Nondeterminism, refinement and probability

Annabelle McIver,
Macquarie University,
Sydney

Quick summary

- A short history of probabilistic programming;
- How to build the semantics you want in three easy stages: general model building techniques;
- An application;
- The “refinement paradox”, and how probability can help shed light.

What we want.

Computations
Probability
Nondeterminism
(Refinement;
abstraction)

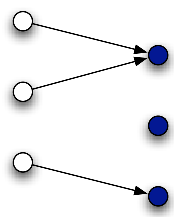
→

Program algebra;
Behavioural model;
Program logic;
Proof techniques;
Compositionality;
Tools.

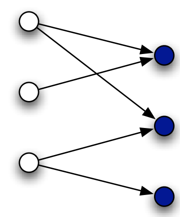
But what does this all mean?
How do these things interact?
What applications are they good for?

Powerdomains

(Originally) a general technique by which a semantic model can be augmented to include nondeterminism in such a way that the underlying computational structure of the original model is maintained.



Functions,
S ⟶ S

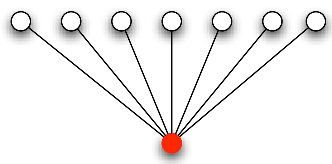


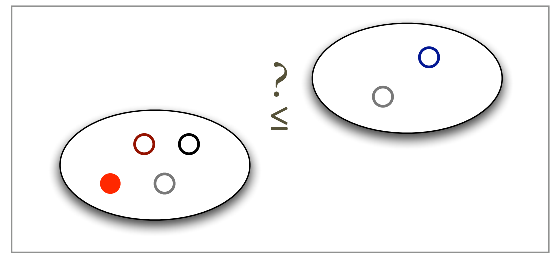
Relations,
S ⟶ PS

Powerdomains

When we want to distinguish nontermination from other behaviour, we introduce a special "bottom state"

How do we order the programs so that is worse than everything, and reducing the range of behaviours corresponds to "more refined".



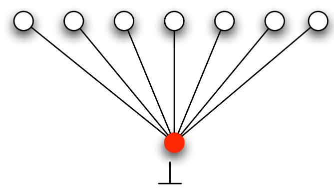


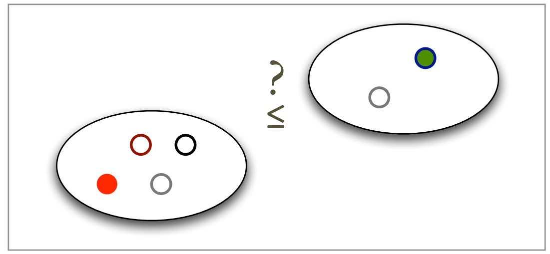
Powerdomains: the Smyth order

$$A \leq_S B \quad \textit{iff} \quad (\forall b \in B(\exists a \in A \cdot a \leq b))$$

In the flat domain, this becomes

$$A \leq_S B \quad \textit{iff} \quad (\bot \in A) \vee (B \subseteq A)$$



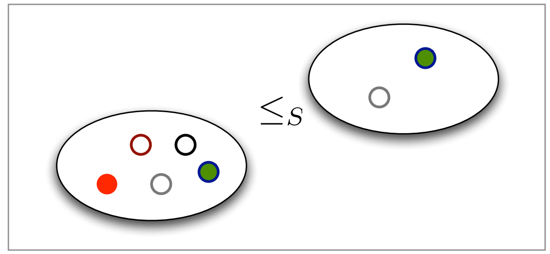


Powerdomains: the Smyth order

$$A\uparrow \quad \hat{=} \quad \{s \in S_\perp \mid (\exists a \in A \cdot a \leq s\}$$

$\leq_S$ becomes an order (rather than a pre-order) on up-closed sets.

On up-closed sets, refinement is simply reverse subset inclusion.



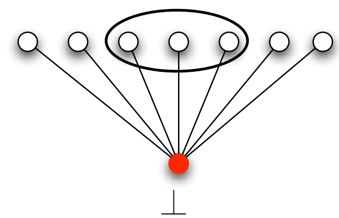
Probabilistic powerdomains

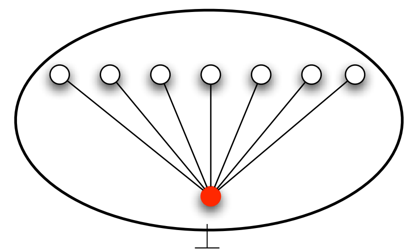
Given a structure (D, ≤), we can construct a powerdomain (Eval.D, ≤) where objects are evaluations over D, and the order is defined to make "appropriate distinctions".

