Formal Methods for Probabilistic Systems

Annabelle McIver
Carroll Morgan

- Source-level program logic
  - Introduction to probabilistic-program logic
  - Systematic presentation via structural induction
  - Layout of calculations in practice
  - Random variables and expected values
  - The impact of demonic choice

Combined logic of weakest pre-expectations: *preE* ⇛ wp(*prog*,*postE*)

add *non-determinism and probability*

Formal Methods for Probabilistic Systems

Annabelle McIver
Carroll Morgan

- Source-level program logic
  - Introduction to probabilistic-program logic
  - Systematic presentation via structural induction
  - Layout of calculations in practice
  - Random variables and expected values
  - The impact of demonic choice

When are two probabilistic programs equal?

Equality — first guess

Two (probabilistic) programs are equal iff from all initial states they have equal probability of establishing equal postconditions: for example,

$$coin{:=}\,heads \;\boxed{{}_{2/3}\oplus}\; coin{:=}\,tails$$

and

$$coin{:=}\,tails \;\boxed{{}_{1/3}\oplus}\; coin{:=}\,heads$$

are equal.

Probabilistic choice.

{}	0
{H}	2/3
{T}	1/3
{H,T}	1

What about

Nondeterministic choice.

$$coin{:=}\,edge \boxed{\sqcap} (coin{:=}\,heads \;{}_{1/2}\oplus\; coin{:=}\,tails)$$

and

$$(coin{:=}\,edge \boxed{\sqcap} coin{:=}\,heads) \;{}_{1/2}\oplus\; (coin{:=}\,edge \boxed{\sqcap} coin{:=}\,tails)\,.$$

Are they equal?

Equality — first guess

$$coin{:=}\,edge \sqcap (coin{:=}\,heads \;{}_{1/2}\oplus\; coin{:=}\,tails)$$

postcondition	probability
{}	0
{*edge*}	0 - 1
{*heads*}	0 - 1/2
{*tails*}	0 - 1/2
{*edge, heads*}	1/2 - 1
{*edge, tails*}	1/2 - 1
{*heads, tails*}	0 - 1
{*edge, heads, tails*}	1

Equality — first guess

$$(coin := edge \sqcap coin := heads)$$
$$_{1/2}\oplus \; (coin := edge \sqcap coin := tails)$$

postcondition	probability
{}	0
{*edge*}	0 - 1
{*heads*}	0 - 1/2
{*tails*}	0 - 1/2
{*edge, heads*}	1/2 - 1
{*edge, tails*}	1/2 - 1
{*heads, tails*}	0 - 1
{*edge, heads, tails*}	1

Equality — first guess

Two (probabilistic) programs are equal iff from all initial states they have equal probability of establishing equal postconditions: for example,

{}	0
{H}	2/3
{T}	1/3
{H,T}	1

$$coin := heads \;\; _{2/3}\oplus \;\; coin := tails$$

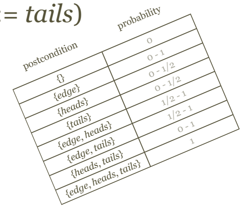
and

$$coin := tails \;\; _{1/3}\oplus \;\; coin := heads$$

are equal.

What about

$$coin := edge \sqcap (coin := heads \; _{1/2}\oplus \; coin := tails)$$



and

$$(coin := edge \sqcap coin := heads)$$
$$_{1/2}\oplus \; (coin := edge \sqcap coin := tails) \; .$$

Are they equal? Apparently they are.

Equality — first guess, wrong guess

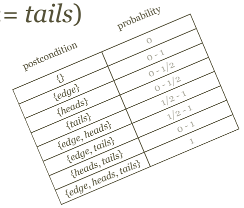
But *should* they be?

No, they should not — and their indistinguishability in this simple probabilistic logic makes it *non-compositional*.

The logic we now present represents the "least extra effort" —in a sense that can be made precise— that regains compositionality.

What about

$$coin := edge \sqcap (coin := heads \; _{1/2}\oplus \; coin := tails)$$



and

$$(coin := edge \sqcap coin := heads)$$
$$_{1/2}\oplus \; (coin := edge \sqcap coin := tails) \; .$$

Are they equal? Apparently they are.

