Reasoning about Probabilistic Programs

Oregon PL Summer School 2021

Justin Hsu
UW–Madison
Cornell University

Day 1: Introducing Probabilistic Programs

- Motivations and key questions
- Mathematical preliminaries

Day 2: First-Order Programs 1

- Probabilistic While language, monadic semantics
- Weakest pre-expectation calculus

Day 3: First-Order Programs 2

- Probabilistic While language, transformer semantics
- Probabilistic separation logic

Day 4: Higher-Order Programs

- Type system: probability monad
- Type system: probabilistic PCF

Last time: pWhile programs

Can you guess what this program does?

$$\begin{array}{l} r \leftarrow 0; \\ \text{while } r < 4 \text{ do} \\ \qquad r \xleftarrow{\$} \mathbf{Roll} \end{array}$$

Last time: PWHILE programs

Can you guess what this program does?

$$\begin{array}{l} r \leftarrow 0; \\ \text{while } r < 4 \text{ do} \\ \qquad r \xleftarrow{\$} \mathbf{Roll} \end{array}$$

Uniform sample from $\{4, 5, 6\}$

- Start with dice roll, condition on $r \geq 4$

More formally: pWhile expressions

Grammar of boolean and numeric expressions

$$\begin{aligned}
\mathcal{E} \ni e &\coloneqq x \in \mathcal{X} && \text{(variables)} \\
&\mid b \in \mathbb{B} \mid \mathcal{E} > \mathcal{E} \mid \mathcal{E} = \mathcal{E} && \text{(booleans)} \\
&\mid n \in \mathbb{N} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} \cdot \mathcal{E} && \text{(numbers)}
\end{aligned}$$

Basic expression language

- Expression language can be extended if needed
- Assume: programs only use well-typed expressions

More formally: pWHILE d-expressions

Grammar of d-expressions

$$\begin{aligned} \mathcal{DE} \ni d &:= \mathbf{Flip} && \text{(fair coin flip)} \\ &\mid \mathbf{Flip}(p) && (p\text{-biased coin flip, } p \in [0,1]) \\ &\mid \mathbf{Roll} && \text{(fair dice roll)} \end{aligned}$$

“Built-in” or “primitive” distributions

- Distributions can be extended if needed
- “Mathematically standard” distributions
- Distributions that can be sampled from in hardware

More formally: pWHILE commands

Grammar of commands

$$
\begin{array}{rll}
\mathcal{C} \ni c := & \mathsf{skip} & \text{(do nothing)} \\
\mid & \mathcal{X} \leftarrow \mathcal{E} & \text{(assignment)} \\
\mid & \mathcal{X} \xleftarrow{\$} \mathcal{D}\mathcal{E} & \text{(sampling)} \\
\mid & \mathcal{C} \,;\, \mathcal{C} & \text{(sequencing)} \\
\mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{(if-then-else)} \\
\mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{(while-loop)}
\end{array}
$$

Imperative language with sampling

- Bare-bones imperative language
- Many possible extensions: procedures, pointers, etc.

A First Semantics for pWhile

Monadic Semantics

Program states

Programs modify memories

- Memories m assign a value $v \in \mathcal{V}$ to each variable $x \in \mathcal{X}$
- Just like memories in imperative languages

Program states

Programs modify memories

- Memories m assign a value $v \in \mathcal{V}$ to each variable $x \in \mathcal{X}$
- Just like memories in imperative languages

More formally:

$$m \in \mathcal{M} \triangleq \mathcal{X} \to \mathcal{V}$$

Semantics of expressions

The value of an expression depends on the memory

- Example: value of $x + 1$ depends on the memory m
- Semantics of expressions takes memory as parameter

Semantics of expressions

The value of an expression depends on the memory

- Example: value of $x + 1$ depends on the memory m
- Semantics of expressions takes memory as parameter

More formally:

$$[\![-]\!] : \mathcal{E} \to \mathcal{M} \to \mathcal{V}$$

For example:

- Expression $x + 1$
- Memory m with $m(x) = 3$
- $[\![x + 1]\!]m \triangleq [\![x]\!]m + [\![1]\!]m \triangleq m(x) + 1 = 3 + 1 = 4$

Semantics of distributions

Semantics of d-expression is distribution over values

- From d-expression to a (mathematical) distribution
- (Easy) extension: d-expression with parameters