- Evaluations are real-valued functions which are defined over the open sets of a (fixed) topology; under certain conditions they can be extended to probability distributions.

- Computational domains can be reformulated in terms of the Scott Topology: a set is Scott open if it is "up-closed" and "inaccessible" (any limit of a chain inside the set can only happen if the chain intersects the set).

Probabilistic powerdomains: Evaluations





$Eval.S_\bot \mathrel{\hat{=}} \mathcal{O}S_\bot \to [0,1]$

Monotone; additive

$$d \le d' \quad \mathit{iff} \quad (\forall O \in \mathcal{O}S_\bot \cdot d.O \le d'.O)$$

$$d \le d' \quad \mathit{iff} \quad (\forall s \in S \cdot d.\{s\} \le d'.\{s\})$$

Probability can increase up the refinement order!

Probabilistic powerdomains: Semantics

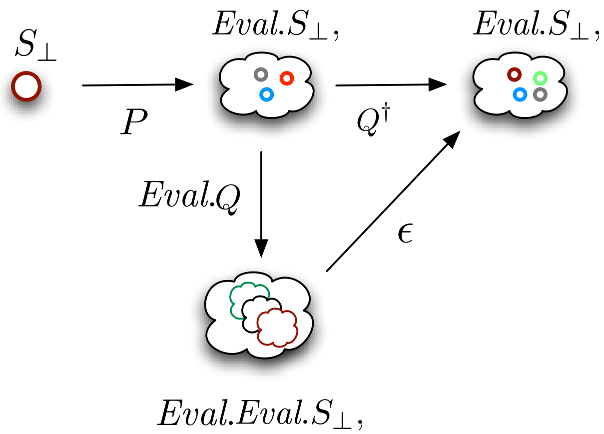
Given a structure (D, ≤), we can construct a powerdomain (Eval.D, ≤) where objects are evaluations over D, and the order is defined to make "appropriate distinctions".

Programs	$S_\perp \to Eval.S_\perp$
Probabilistic choice	$(P\ {}_p\!\oplus Q).s \quad \hat{=} \quad p{\times}P.s + (1{-}p){\times}Q.s$
Sequence	$P;Q \quad \hat{=} \quad P \circ Q^\dagger$

$$Q^\dagger : Eval.S_\perp \to Eval.S_\perp$$
$$Q^\dagger.d \quad \hat{=} \quad \textstyle\sum_{s:S}(d.s){\times}Q.s$$

Probabilistic powerdomains: Defining $Q^\dagger$

Sequence $\qquad P;Q \quad \hat{=} \quad P \circ Q^\dagger$



Now we have all the ingredients for instant probabilistic semantics.

“It’s marvelous! You just add water.”

First try:

You will need a flat domain, the Smyth Powerdomain, and the probabilistic powerdomain.

- Start with simple deterministic computations with nontermination; $(S_\perp \to S_\perp, \sqsubseteq)$

- Next apply the Smyth construction to introduce nondeterminism... $(S_\perp \to \mathbb{P}S_\perp, \sqsubseteq_S)$

- Finally fold in probability, stirring gently ... $(Eval.(S_\perp \to \mathbb{P}S_\perp), \sqsubseteq_S)$

Voila! But what is it?

- Probabilistic arithmetic $$(P \,{}_p\!\oplus Q) \,{}_q\!\oplus R \quad = \quad P \,{}_{pq}\!\oplus (Q \,{}_{\frac{(1-p)q}{1-pq}}\!\oplus R)$$

- Universal probabilistic distributivity .... $$(P \,{}_p\!\oplus Q) \sqcap R \quad = \quad (P \sqcap R) \,{}_p\!\oplus (Q \sqcap R)$$

.... which implies this ....

Probability versus nondeterminism

$$(y := 0 \sqcap y := 1); (x := 0 \,{}_{1/2}\!\oplus\, x := 1)$$

What’s the chance that the demon can guess the value of x?

Probability versus nondeterminism

$$(y := 0 \sqcap y := 1); (x := 0 \;{}_{1/2}\!\oplus x := 1) \quad \text{Prob distributes over nondet}$$

$$= \boxed{(y := 0 \sqcap y := 1); x := 0} \;{}_{1/2}\!\oplus \boxed{(y := 0 \sqcap y := 1); x := 1}$$

In this model, we can reproduce the demon's choice within each probabilistic branch....