Formal Methods for Probabilistic Systems

Annabelle McIver
Carroll Morgan

Source-level program logic

- Introduction to probabilistic-program logic (compositional)
- Systematic presentation via structural induction
- Layout of calculations in practice
- Random variables and expected values
- The impact of demonic choice

Probabilistic-program logic: introduction

What is the probability that the probabilistic program

$$coin{:=}\ heads\ \boxed{{}_{1/2}\oplus}\ \ coin{:=}\ tails$$

establishes the postcondition $coin = heads$?

Probabilistic choice: 1/2 left; (1-1/2) right.

We can abbreviate "$coin{:=}\ heads\ {}_{1/2}\oplus\ \ coin{:=}\ tails$" as just

$$coin{:=}\ heads\ {}_{1/2}\oplus\ tails$$

because the left-hand sides "$coin{:=}$" are the same.

Probabilistic-program logic

What is the probability that the program

$$coin{:=}\ heads\ {}_{1/2}\oplus\ tails$$

establishes the postcondition $coin = heads$?

In the program logic we write

$$wp.(\boxed{coin{:=}\ heads\ {}_{1/2}\oplus\ tails}).[\boxed{coin = heads}]\ \equiv\ \boxed{1/2}$$

to say that the probability is just 1/2.

CC Morgan, AK McIver and K Seidel. *Probabilistic predicate transformers.* TOPLAS 18(3):325-353, 1996.

D Kozen. *A probabilistic PDL.* J. Comp. & Sys. Sci. 30(2):162-178, 1985.

Probabilistic programs

1. Assignment statements;
2. Probabilistic choice;
3. Conditionals;
4. Sequential composition;
5. Demonic choice.

— written in *pGCL*.

We will look at these in turn: what we need to know for each type of program fragment *prog* is

What is $wp.prog.B$ for arbitrary postcondition B ?

The usual technique for setting this out is *structural induction* over the syntax of the programming language.

CC Morgan, AK McIver. *pGCL: Formal reasoning for random algorithms.* SACJ 22, 1999.

Probabilistic programs: assignment statements

$x{:=}\ E$ — Assign the value of expression E to the variable x. — Informal description.

$$wp.(x{:=}\ E).B\ \equiv\ B\boxed{\langle}x{:=}\ E\boxed{\rangle}$$

Syntactic substitution.

Definition.

$$\begin{array}{lll}
 & wp.(x{:=}\ x{+}1).[x{=}3] & \\
\equiv & [x{=}3]\langle x{:=}\ x{+}1\rangle & \textit{definition} \\
\equiv & [(x{+}1){=}3] & \textit{substitution} \\
\equiv & \boxed{[}x{=}2\boxed{]} & \textit{arithmetic}
\end{array}$$

Why are these here?

Example.

Probabilistic programs: embedded Booleans

$$
\begin{array}{lll}
 & wp.(x{:=}\,x{+}1).[x{=}3] & \\
\equiv & [x{=}3]\langle x{:=}\,x{+}1\rangle & \textit{definition} \\
\equiv & [(x{+}1){=}3] & \textit{substitution} \\
\equiv & \boxed{[}x{=}2\boxed{]} & \textit{arithmetic}
\end{array}
$$

The *probability* that $x{:=}\,x{+}1$ achieves $x{=}3$ is *one* if $x{=}2$ initially, and *zero* otherwise.

Thus "$\boxed{[\bullet]}$" must be an *embedding function* that takes *true* to one and *false* to zero.

Probabilistic programs: probabilistic choice

$prog_1 \;{}_p\!\oplus\; prog_2$ — Execute the left-hand side with probability p, otherwise execute the right-hand side (probability $1{-}p$).

$$
wp.(prog_1 \;{}_p\!\oplus\; prog_2).B \;\equiv\; \begin{array}{rl} p & \times\; wp.prog_1.B \\ +\;(1{-}p) & \times\; wp.prog_2.B \end{array}
$$

$$
\begin{array}{lll}
 & wp.(\,c{:=}\,H \;{}_{1/2}\!\oplus\; T\,).[c{=}H] & \\
\equiv & \quad 1/2 \;\times wp.(c{:=}\,H).[c{=}H] & \textit{definition} \\
 & +\,(1{-}1/2) \;\times wp.(c{:=}\,T\,).[c{=}H] & \\
\equiv & 1/2 \times [H{=}H] \;+\; 1/2 \times [T{=}H] & \textit{assignment} \\
\equiv & 1/2 \times 1 \;+\; 1/2 \times 0 & \textit{embedding} \\
\equiv & 1/2\,. & \textit{arithmetic}
\end{array}
$$

