Formal Methods for Probabilistic Systems

Annabelle McIver
Carroll Morgan

- Probabilistic temporal logic: \( qTL \)
- Probabilistic sequential-programming logic: \( pGCL \)
  - Origins of (this) program logic
  - Syntax and semantics of \( pGCL \)
  - Geometric interpretation (informal)
  - Metatheorems (for iteration)
Program logic

- What *kind* of Logic is it?
- *Where* did it come from?
- How does it *fit in*?
Program logic

Hoare logic of sequential programs: \{pre\} prog \{post\}

Floyd static annotations of flowchart programs

Inspired...
Program logic

relational model: $S \rightarrow S$

Hoare logic of sequential programs: $\{pre\} \text{prog} \{post\}$

Floyd static annotations of flowchart programs

Inspired... Is modelled by...

C.A.R. Hoare.
An axiomatic basis for computer programming.
Comm. A.C.M. 12(10), 1969
Dijkstra logic of weakest preconditions: 
\[ pre = \text{wp}(prog, post) \]

Is modelled by...

Floyd static annotations of flowchart programs

Generalises to...

Add non-determinism

Dijkstra logic of weakest preconditions: 
\[ pre \Rightarrow \text{wp}(prog, post) \]
Program logic

Dijkstra logic of weakest preconditions:
\( pre = \text{wp}(\text{prog}, \text{post}) \)

Hoare logic of sequential programs:
\( \{pre\} \text{prog} \{post\} \)

Floyd static annotations of flowchart programs

transformer model:
\( \mathbb{P}S \rightarrow \mathbb{P}S \)
(conjunctive)

relational model:
\( S \rightarrow \mathbb{P}S \)

relational model:
\( S \rightarrow S \)

Inspired...
Is modelled by...
Generalises to...

add non-determinism
Program logic

Dijkstra logic of weakest preconditions:
\[ \text{pre} = \text{wp}(\text{prog,post}) \]

Hoare logic of sequential programs:
\[ \{\text{pre}\} \text{prog} \{\text{post}\} \]

Floyd static annotations of flowchart programs

**Has a Galois connection between...**

relational model:
\[ S \rightarrow \mathbb{P}S \]
(conjunctive)

transformer model:
\[ \mathbb{P}S \rightarrow \mathbb{P}S \]

**Inspired...**

Floyd static annotations of flowchart programs

Generalises to...

add non-determinism

Has a Galois connection between...

relational model:
\[ S \rightarrow \mathbb{P}S \]

relational model:
\[ S \rightarrow S \]

transformer model:
\[ \mathbb{P}S \rightarrow \mathbb{P}S \]
(conjunctive)
Program logic

Dijkstra logic of weakest preconditions:
\( pre = \text{wp}(\text{prog}, \text{post}) \)

Hoare logic of sequential programs:
\( \{pre\} \text{prog} \{post\} \)

Floyd static annotations of flowchart programs

transformer model:
\( \mathbb{P}S \rightarrow \mathbb{P}S \)
(conjunctive)

relational model:
\( S \rightarrow \mathbb{P}S \)

relational model:
\( S \rightarrow S \)

add non-determinism

Inspired...
Is modelled by...
Generalises to...
Has a Galois connection between...

transformer model:
\( \mathbb{P}S \rightarrow \mathbb{P}S \)
(conjunctive)

relational model:
\( S \rightarrow \mathbb{P}S \)
Kozen logic of probabilistic programs: 
\{preE\} prog \{postE\}

D. Kozen. 
Semantics of probabilistic programs. 
*Journal of Computer and System Sciences*, 1981

A probabilistic PDL. *Proc. 15th STOC*, ACM, 1983
Program logic

- Dijkstra logic of weakest preconditions: \( pre = wp(prog, post) \)
- Hoare logic of sequential programs: \( \{pre\} prog \{post\} \)
- Kozen logic of probabilistic programs: \( \{preE\} prog \{postE\} \)

- Transformer model: \( \text{fun.} S \rightarrow \text{fun.} S \) (linear)
- Relational model: \( S \rightarrow \text{dist.} S \)
- Dijkstra logic of weakest preconditions: \( PS \rightarrow PS \) (conjunctive)
- Hoare logic of sequential programs: \( S \rightarrow PS \)

Add non-determinism
Add probability

Is modelled by...
Generalises to...
Has a Galois connection between...