.... effectively making the demon able to see into the future.

Whoops!

Next try:

You will need a flat domain, the Smyth Powerdomain, the probabilistic powerdomain, and compactness and convexity.

- First add probability $(Eval.S_\perp, \leq)$
- Next add nondeterminism $(S_\perp \rightarrow \mathbb{P}Eval.S_\perp, \sqsubseteq_P)$
- We need some extra closure conditions:
  (a) up-closed - for termination.
  (b) Convex closed - $P\ {}_p\!\oplus P = P$
  (c) Compact - so that iteration can be approximated by "finite" computations.

As before, refinement is reverse subset inclusion.

Relational-style semantics for a small sequential language

identity	$[\![\mathsf{skip}]\!].s$	$\hat{=}$	$\{\overline{s}\}$
assignment	$[\![x := a]\!].s$	$\hat{=}$	$\{\overline{s[x \mapsto a]}\}$
composition	$[\![P;P']\!].s$	$\hat{=}$	$\{\sum_{s' : S} d.s' \times f'.s' \mid d \in [\![P]\!].s; f' \sqsubseteq [\![P']\!]\}$
	where $f' \in S \to \overline{S}_\perp$ and in general $f' \sqsubseteq r'$ means $f'.s \in r'.s$ for all s.		
choice	$[\![\mathsf{if}\ B\ \mathsf{then}\ P\ \mathsf{else}\ P']\!].s$	$\hat{=}$	$\text{if } B.s \text{ then } [\![P]\!].s \text{ else } [\![P']\!].s$
probability	$[\![P\ {}_p\!\oplus P']\!].s$	$\hat{=}$	$\{d\ {}_p\!\oplus d' \mid d \in [\![P]\!].s; d' \in [\![P']\!].s\}$
nondeterminism	$[\![P \sqcap P']\!].s$	$\hat{=}$	$\lceil\, [\![P]\!].s \cup [\![P']\!].s\, \rceil$,
	where in general $\lceil D \rceil$ is the up-, convex- and Cauchy closure of D.		
iteration	$\mathsf{do}\ \ G \to P\ \ \mathsf{od}$	$\hat{=}$	$(\mu X \cdot \mathsf{if}\ \ G\ \ \mathsf{then}\ [\![P]\!]; X\ \ \mathsf{else}\ \ [\![\mathsf{skip}]\!])$.

Probabilistic models for the guarded command language.
He Ji Feng et al.
Special issue SCP containing selected papers
from the FMTA '95 conference (May 1995, Warsaw)

Some nice laws....

$$P \sqcap P = P$$

$$P \sqcap P \sqsubseteq_P P\ {}_p\!\oplus P = P$$

$$P; (Q\ {}_p\!\oplus R) \sqsubseteq_P P; Q\ {}_p\!\oplus P; R$$

$$(Q\ {}_p\!\oplus R); P = (Q; P\ {}_p\!\oplus R; P)$$

$$(P \sqcap Q)\ {}_p\!\oplus (P \sqcap R) \sqsubseteq_P P \sqcap (Q\ {}_p\!\oplus R)$$

$$P\ {}_p\!\oplus (Q \sqcap R) = (P\ {}_p\!\oplus Q) \sqcap (P\ {}_p\!\oplus R)$$

This nondeterminism (demon) can see what happened after a coin flip, but not before.

Probability versus nondeterminism

$$(y := 0 \sqcap y := 1); (x := 0 \;{}_{1/2}{\oplus}\; x := 1)$$

What's the chance that the demon can guess the value of x?

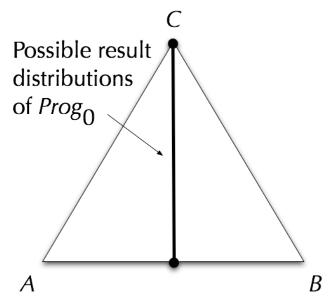
Probability versus nondeterminism

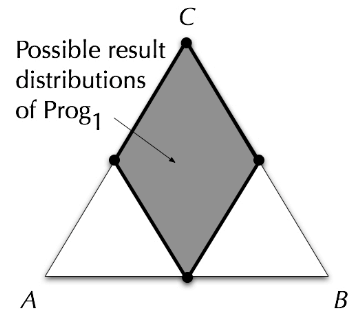
$$(y := 0 \sqcap y := 1); (x := 0 \ {}_{1/2}\oplus \ x := 1)$$ Nondet distributes over pro

$$= \boxed{(y := 0); (x := 0 \ {}_{1/2}\oplus \ x := 1)} \sqcap \boxed{(y := 1); (x := 0 \ {}_{1/2}\oplus \ x := 1)}$$

What's the chance that the demon can guess the value of x?
Answer is 1/2.