Probabilistic programs: syntactic "sugar"

if G **then** *prog* **fi** — If guard G holds, then execute the body *prog*; otherwise do nothing.

if G **then** *prog* **else skip fi**

skip — Do nothing. — $x{:=}\,x$

if G **then** $prog_1$ **else** $prog_2$ **fi** — If guard G holds, then execute $prog_1$; otherwise execute $prog_2$.

If G holds, then go left with probability 1, and vice versa. — $prog_1 \;{}_{[G]}\!\oplus\; prog_2$

Probabilistic programs: conditional

$$
\begin{array}{lll}
 & wp.(\textbf{if}\; x \geq 1\; \textbf{then}\; x{:=}\,x-1\; \textbf{else}\; x{:=}x+2\; \textbf{fi}).[x \geq 2] & \\
\equiv & wp.(x{:=}\,x-1 \;{}_{[x \geq 1]}\!\oplus\; x{:=}x+2).[x \geq 2] & \textit{“sugar”} \\
\equiv & \quad [x \geq 1] \;\times\; wp.(x{:=}\,x-1).[x \geq 2] & \textit{prob. choice} \\
 & +\,\boxed{(1{-}[x \geq 1])} \;\times\; wp.(x{:=}x+2).[x \geq 2] & \\
\equiv & [x \geq 1] \times [(x{-}1) \geq 2] \;+\; \boxed{[x < 1]} \times [(x{+}2) \geq 2] & \textit{assignment} \\
\equiv & [x \geq 1] \times [x \geq 3] \;+\; [x < 1] \times [x \geq 0] & \textit{arithmetic} \\
\equiv & [x \geq 1 \wedge x \geq 3 \;\vee\; x < 1 \wedge x \geq 0] & \textit{embedding} \\
\equiv & [x \geq 3 \;\vee\; 0 \leq x < 1]\,. & \textit{logic}
\end{array}
$$

For a *standard* conditional, the reasoning is just "as usual".

Probabilistic programs: sequential composition

$prog_1 ;\ prog_2$ — Execute the first program; then execute the second.

$$wp.(prog_1 ;\ prog_2).B \equiv wp.prog_1.(wp.prog_2.B)$$

$wp.(c{:=}\ H\ {}_{1/2}\oplus\ T;\ d{:=}\ H\ {}_{1/2}\oplus\ T).[c{=}d]$

$\equiv\ wp.(c{:=}\ H\ {}_{1/2}\oplus\ T).(\boxed{wp.(d{:=}\ H\ {}_{1/2}\oplus\ T).[c{=}d]})$ — *definition*

$\equiv\ wp.(c{:=}\ H\ {}_{1/2}\oplus\ T).($ — *prob. choice; assignment*

$\quad \boxed{1/2 \times [c{=}H]\ +\ 1/2 \times [c{=}T]}$

$)$

$\equiv\ \ 1/2 \times (1/2 \times [H{=}H] + 1/2 \times [H{=}T])$ — *prob. choice; assignment*

$+\ \ 1/2 \times (1/2 \times [T{=}H] + 1/2 \times [T{=}T])$

$\equiv\ 1/4 + 1/4$ — *embedding*

$\equiv\ 1/2\ .$ — *arithmetic*

Probabilistic programs: layout of calculations

$wp.(prog_1 ;\ prog_2;\ prog_3\ \ldots).B$ — The component programs might have to be written and rewritten many times...

$wp.(c{:=}\ H\ {}_{1/2}\oplus\ T;\ d{:=}\ H\ {}_{1/2}\oplus\ T).[c{=}d]$

$c{:=}\ H\ {}_{1/2}\oplus\ T$

$c{:=}\ H\ {}_{1/2}\oplus\ T$

$c{:=}\ H\ {}_{1/2}\oplus\ T$

Probabilistic programs: layout of calculations

$wp.(prog_1 ;\ prog_2;\ prog_3\ \ldots).B$ — The component programs might have to be written and rewritten many times...
unless we do this:

$wp.(c{:=}\ H\ {}_{1/2}\oplus\ T;\ d{:=}\ H\ {}_{1/2}\oplus\ T).[c{=}d]$

Probabilistic programs: layout of calculations

$wp.(c{:=}\ H\ {}_{1/2}\oplus\ T;\ \underline{d{:=}\ H\ {}_{1/2}\oplus\ T}).[c{=}d]$

Work from back to front, writing each program fragment only when it is used in the calculation.

