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:
\( preE \Rightarrow wp(prog,postE) \)

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,

\[
\begin{align*}
\text{coin} := \text{heads} & \quad \frac{2}{3} \oplus \\
\text{and} \quad \text{coin} := \text{tails} & \quad \frac{1}{3} \oplus
\end{align*}
\]

are equal.

What about

\[
\begin{align*}
\text{coin} := \text{edge} & \quad \text{coin} := \text{heads} \quad \frac{1}{2} \oplus \\
\text{and} \quad \text{coin} := \text{edge} & \quad \text{coin} := \text{tails} \quad \frac{1}{2} \oplus
\end{align*}
\]

Are they equal?
**Equality — first guess**

\[ \text{coin} := \text{edge} \sqcap (\text{coin} := \text{heads} \oplus \text{coin} := \text{tails}) \]

<table>
<thead>
<tr>
<th>postcondition</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>0</td>
</tr>
<tr>
<td>{edge}</td>
<td>0 - 1</td>
</tr>
<tr>
<td>{heads}</td>
<td>0 - 1/2</td>
</tr>
<tr>
<td>{tails}</td>
<td>0 - 1/2</td>
</tr>
<tr>
<td>{edge, heads}</td>
<td>1/2 - 1</td>
</tr>
<tr>
<td>{edge, tails}</td>
<td>1/2 - 1</td>
</tr>
<tr>
<td>{heads, tails}</td>
<td>0 - 1</td>
</tr>
<tr>
<td>{edge, heads, tails}</td>
<td>1</td>
</tr>
</tbody>
</table>
Equality — first guess

\[(\text{coin:= edge} \land \text{coin:= heads})\]
\[1/2 \oplus \]
\[(\text{coin:= edge} \land \text{coin:= tails})\]
Equality — first guess

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

\[
\text{coin} := \text{heads} \ 2/3 \oplus \ \text{coin} := \text{tails} \\
\text{and} \quad \text{coin} := \text{tails} \ 1/3 \oplus \ \text{coin} := \text{heads}
\]

are equal.

What about

\[
\text{coin} := \text{edge} \cap (\text{coin} := \text{heads} \ 1/2 \oplus \ \text{coin} := \text{tails})
\]

and

\[
1/2 \oplus (\text{coin} := \text{edge} \cap \text{coin} := \text{heads}) \quad (\text{coin} := \text{edge} \cap \text{coin} := \text{tails})
\]

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

\[ \text{coin} := \text{edge} \land (\text{coin} := \text{heads}) \lor (\text{coin} := \text{tails}) \]

and

\[ 1/2 \oplus (\text{coin} := \text{edge} \land \text{coin} := \text{heads}) \]

\[ 1/2 \oplus (\text{coin} := \text{edge} \land \text{coin} := \text{tails}) . \]

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

\[
\textit{coin} := \textit{heads} \quad \frac{1}{2} \oplus \quad \textit{coin} := \textit{tails}
\]

establishes the postcondition \(\textit{coin} = \textit{heads} \) ?

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

We can abbreviate “\(\textit{coin} := \textit{heads} \quad \frac{1}{2} \oplus \quad \textit{coin} := \textit{tails}\)” as just

\[
\textit{coin} := \textit{heads} \quad \frac{1}{2} \oplus \quad \textit{tails}
\]

because the left-hand sides “\(\textit{coin} := \)” are the same.
In the program logic we write

\[ \text{\( wp. (\text{coin} := \text{heads} \frac{1}{2} \oplus \text{tails} ). [\text{coin} = \text{heads}] \equiv \frac{1}{2} \)} \]

to say that the probability is just \( \frac{1}{2} \).

What is the probability that the program

\[ \text{\( \text{coin} := \text{heads} \frac{1}{2} \oplus \text{tails} \)} \]

establishes the postcondition \( \text{coin} = \text{heads} \)?

Probabilistic-program logic


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 \textit{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 \( x \).