Relational model: \( S \rightarrow \text{dist.} S \)
Program logic

- Dijkstra logic of weakest preconditions: $\text{pre} = \text{wp}(\text{prog}, \text{post})$
- Hoare logic of sequential programs: $\{\text{pre}\} \text{ prog} \{\text{post}\}$
- Kozen logic of probabilistic programs: $\{\text{preE}\} \text{ prog} \{\text{postE}\}$

- Transformer model: $\text{fun.}S \rightarrow \text{fun.}S$
  - (linear)

- Relational model: $S \rightarrow \text{dist.}S$

- Relational model: $S \rightarrow S$

- Relational model: $\text{proj}_1 S \rightarrow \text{proj}_1 S$

- Transformer model: $\text{proj}_1 S \rightarrow \text{proj}_1 S$
  - (conjunctive)

Add non-determinism
Add probability

Is modelled by...
Generalises to...
Has a Galois connection between...

relational model: $S \rightarrow \text{dist.}S$

transformer model: $\text{fun.}S \rightarrow \text{fun.}S$
  - (linear)
1983–96...


Combined logic of weakest pre-expectations:
\[ \text{preE} \Rightarrow \text{wp}(\text{prog}, \text{postE}) \]

*add non-determinism and probability*

Has a Galois connection between...

Generalises to...

Probabilistic models for the guarded command language


J. He, A. McIver. K. Seidel

Probabilistic predicate transformers

ACM TOPLAS 18(3), 1996

C.C. Morgan, A. McIver. K. Seidel
Program logic

Dijkstra logic of weakest preconditions:
\[ pre = \text{wp}(\text{prog}, \text{post}) \]

Hoare logic of sequential programs:
\[ \{\text{pre}\} \text{ prog } \{\text{post}\} \]

Kozen logic of probabilistic programs:
\[ \{\text{preE}\} \text{ prog } \{\text{postE}\} \]

Combined logic of weakest pre-expectations:
\[ \text{preE} \Rightarrow \text{wp}(\text{prog}, \text{postE}) \]

Has a Galois connection between...

Is modelled by...
Generalises to...

add non-determinism

add probability

add non-determinism and probability
Probability, abstraction and refinement

Quantitative mu-calculus; two-player games

Probabilistic process algebras

- Transformer model: \( PS \rightarrow PS \) (conjunctive)
- Relational model: \( S \rightarrow PS \)
- Relational model: \( S \rightarrow S \)
- Relational model: \( S \rightarrow dist.S \)
- Transformer model: \( fun.S \rightarrow fun.S \) (linear)

- Transformer model: \( fun.S \rightarrow fun.S \) (sublinear)
- Relational model: \( S \rightarrow P(dist.S) \)

Combined logic of weakest pre-expectations: \( preE \Rightarrow wp(prog, postE) \)

Probability, abstraction and refinement

Quantitative mu-calculus; two-player games

Combined logic of weakest pre-expectations: \( preE \Rightarrow wp(prog, postE) \)

Probabilistic process algebras

- Transformer model: \( fun.S \rightarrow fun.S \) (linear)
- Relational model: \( S \rightarrow dist.S \)
- Relational model: \( S \rightarrow S \)
- Relational model: \( S \rightarrow PS \)
- Transformer model: \( PS \rightarrow PS \) (conjunctive)

- Transformer model: \( fun.S \rightarrow fun.S \) (sublinear)
- Relational model: \( S \rightarrow P(dist.S) \)

- Probability, abstraction and refinement
- Quantitative mu-calculus; two-player games
- Combined logic of weakest pre-expectations: \( preE \Rightarrow wp(prog, postE) \)
- Probabilistic process algebras
- ...
What is the **probability** that the **program**

\[
\text{coin} := \text{heads} \begin{array}{c} \oplus \text{tails} \\ 1/2 \end{array}
\]

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

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

We can abbreviate “\(\text{coin} := \text{heads} \begin{array}{c} \oplus \text{tails} \\ 1/2 \end{array}\text{coin} := \text{tails}\)” as just

\[
\text{coin} := \text{heads} \begin{array}{c} \oplus \text{tails} \\ 1/2 \end{array}
\]

because the left-hand sides “\(\text{coin} :=\)” are the same.
What is the probability that the program

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

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

In the program logic we write

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

to say that the probability is just 1/2.


Probabilistic-program logic: introduction

program (fragment)

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

non-negative real-valued expression over program variables
We will look at these in turn: what we need to know for each type of program fragment \textit{prog} is

What is \textit{wp.prog.B} for arbitrary postcondition \textit{B}?

The usual technique for setting this out is \textit{structurally} over the syntax of the programming language.