$[c{=}d]$

• $\boxed{\equiv}\ \ \boxed{1/2 \times [c{=}H]\ +\ 1/2 \times [c{=}T]}$ — $wp.(d{:=}\ H\ {}_{1/2}\oplus\ T)$

• $\equiv\ \ 1/2 \times (1/2 \times [H{=}H] + 1/2 \times [H{=}T])$ — $\boxed{wp.(c{:=}\ H\ {}_{1/2}\oplus\ T)}$

$+\ \ 1/2 \times (1/2 \times [T{=}H] + 1/2 \times [T{=}T])$

$\equiv\ 1/2\ .$ — *arithmetic*

AJM van Gasteren and A Bijlsma. *An extension of the program derivation format*. Proc. PROCOMET 98, Programming Concepts and Methods, D Gries and W-P de Roever (eds). IFIP Conf. Procs. 125:167-185, 1998.

Probabilistic programs: "proper" *probabilistic* postconditions

$$wp.(c{:=}\ H\ {}_{1/2}\!\oplus\ T\,).(1/2 \times [c{=}H] + 1/2 \times [c{=}T])$$

$$\begin{aligned}
\equiv &\quad 1/2 \times (1/2 \times [H{=}H] + 1/2 \times [H{=}T]) \\
+ &\quad 1/2 \times (1/2 \times [T{=}H] + 1/2 \times [T{=}T])
\end{aligned}$$

$$\equiv \quad 1/2\,.$$

The *expected value* of the function $1/2 \times [c{=}H] + 1/2 \times [c{=}T]$ over the distribution of states produced by the program is $1/2$.

As a special case (from elementary probability theory) we know that the expected value of the function $[pred]$, for some Boolean *pred*, is just the probability that *pred* holds.

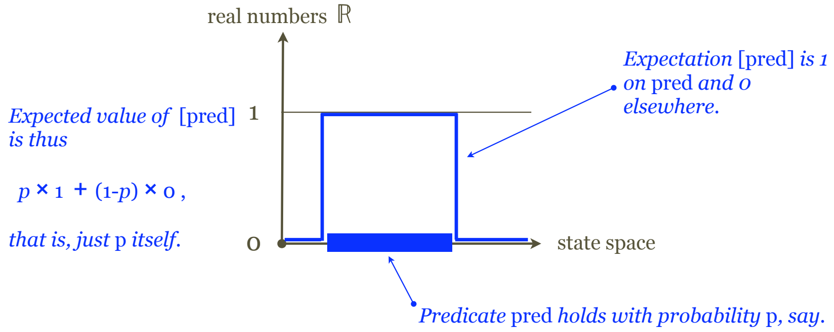
That's why $wp.prog.[pred]$ gives the probability that *pred* is achieved by *prog*. But, as we see above, we can be much more general if we wish.

Probabilistic programs: proper *post-expectations*

The expression $wp.prog.B$ gives, as a function of the initial state, the *expected value* of the "post-expectation" *B* over the distribution of final states that *prog* will produce from there.

We call it the *greatest pre-expectation* of *prog* with respect to *B*. When *prog* and *B* are standard (*i.e.* non-probabilistic), it is the same as the *weakest precondition*... except that it is 0/1-valued rather than Boolean.

As a "hybrid", we have that $wp.prog.[pred]$ is the probability that *pred* will be achieved.



$$\boxed{\begin{array}{rcl}
wp.\mathbf{abort}.postE & := & 0 \\
wp.\mathbf{skip}.postE & := & postE \\
wp.(x{:=}\ expr).postE & := & postE\ \langle x \mapsto expr\rangle \\
wp.(prog;\,prog').postE & := & wp.prog.(wp.prog'.postE) \\
wp.(prog \sqcap prog').postE & := & wp.prog.postE\ \ \mathsf{min}\ \ wp.prog'.postE \\
wp.(prog\ {}_p\!\oplus\ prog').postE & := & p * wp.prog.postE\ \ +\ \ \overline{p} * wp.prog'.postE
\end{array}}$$

Recall that $\overline{p}$ is the complement of p.