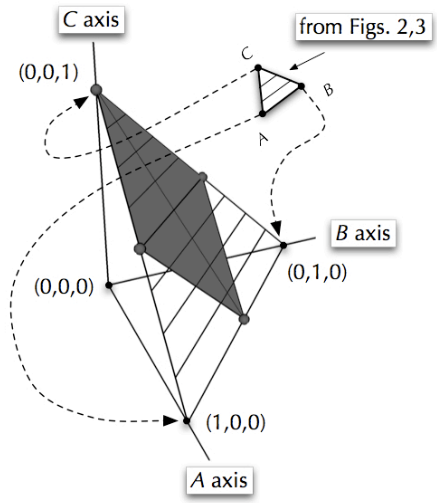
Geometrical interpretation.





$$Prog_0 \quad \hat{=} \quad (s := A \;{}_{0.5}\!\oplus\; s := B) \;\sqcap\; s := C$$
$$Prog_1 \quad \hat{=} \quad (s := A \sqcap s := C) \;{}_{0.5}\!\oplus\; (s := B \sqcap s := C)$$

Geometrical interpretation.



Plotted on the same diagram, we can see immediately the relationship between the two programs.

$$Prog_0 \quad \hat{=} \quad (s := A \ {}_{0.5}\oplus \ s := B) \ \sqcap \ s := C$$
$$Prog_1 \quad \hat{=} \quad (s := A \sqcap s := C) \ {}_{0.5}\oplus \ (s := B \sqcap s := C)$$

Logic and properties: Generalising Hoare Logic

Properties are now quantitative; use random variables.

$$\mathbb{E}S \;\hat{=}\; S \to [0,1]$$
$$e \leq e' \;=\; (\forall s : S \cdot e.s \leq e'.s)$$

$$\mathsf{wp}.P.e.s \;\hat{=}\; (\sqcap \; d \in P.s \cdot \int_d e)$$

Greatest guaranteed expected value of e with respect to the results of P from initial state s.

$$d \in EvalS_\perp,\; e \in \mathbb{E}S, \quad \int_d e \;\hat{=}\; \sum_{s:S} d.s \times e.s$$

Transformer semantics for a small sequential language

identity	$\mathsf{wp.skip}.expt$	$\mathrel{\hat{=}}\ expt$
assignment	$\mathsf{wp}.(x := E).expt$	$\mathrel{\hat{=}}\ expt[x := E]$
composition	$\mathsf{wp}.(P; P').expt$	$\mathrel{\hat{=}}\ \mathsf{wp}.P.(\mathsf{wp}.P'.expt)$
choice	$\mathsf{wp}.(\mathsf{if}\ B\ \mathsf{then}\ P\ \mathsf{else}\ P'\ \mathsf{fi}).expt$ $\mathrel{\hat{=}} [B] \times \mathsf{wp}.P.expt + [\neg B] \times \mathsf{wp}.P'.expt$	
probability	$\mathsf{wp}.(P\ {}_p\!\oplus P').expt$ $\mathrel{\hat{=}} p \times \mathsf{wp}.P.expt + (1-p) \times \mathsf{wp}.P'.expt$	
nondeterminism	$\mathsf{wp}.(P \sqcap P').expt$	$\mathrel{\hat{=}}\ \mathsf{wp}.P.expt\ \mathbf{min}\ \mathsf{wp}.P'.expt$
iteration	$\mathsf{wp}.(\mathsf{do}\ B \rightarrow r\ \mathsf{od}).e \mathrel{\hat{=}} (\mu X \bullet [B] \times \mathsf{wp}.r.X + [\neg B] \times e)$.	

