G_s^e) \\
= \quad & (\mathcal{E}s\colon \delta \mid R \bullet G_s^F \times E_s^F)/(\mathcal{E}s\colon \delta \mid R \bullet G_s^F) && \text{“Sec. D.3”} \\
\\
= \quad & \frac{(\mathcal{E}s\colon \delta \bullet R \times G_s^F \times E_s^F)/(\mathcal{E}s\colon \delta \bullet R)}{(\mathcal{E}s\colon \delta \bullet R \times G_s^F)/(\mathcal{E}s\colon \delta \bullet R)} && \text{“if } (\mathcal{E}s\colon \delta \bullet R) \text{ nonzero”} \\
\\
= \quad & (\mathcal{E}s\colon \delta \bullet R \times G_s^F \times E_s^F) \;/\; (\mathcal{E}s\colon \delta \bullet R \times G_s^F) \\
= \quad & (\mathcal{E}s\colon \delta \mid R \times G_s^F \bullet E_s^F)\ .
\end{aligned}
$$

D.6 —————————— **D.5 for distributions**

$$
\begin{aligned}
& \{\!\{s\colon \{\!\{s\colon \delta \mid R \bullet F\}\!\} \mid G \bullet E\}\!\} \\
= \quad & \{\!\{s\colon \delta \mid R \times G_s^F \bullet E_s^F\}\!\}\ . && \text{“from Sec. D.5, under the same conditions”}
\end{aligned}
$$