Interpretation: assignments

\( x := E \)

Assign the value of expression \( E \) to the variable \( x \).

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

Syntactic substitution.

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

Why are these here?

Informal description.

Definition.

Example.

Interpretation: assignments

Assign the value of expression \( E \) to the variable \( x \).
Interpretation: embedding Booleans

\[ wp.(x:= x+1).\{x=3\} \]
\[ \equiv \ [x=3]\langle x:= x+1 \rangle \quad \text{definition} \]
\[ \equiv \ [(x+1)=3] \quad \text{substitution} \]
\[ \equiv \ \boxed{[x=2]} \quad \text{arithmetic} \]

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

Thus "[ ]" must be an embedding function that takes true to one and false to zero.
Interpretation: probabilistic choice

\( \text{prog}_1 \ p \oplus \ \text{prog}_2 \quad \text{Execute the left-hand side with probability } p, \text{ otherwise execute the right-hand side (probability } 1-p). \)

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

\[
wp.(c := H_{1/2} \oplus T).[c=H]
\equiv \frac{1}{2} \times wp.(c := H).[c=H] + (1-\frac{1}{2}) \times wp.(c := T).[c=H]
\equiv \frac{1}{2} \times [H=H] + \frac{1}{2} \times [T=H]
\equiv \frac{1}{2} \times 1 + \frac{1}{2} \times 0
\equiv \frac{1}{2}.
\]
Interpretation: deterministic choice

\[
\text{if } G \text{ then } \text{prog} \text{ fi}
\]
If guard \( G \) holds, then execute the body \( \text{prog} \); otherwise do nothing.

\[
\text{if } G \text{ then } \text{prog} \text{ else skip fi}
\]
If guard \( G \) holds, then execute \( \text{prog}_1 \); otherwise execute \( \text{prog}_2 \).

\[
x := x
\]

\[
\text{skip}
\]
Do nothing.

\[
\text{if } G \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi}
\]
If \( G \) holds, then go left with probability 1, and vice versa.

\[
\text{prog}_1 \oplus [G] \text{ prog}_2
\]
Interpretation: deterministic choice

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

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

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

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

\[ \equiv [x \geq 1] \triangleleft [x \geq 3] \bigoplus [x < 1] \triangleleft [x \geq 0] \]

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

For a \textit{standard} conditional, the reasoning is just “as usual”.
Interpretation: sequential composition

\[ \text{prog}_1 ; \text{prog}_2 \quad \text{Execute the first program; then execute the second.} \]

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

\[ \text{wp.}(c:= H_{1/2} \oplus T ; \; d:= H_{1/2} \oplus T).[c=d] \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). (\underbrace{\text{wp.}(d:= H_{1/2} \oplus T).[c=d]}) \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). (1/2 \times [c=H] + 1/2 \times [c=T]) \]
\[ \equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) \]
\[ + 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \]
\[ \equiv 1/4 + 1/4 \quad \text{prob. choice; assignment} \]
\[ \equiv 1/2. \quad \text{embedding} \]

\[ \text{wp.}(c:= H_{1/2} \oplus T ; \; d:= H_{1/2} \oplus T).[c=d] \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). (\underbrace{\text{wp.}(d:= H_{1/2} \oplus T).[c=d]}) \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). (1/2 \times [c=H] + 1/2 \times [c=T]) \]
\[ \equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) \]
\[ + 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \]
\[ \equiv 1/4 + 1/4 \quad \text{prob. choice; assignment} \]
\[ \equiv 1/2. \quad \text{embedding} \]
Interpretation: sequential composition

\[ \text{Execute the first program; then execute the second.} \]

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

\[ \text{wp.}(c:= H_{1/2} \oplus T; \ d:= H_{1/2} \oplus T).[c=d] \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). ( \text{wp.}(d:= H_{1/2} \oplus T).[c=d] ) \]
\[ \equiv \text{wp.}(c:= H_{1/2} \oplus T). ( \frac{1}{2} \times [c=H] + \frac{1}{2} \times [c=T] ) \]
\[ \equiv \frac{1}{2} \times (\frac{1}{2} \times [H=H] + \frac{1}{2} \times [H=T]) + \frac{1}{2} \times (\frac{1}{2} \times [T=H] + \frac{1}{2} \times [T=T]) \]
\[ \equiv \frac{1}{4} + \frac{1}{4} \]
\[ \equiv \frac{1}{2} . \]
**Interpretation: a proper extension**

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

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

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

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

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.
**Interpretation: a proper extension**

The expression \( wp.\text{prog}.B \) gives, as a function of the initial state, the *expected value* of the “post-expectation” \( B \) over the distribution of final states that \( \text{prog} \) will produce from there.