Logic and properties: the monotonic transformers

$$\mathbb{T}S \quad \hat{=} \quad \mathbb{E}S \leftarrow \mathbb{E}S$$

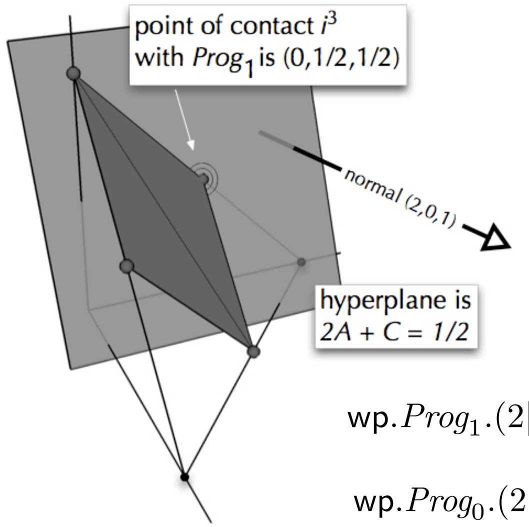
$$S_\perp \to \mathbb{P}Eval.S_\perp \quad \underset{rp}{\overset{\mathsf{wp}}{\rightleftarrows}} \quad \mathbb{T}S$$

$$\mathsf{wp} \circ rp \quad = \quad id$$

$$rp \circ \mathsf{wp} \quad = \quad id \text{ , if } \begin{cases} t.(e \,_p\!\oplus e') \geq t.e \,_p\!\oplus t.e' \\ t.(\mathbf{k}e) = \mathbf{k}t.e \\ t.(e - \mathbf{k}) \geq t.e - \mathbf{k} \end{cases}$$

"Sublinear"

Geometrical interpretation:



Random variables are "hyperplanes".

$$\mathsf{wp}.Prog_1.(2[s = A] + [s = C]) \quad = \quad 1/2$$

$$\mathsf{wp}.Prog_0.(2[s = A] + [s = C]) \quad = \quad 1$$

Why so complicated: can't we just have a whole logic based on probabilities, rather than random variables?

It's a question of compositionality:

$$
\begin{array}{lcl}
Prog_0 & \hat{=} & (s := A \;{}_{0.5}\!\oplus\; s := B) \;\sqcap\; s := C \\
Prog_1 & \hat{=} & (s := A \sqcap s := C) \;{}_{0.5}\!\oplus\; (s := B \sqcap s := C)
\end{array}
$$

Allowed final value(s) of s	A	B	C	A, B	B, C	C, A
Maximim possible probability	1/2	1/2	1	1	1	1
Minimim possible probability	0	0	0	0	1/2	1/2

A quantitative logic based on probabilities *is not compositional.*

Consider the following “context”:

$$Prog_0;\quad \textsf{if } s{=}C \textsf{ then } (s := A \ {}_{0.5}\!\oplus s := B) \textsf{ fi} \qquad 1/2$$
$$Prog_1;\quad \textsf{if } s{=}C \textsf{ then } (s := A \ {}_{0.5}\!\oplus s := B) \textsf{ fi} \qquad 1/4$$

What’s the probability that the state is A finally?

As we have seen, the two programs can be distinguished in the transformer semantics (by a random variable encoded as an expectation).

$$\mathsf{wp}.Prog_0.(2[s = A] + [s = C]) \quad = \quad 1$$

$$\mathsf{wp}.Prog_1.(2[s = A] + [s = C]) \quad = \quad 1/2$$

The transformer semantics, based on full random variables, *is* compositional.

A nice proof rule, proved using the transformer semantics:

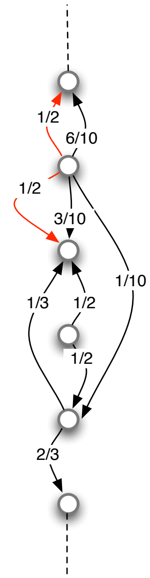
A loop: $\mathsf{do}\ \ G\ \rightarrow \mathit{body}\ \ \mathsf{od}$

An invariant: $[G]{\times}I \quad \leq \quad \mathsf{wp}.\mathit{body}.I$

Termination condition: $T \quad \hat{=} \quad \mathsf{wp}.(\mathsf{do}\ \ G\ \rightarrow \mathit{body}\ \ \mathsf{od}).1$

A rule: $I \leq T \quad \Rightarrow \quad I \leq \mathsf{wp}.(\mathsf{do}\ \ G\ \rightarrow \mathit{body}\ \ \mathsf{od}).I$

The “jumping bean” : specification.



$$[n = N] \quad \leq \quad \lceil \mathsf{wp}.jump.[n \neq N] \rceil$$

The bean must move...

$$[n = N] \quad \leq \quad \mathsf{wp}.jump.[N{-}K \leq n \leq N{+}K]$$

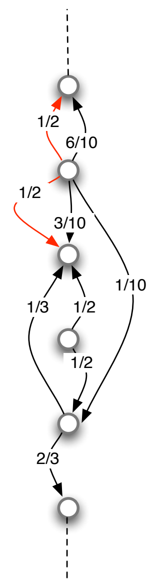
(K is a fixed constant.)