**Definition.**

\[ wp.(x := E).B \equiv B\langle x := E \rangle \]

Syntactic substitution.

\[ wp.(x := x+1).[x=3] \]
\[ \equiv [x=3]\langle x := x+1 \rangle \]
\[ \equiv [(x+1)=3] \]
\[ \equiv [x=2] \]

Example.

Why are these here?
Probabilistic programs: embedded Booleans

\[
wp.(x:= x+1).[x=3] \\
≡ [x=3]\langle x:= x+1\rangle \\
≡ [(x+1)=3] \\
≡ [x=2]
\]

The probability that \( x:= x+1 \) achieves \( x=3 \) is one if \( x=2 \) initially, and zero otherwise.

Thus “\([\bullet]\)” must be an embedding function that takes true to one and false to zero.
**Probabilistic programs:** probabilistic choice

\[ \text{prog}_1 \ p \oplus \text{prog}_2 \]

Execute the left-hand side with probability \( p \), otherwise execute the right-hand side (probability \( 1-p \)).

\[
\text{wp.}(\text{prog}_1 \ p \oplus \text{prog}_2).B \equiv \ p \times \text{wp.}\text{prog}_1.B \\
+ (1-p) \times \text{wp.}\text{prog}_2.B
\]

\[
\text{wp.}(c:= H_{1/2} \oplus T).[c=H] \\
\equiv 1/2 \times \text{wp.}(c:= H).[c=H] \\
+ (1-1/2) \times \text{wp.}(c:= T).[c=H]
\]

\[
\equiv 1/2 \times [H=H] + 1/2 \times [T=H] \\
\equiv 1/2 \times 1 + 1/2 \times 0 \\
\equiv 1/2 .
\]
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**

If guard $G$ holds, then execute $prog$; otherwise do nothing.

**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

\[ wp.(\text{if } x \geq 1 \text{ then } x := x - 1 \text{ else } x := x + 2 \text{ fi}).[x \geq 2] \]

\[ \equiv \quad wp.(x := x - 1 \ [x \geq 1] \oplus x := x + 2).[x \geq 2] \]

\[ \equiv \quad [x \geq 1] \times wp.(x := x - 1).[x \geq 2] \]
\[ \quad + \quad (1-[x \geq 1]) \times wp.(x := x + 2).[x \geq 2] \]

\[ \equiv \quad [x \geq 1] \times [(x-1) \geq 2] \quad + \quad [x < 1] \times [(x+2) \geq 2] \]

\[ \equiv \quad [x \geq 1] \bigtriangleup [x \geq 3] \quad + \quad [x < 1] \bigtriangleup [x \geq 0] \]

\[ \equiv \quad [x \geq 1] \bigtriangleup x \geq 3 \quad \lor \quad x < 1 \bigtriangleup x \geq 0 \]

\[ \equiv \quad [x \geq 3 \lor 0 \leq x < 1] . \]

For a standard conditional, the reasoning is just “as usual”.
**Probabilistic programs:** sequential composition

\[ \text{prog}_1 ; \text{prog}_2 \quad \begin{align*} & \text{Execute the first program;} \\ & \text{then execute the second.} \end{align*} \]

\[ \text{wp.}(\text{prog}_1 ; \text{prog}_2 ).B \equiv \text{wp.} \text{prog}_1 .(\text{wp.} \text{prog}_2 .B) \]

\[
\begin{align*}
\text{wp.}(c := H_{1/2} \oplus T ; \ d := H_{1/2} \oplus T ).[c=d] \\
\equiv \text{wp.}(c := H_{1/2} \oplus T ).(\text{wp.}(d := H_{1/2} \oplus T ).[c=d]) \\
\equiv \text{wp.}(c := H_{1/2} \oplus T ).(
\quad 1/2 \times [c=H] + 1/2 \times [c=T] \\
\quad ) \\
\equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) \\
+ 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \\
\equiv 1/4 + 1/4 \\
\equiv 1/2.
\end{align*}
\]

*definition*  
*prob. choice; assignment*  
*prob. choice; assignment*  
*embedding*  
*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...