We call it the *greatest pre-expectation* of \( \text{prog} \) with respect to \( B \). When \( \text{prog} \) 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 \( wp.\text{prog}.[\text{pred}] \) is the probability that \( \text{pred} \) will be achieved.

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

that is, just \( p \) itself.

**Expected value of \([\text{pred}]\) is thus**

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

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

**state space**

**Real numbers \( \mathbb{R} \)**

**Expectation \([\text{pred}]\) is 1 on \( \text{pred} \) and 0 elsewhere.**
Interpretation: a conservative extension

We note that the standard logic can be embedded in the probabilistic logic simply by converting all Booleans false, true to the integers 0,1 (a technique familiar to C programmers). The probabilistic wp-logic (greatest pre-expectations) extends the standard wp-logic (weakest preconditions) conservatively in this sense:

If we restrict ourselves to standard programs (i.e. do not use the probabilistic choice operator), then the theorems for those programs are exactly the same as before.

Mathematically this is expressed as follows:

For all standard programs prog, and Boolean postconditions post, we have

\[ [wp.prog.post] \equiv wp.prog.[post] \]

where on the left the wp is weakest precondition, and on the right it is greatest pre-expectation.
\[
\begin{align*}
wp.\text{abort}.postE & := 0 \\
wp.\text{skip}.postE & := postE \\
wp.(x:= \text{expr}).postE & := postE \ (x \mapsto \text{expr}) \\
wp.(\text{prog}; \text{prog}').postE & := wp.\text{prog}.(wp.\text{prog}'.postE) \\
wp.(\text{prog} \cap \text{prog}').postE & := \text{wp.\text{prog}.postE} \ \text{min} \ \text{wp.\text{prog}'.postE} \\
wp.(\text{prog} \oplus \text{prog}').postE & := p \ast \text{wp.\text{prog}.postE} + \overline{p} \ast \text{wp.\text{prog}'.postE}
\end{align*}
\]

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 $\mathbb{ES}$ 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

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

Figure 1.5.3. Probabilistic $wp$-semantics of $pGCL$
Interpretation: demonic choice

\[ \text{prog}_1 \sqcap \text{prog}_2 \]

Execute the left-hand side — or maybe execute the right-hand side. Whatever...

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

\[
wp.(c := H \sqcap c := T).[c=H]
\]

\[
\equiv \ wp.(c := H).[c=H] \ \textbf{min} \ wp.(c := T).[c=H]
\]

\[
\equiv [H=H] \ \textbf{min} \ [T=H]
\]

\[
\equiv 1 \ \textbf{min} \ 0
\]

\[
\equiv 0. 
\]

Although the program might achieve \( c=H \), the largest probability of that which can be guaranteed... is zero.
Interpretation: a geometric view

An unbiased coin.

Interpretation: a geometric view

A heads-biased coin.

(2/3, 1/3)
Interpretation: a geometric view

A tails-biased coin.
Interpretation: a geometric view

Demonic choice between these

A biased coin, up to 1/6 either way...

... one refinement of which is an unbiased coin.
Interpretation: a geometric view

A possibly nonterminating coin... whose refinements include all three coins before.
Demonically, either of two possibly nonterminating coins.
Demonically, either of two possibly nonterminating coins.
Demonically, either of two possibly nonterminating coins.

Demonically, either of two possibly nonterminating coins.


Interpretation: a geometric view ... but what’s the connection with the programming logic?
Demonically, either of two possibly nonterminating coins.

\[x + 2y = 0\]
\[x + 2y = \frac{1}{3} + 2\left(\frac{1}{4}\right) = \frac{5}{6}\]

What is the greatest guaranteed expected value of this program with respect to the post-expectation \([c=H] + 2[c=T]\)?

It's 5/6.

... but what's the connection with the programming logic?

Interpretation: a geometric view
Demonically, either of two possibly nonterminating coins.

$\begin{align*}
x + 2y &= \frac{1}{3} + 2(\frac{1}{4}) \\
&= \frac{5}{6}
\end{align*}$

$\begin{align*}
x + 2y &= 0 \\
\Rightarrow \quad (x, y) &= (\frac{1}{4}, \frac{1}{3}) \\
\Rightarrow \quad (x, y) &= (\frac{1}{3}, \frac{1}{4})
\end{align*}$

What is the greatest guaranteed expected value of this program with respect to the post-expectation $2[c=H] + [c=T]$?