The bean can’t move too much...

$$n \quad \leq \quad \mathsf{wp}.jump.n$$

The expected move is at least 0.

The “jumping bean”.



$$Bean \quad \hat{=} \quad \mathsf{wp}.(\mathsf{do} \;\; (n \leq N) \rightarrow \; jump \;\; \mathsf{od})$$

The bean continues to jump, until it exceeds N.

$$1 \quad = \quad \mathsf{wp}.Bean.[n > N]$$

The conditions on its behaviour guarantee that it will eventually exceed any bound.

Exercise: use the properties of the transformers to prove this. (Should be about 10 lines of proof.)

Automated invariant generation.

Usually the user/prover must supply the loop invariants to enable programs to be verified.

For certain classes of invariants/programs we can automate the process:

• Linear invariants and linear programs;
• Wp- under these conditions preserves linearity;
• Reduce searching for invariants to the solution of linear equations.

```
x := p;  b := true;
while b do
  b := false 1/2⊕ true;
  if b then
      x := 2x;
      if (x ≥ 1) then x := x−1 else skip fi
  elseif (x ≥ 1/2) then x := 1
  else x := 0
  fi
od
```

x is a variable of type $\mathbb{R}$ and b of type $\mathbb{B}$. This program is supposed to set x to 1 with probability exactly p.

Fig. 4. Generating a biased coin from a fair one.

Probability versus nondeterminism:

$$
\begin{aligned}
& (x := 0 \sqcap x := 1); (y := 0 \;{}_{1/2}\!\oplus\; y := 1) \\
= \quad & (x := 0 \sqcap x := 1); (y := 0) \\
& {}_{1/2}\!\oplus \\
& (x := 0 \sqcap x := 1); y := 1)
\end{aligned}
$$

The demon can predict the future.

$$
\begin{aligned}
& (y := 0 \;{}_{1/2}\!\oplus\; y := 1); (x := 0 \sqcap x := 1) \\
= \quad & y := 0; (x := 0 \sqcap x := 1) \\
& {}_{1/2}\!\oplus \\
& y := 1; (x := 0 \sqcap x := 1)
\end{aligned}
$$

The demon can access the past.

Probability versus nondeterminism:

Smyth powerdomain, for nondeterminism; then the probabilistic powerdomain on top of that.	The demon can predict the future.
Probabilistic powerdomain to make $Eval S_{\perp}$, then the Smyth powerdomain to make, $S_{\perp} \rightarrow \mathbb{P} Eval.S_{\perp}$ with a special definition of ";"	The demon can access the past.

Suppose we wanted to prevent the demon from accessing the past, i.e.

$$
\begin{array}{ll}
 & (y := 0 \;{}_{1/2}\!\oplus\; y := 1); (x := 0 \;\sqcap\; x := 1) \\
= & (y := 0 \;{}_{1/2}\!\oplus\; y := 1); x := 0 \\
 & \sqcap \\
 & (y := 0 \;{}_{1/2}\!\oplus\; y := 1); x := 1
\end{array}
$$

How would we build a semantic domain justifying this algebraic property?

Suppose we wanted to prevent the demon from accessing the past, i.e.

Use the probabilistic powerdomain to build $Eval.S_{\perp} \rightarrow Eval.S_{\perp}$,
and then the Smyth powerdomain to build $Eval.S_{\perp} \rightarrow \mathbb{P}Eval.S_{\perp}$

Key thing is to define the sequence operator so that it doesn't "split up" the probabilistic results.

The “refinement paradox”

Properties of the logic/algebra in the context where “hidden state” is an issue are hard to get right, even when there are no probabilities.

It turns out to be a really hard problem to find a formalisation which behaves properly for refinement

h “High security” variables (are “private”)

l “Low security” variables (are “public”)

“Obviously” we want to make sure that going up the refinement order preserves our security properties.