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

\[
\begin{align*}
  c &:= H_{1/2} \oplus T \\
  c &:= H_{1/2} \oplus T \\
  c &:= H_{1/2} \oplus T \\
  c &:= H_{1/2} \oplus T
\end{align*}
\]
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:

\[ wp.\left(c:= H_{1/2} \oplus T ; \ d:= H_{1/2} \oplus T \right).[c=d] \]
**Probabilistic programs:** layout of calculations

\[
wp.(c:= H_{1/2} \oplus T; \quad d:= H_{1/2} \oplus T).[c=d]
\]

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

\[
[c=d] \\
\equiv \frac{1}{2} \times [c=H] + \frac{1}{2} \times [c=T]
\]

\[
\equiv \frac{1}{2} \times (\frac{1}{2} \times [H=H] + \frac{1}{2} \times [H=T]) \\
+ \frac{1}{2} \times (\frac{1}{2} \times [T=H] + \frac{1}{2} \times [T=T])
\]

\[
\equiv \frac{1}{2}.
\]

Probabilistic programs: “proper” probabilistic postconditions

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

\[ \equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) + 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \]

\[ \equiv 1/2 . \]

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.

\[
p \times 1 + (1-p) \times 0 ,
\]

that is, just \( p \) itself.

*Predicate \( pred \) holds with probability \( p \), say.*

*Expected value of \( [pred] \) is thus*
wp.abort.postE := 0
wp.skip.postE := postE
wp.(x := expr).postE := postE (x ↦ expr)
wp.(prog; prog').postE := wp.prog.(wp.prog'.postE)
wp.(prog ∨ prog').postE := wp.prog.postE \min wp.prog'.postE
wp.(prog ⊕ prog').postE := p * wp.prog.postE + \overline{p} * wp.prog'.postE

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

\[
\text{(mu } xxx \cdot C) := \text{ least fixed-point of the function } cntx : \mathbb{T}S \rightarrow \mathbb{T}S
\]
defined so that \(cntx.(wp.xxx) = wp.C\). \(40\)

Figure 1.5.3. Probabilistic \(wp\)-semantics of \(pGCL\)
**Probabilistic programs:** demonic choice

\[ \text{prog}_1 \sqcap \text{prog}_2 \quad \text{Execute the left-hand side — or maybe execute the right-hand side. Whatever...} \]

\[ \text{wp.}(\text{prog}_1 \sqcap \text{prog}_2).B \equiv \text{wp.} \text{prog}_1.B \min \text{wp.} \text{prog}_2.B \]

\[ \text{wp.}(c := H \sqcap c := T).[c=H] \]
\[ \equiv \text{wp.}(c := H).[c=H] \min \text{wp.}(c := T).[c=H] \quad \text{definition} \]
\[ \equiv [H=H] \min [T=H] \quad \text{assignment} \]
\[ \equiv 1 \min 0 \quad \text{embedding} \]
\[ \equiv 0 . \quad \text{Although the program might achieve } c=H, \text{ the largest probability of that which can be guaranteed... is zero.} \]
Exercises

Ex. 1: Probabilistic then demonic choice

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

Ex. 2: Demonic then probabilistic choice

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

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

\[ c := H_{1/2} \oplus T ; \quad d := H_p \oplus T \]

for arbitrary \( p \),

and compare it with the program of Ex. 1. Explain your conclusions in layman’s terms.
Ex. 5: Compositionality

Recall programs

\[ A: \quad \text{coin} := \text{edge} \land (\text{coin} := \text{heads} \oplus \text{coin} := \text{tails}) \]

\[ B: \quad (\text{coin} := \text{edge} \land \text{coin} := \text{heads}) \oplus (\text{coin} := \text{edge} \land \text{coin} := \text{tails}) \]

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?