Interpretation: a geometric view

... but what's the connection with the programming logic?
Demonically, either of two possibly nonterminating coins.

\[
2x + y = 2/3 + 1/4 = 11/12
\]

What is the greatest guaranteed expected value of this program with respect to the post-expectation?

\[2[c=H] + [c=T] \quad ?\]

It’s not \(11/12\).
What is the greatest guaranteed expected value of this program with respect to the post-expectation $2[c=H] + [c=T]$?

It's $5/6$ again, because this time the demon goes to the other extreme.

Interpretation: a geometric view ... but what's the connection with the programming logic?
Metatheorems for iteration

// Implement $p \oplus$ using unbiased random bits only.

\[ x := p; \]
\[ b := \text{true} \oplus \frac{1}{2} \oplus \text{false}; \]
\[ \text{do } b \rightarrow \]
\[ x := 2x - [x \geq 1/2]; \]
\[ b := \text{true} \oplus \frac{1}{2} \oplus \text{false}; \]
\[ \text{od}; \]
Metatheorems for iteration

// Implement $p \oplus$ using unbiased random bits only.

\[
x := p;
\]

\[
b := \text{true}_{1/2} \oplus \text{false};
\]

\[
\textbf{do} \quad b \rightarrow 
\]

\[
x := 2x - [x \geq 1/2];
\]

\[
b := \text{true}_{1/2} \oplus \text{false};
\]

\[
\textbf{od};
\]

\[
\textbf{if} \quad x \geq 1/2
\]

\[
\textbf{then} \quad \text{prog}_1
\]

\[
\textbf{else} \quad \text{prog}_2
\]

\[
\textbf{fi}
\]

// Variable $x$ at least $1/2$ with probability exactly $p$.

Example due to Joe Hurd (Cambridge, now Oxford).

Metatheorems for iteration

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

is implemented by

and on the average uses only two random bits.
Iteration: reminder of standard metatheorems

\begin{verbatim}
x,b,e := 1,B,E;
do e \neq 0 \rightarrow
   if even e
      then b,e := b^2, e \div 2
   else e,x := e-1, x \times b
fi
od
\end{verbatim}

Set \(x\) to \(B^E\) in logarithmic time.


EW Dijkstra, 1975.
Gries, Backhouse, Kaldewaij, Cohen...
Standard metatheorems: invariants

\{ B > 0 \text{ and } E \geq 0 \}

\begin{align*}
x, b, e &:= 1, B, E; \\
\{ b > 0 \text{ and } e \geq 0 \text{ and } B^E = x \times b^e \} \\
d &\begin{array}{c}
\text{do } e \neq 0 \\
\{ \ldots \text{ and } e > 0 \}
\end{array} \\
&\begin{array}{c}
\text{if even } e \\
\{ \ldots \text{ and } e > 0 \}
\end{array} \\
&\begin{array}{c}
\text{then } \{ e \geq 2 \text{ and even } e \ldots \} \\
\{ b, e := b^2, \; e \div 2 \} \\
\{ B^E = x \times b^e \} \\
\ldots \text{ and } B^E = x \times b^e \}
\end{array} \\
&\begin{array}{c}
\text{else } \{ B^E = x \times b^e \} \\
\{ e, x := e-1, \; x \times b \} \\
\{ B^E = x \times b^e \} \\
\end{array} \\
&\begin{array}{c}
\text{fi} \\
\{ B^E = x \times b^e \}
\end{array} \\
&\begin{array}{c}
\text{od} \\
\{ x = B^E \}
\end{array}
\end{align*}
Standard metatheorems: invariants

\{	extit{pre}\}\}

\textit{init;}

\{	extit{inv}\}\}

\textbf{do}\ G \rightarrow

\{	extit{G} \land \textit{inv}\}\}

\textit{body}

\{	extit{inv}\}\}

\textbf{od}

\{
eg G \land \textit{inv}\}\}
Probabilistic metatheorems: invariants again

\{ pre \} 
init;
\{ inv \}
do G →
\{ [G] × inv \} 
\text{body}
\{ inv \}
od 
\{ [¬G] × inv \} 

\{ pre \} 
init;
\{ inv \}
do G →
\{ G ∧ inv \} 
\text{body}
\{ inv \}
od 
\{ ¬G ∧ inv \}
Iteration: probabilistic example