The expression on the right gives the *greatest pre-expectation* of *postE* with respect to each *pGCL* construct, where *postE* is an expression of type $\mathbb{E}S$ over the variables in state space S. (For historical reasons we continue to write *wp* instead of *gp*.)

In the case of recursion, however, we cannot give a purely syntactic definition. Instead we say that

$$\begin{array}{rcl}
(\mathbf{mu}\ xxx \cdot \mathcal{C}) & := & \text{least fixed-point of the function } cntx{:}\ \mathbb{T}S \to \mathbb{T}S \\
& & \text{defined so that } cntx.(wp.xxx) = wp.\mathcal{C}.
\end{array}$$ [40]

Figure 1.5.3. Probabilistic *wp*-semantics of *pGCL*

Probabilistic programs: demonic choice

$prog_1 \sqcap prog_2$ — Execute the left-hand side — or maybe execute the right-hand side. *Whatever...*

$$wp.(prog_1 \sqcap prog_2).B \quad\equiv\quad wp.prog_1.B\ \ \mathbf{min}\ \ wp.prog_2.B$$

$$\begin{array}{lll}
& wp.(\,c{:=}\ H \ \sqcap\ c{:=}\ T\,).[c{=}H] & \\
\equiv & wp.(c{:=}\ H).[c{=}H]\ \ \mathbf{min}\ \ wp.(c{:=}\ T\,).[c{=}H] & \textit{definition} \\
\equiv & [H{=}H]\ \ \mathbf{min}\ \ [T{=}H] & \textit{assignment} \\
\equiv & 1\ \ \mathbf{min}\ \ 0 & \textit{embedding} \\
\equiv & 0\,. & \textit{arithmetic}
\end{array}$$

Although the program *might* achieve *c=H*, the largest probability of that which can be *guaranteed*... is zero.

Exercises

Ex. 1: Probabilistic then demonic choice

Calculate *wp*.($c{:=}\ H\ {}_{1/2}{\oplus}\ T\ ;\ \ d{:=}\ H \sqcap T$).[c=*d*] .

Ex. 2: Demonic then probabilistic choice

Calculate *wp*.($d{:=}\ H \sqcap T\ ;\ \ c{:=}\ H\ {}_{1/2}{\oplus}\ T$).[c=*d*] .

Ex. 3: Explain the difference

The answers you get to Ex. 1 and Ex. 2 should differ. Explain "in layman's terms" why they do.

(Hint: Imagine an experiment with two people and two coins, in each case.)

Ex. 4: The nature of demonic choice

It is sometimes suggested that *demonic* choice can be regarded as an arbitrary but unpredictable *probabilistic* choice; this would simplify matters because there would then only be one kind of choice to deal with.

Use our logic to investigate this suggestion; in particular, look at the behaviour of

$c{:=}\ H\ {}_{1/2}{\oplus}\ T\ ;\ \ d{:=}\ H\ {}_{p}{\oplus}\ T$ for arbitrary p,

and compare it with the program of Ex. 1. Explain your conclusions in layman's terms.

Ex. 5: Compositionality

Recall programs

A: *coin*:= *edge* ⊓ (*coin*:= *heads* $_{1/2}{\oplus}$ *coin*:= *tails*)

B: (*coin*:= *edge* ⊓ *coin*:= *heads*)
$_{1/2}{\oplus}$ (*coin*:= *edge* ⊓ *coin*:= *tails*) ,

which we now call *A* and *B*. Say that they are *similar* because from any initial state they have the same worst-case probability of achieving any given postcondition. (We showed this by tabulation.)

Find a program *C* such that *A;C* and *B;C* are *not similar* (even though *A* and *B* are). (Use the *wp*-definition of ";" .) What does this tell you about the simple program logic we considered briefly at the beginning?

More generally, let *A* and *B* be *any* two programs that are similar, but not equal in our *wp* logic. Show that there is *always* a program *C* as above, *i.e.* such that *A*;*C* and *B*;*C* are not similar. What does that tell you about our quantitative logic when compared to the simple logic?