\{ ? \}
\begin{align*}
x &:= p; \\
b &:= \text{true}_{1/2} \oplus \text{false}; \\
\textbf{do} & \quad b \rightarrow \\
\quad x &:= 2x - [x \geq 1/2]; \\
\quad b &:= \text{true}_{1/2} \oplus \text{false}; \\
\textbf{od}
\end{align*}
\{ [x \geq 1/2] \}

What is the probability that \( x \) exceeds 1/2 on termination?
Example: iteration achieves its goal on termination

\[
[x \geq 1/2] \\
\equiv [-b] \times \\
(2x - [x \geq 1/2] \ltstile{\boxdot}{\boxdot}{b} [x \geq 1/2] ) \\
\]  

... if \ b \ else ...  

\[
x:= p; \\
b:= \text{true } 1/2 \oplus \text{false}; \\
do \ b \rightarrow \\
x:= 2x - [x \geq 1/2]; \\
b:= \text{true } 1/2 \oplus \text{false}; \\
\{ 2x - [x \geq 1/2] \ltstile{\langle}{\rangle}{b} \} \\
\text{od} \\
\{ [x \geq 1/2] \} \\
\]

Example: iteration body preserves invariant

\[
\begin{align*}
\text{do} & \quad b \rightarrow \\
\equiv & \quad x := p; \\
\equiv & \quad b := \text{true}_{1/2} \oplus \text{false} \\
\equiv & \quad b := \text{true}_{1/2} \oplus \text{false} \\
\equiv & \quad (2x - [x \geq 1/2] \triangleleft b \triangleright [x \geq 1/2]) \\
\equiv & \quad (2x)/2 \\
\equiv & \quad x
\end{align*}
\]
Example: iteration properly *initialised*

Assignment;
loop initialisation;
and then we repeat the earlier step.
And finally we see that the pre-expectation overall...

is just $p$.  

Example: correct overall

\[
\begin{align*}
\{ p \} \\
x := p; \\
\{ x \}
\end{align*}
\]

\[
b := \text{true}_{1/2} \oplus \text{false};
\]

\[
\begin{align*}
\{ 2x - [x \geq 1/2] \triangleleft b \triangleright \\
\textbf{do} \quad b \rightarrow \quad [x \geq 1/2] \} \\
\{ 2x - [x \geq 1/2] \} \\
x := 2x - [x \geq 1/2]; \\
\{ x \}
\end{align*}
\]

\[
b := \text{true}_{1/2} \oplus \text{false};
\]

\[
\begin{align*}
\{ 2x - [x \geq 1/2] \triangleleft b \triangleright \\
\textbf{od} \quad [x \geq 1/2] \} \\
\{ [x \geq 1/2] \}
\end{align*}
\]
The probability that the program establishes $x \geq 1/2$ is just $p$.

The loop invariant was

$2x - [x \geq 1/2] \ll b \gg [x \geq 1/2]$

“established” by the initialisation and “maintained” by the body.
Termination of probabilistic iterations

\[
\{ \text{inv} \} \quad \text{do} \quad G \rightarrow \quad \left\{ \begin{array}{l}
[G] \times \text{inv} \\
\text{body}
\end{array} \right\} \quad \{ \text{inv} \} \\
\text{od} \quad \{ \neg G \times \text{inv} \}
\]

\[
\{ \text{inv} \} \quad \text{do} \quad G \rightarrow \quad \left\{ \begin{array}{l}
G \land \text{inv} \\
\text{body}
\end{array} \right\} \quad \{ \text{inv} \} \\
\text{od} \quad \{ \neg G \land \text{inv} \}
\]

In addition, show that \( \text{inv} \Rightarrow \text{term} \), where \( \text{term} \) is the probability of termination ...

... in which case the conclusion \( \{ \text{inv} \} \text{ do } \cdots \text{ od } \{ \neg G \times \text{inv} \} \) expresses total — rather than just partial — correctness.
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

\[ c := H_{1/2} \oplus T; \quad d := H_p \oplus T \quad \text{for arbitrary } p, \]

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

Consider the two programs

\begin{align*}
A: & \quad \text{coin:= edge} \sqcap (\text{coin:= heads } 1/2 \oplus \text{coin:= tails}) \\
B: & \quad (\text{coin:= edge} \sqcap \text{coin:= heads}) \\
       & \quad 1/2 \oplus (\text{coin:= edge} \sqcap \text{coin:= tails}),
\end{align*}

which we will 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. (This can be shown by tabulation: there are only eight possible postconditions.)

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 “;” .) Why is this a problem?

More generally, let A and B be any two programs that are 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 in terms of its possibly being a “minimal complication”?