More formally:

$$[\![-]\!] : \mathcal{DE} \to \mathsf{Distr}(\mathcal{V})$$

For example:

- D-expression $\mathbf{Flip}$
- $[\![\mathbf{Flip}]\!] \triangleq \mu \in \mathsf{Distr}(\mathbb{B})$, where $\mu(tt) = \mu(ff) = 1/2$

Monadic semantics of commands: overview

First choice:

1. Command takes a memory as input, or:
2. Command takes a distribution over memories as input?

Monadic semantics of commands: overview

First choice:

1. Command takes a memory as input, or:
2. Command takes a distribution over memories as input?

Monadic semantics of commands: overview

First choice:

1. Command takes a memory as input, or:
2. Command takes a distribution over memories as input?

This lecture: monadic semantics

$$(\!|-|\!) : \mathcal{C} \to \mathcal{M} \to \mathsf{Distr}(\mathcal{M})$$

Command: input memory to output distribution over memories.

Operations on distributions: unit

The simplest possible distribution

Dirac distribution: Probability 1 of producing a particular element, and probability 0 of producing anything else.

Operations on distributions: unit

The simplest possible distribution

Dirac distribution: Probability 1 of producing a particular element, and probability 0 of producing anything else.

Distribution unit

Let $a \in A$. Then $unit(a) \in \mathsf{Distr}(A)$ is defined to be:

$$unit(a)(x) = \begin{cases} 1 & : x = a \\ 0 & : \text{otherwise} \end{cases}$$

Why “unit”? The unit (“return”) of the distribution monad.

Semantics of commands: skip

Intuition

- Input: memory m
- Output: distribution that always returns m

Semantics of commands: skip

Intuition

- Input: memory m
- Output: distribution that always returns m

Semantics of skip

$$(\!|\mathsf{skip}|\!)m \triangleq unit(m)$$

Semantics of commands: assignment

Intuition

- Input: memory m
- Output: distribution that always returns m with $x \mapsto v$, where v is the original value of e in m.

Semantics of commands: assignment

Intuition

- Input: memory m
- Output: distribution that always returns m with $x \mapsto v$, where v is the original value of e in m.

Semantics of assignment

Let $v \triangleq [\![e]\!]m$. Then:

$$(\!|x \leftarrow e|\!)m \triangleq unit(m[x \mapsto v])$$

Operations on distributions: map

Translate each distribution output to something else

Whenever sample x, sample $f(x)$ instead. Transformation map f is deterministic: function $A \to B$.

Distribution map

Let $f : A \to B$. Then $map(f) : \mathsf{Distr}(A) \to \mathsf{Distr}(B)$ takes $\mu \in \mathsf{Distr}(A)$ to:

$$map(f)(\mu)(b) \triangleq \sum_{a \in A : f(a) = b} \mu(a)$$

Probability of $b \in B$ is sum probability of $a \in A$ mapping to b.

Semantics of commands: sampling

Intuition

- Input: memory m
- Draw sample from $[\![d]\!]$, call it v
- Given v, map to updated output memory $m[x \mapsto v]$

Semantics of commands: sampling

Intuition

- Input: memory m
- Draw sample from $[\![d]\!]$, call it v
- Given v, map to updated output memory $m[x \mapsto v]$

Semantics of sampling

Let $f(v) \triangleq m[x \mapsto v]$. Then:

$$(\!|x \xleftarrow{\$} d|\!)m \triangleq map(f)([\![d]\!])$$

Operations on distributions: bind

Sequence two sampling instructions together

Draw a sample x, then draw a sample from a distribution $f(x)$ depending on x. Transformation map f is randomized: function $A \to \mathsf{Distr}(B)$.

Distribution bind

Let $\mu \in \mathsf{Distr}(A)$ and $f : A \to \mathsf{Distr}(B)$. Then $bind(\mu, f) \in \mathsf{Distr}(B)$ is defined to be:

$$bind(\mu, f)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$$

Semantics of commands: sequencing

Intuition

- Input: memory m
- Run first command, get distribution μ_1
- Sample m' from μ_1, bind into second command

Semantics of commands: sequencing

Intuition

- Input: memory m
- Run first command, get distribution μ_1
- Sample m' from μ_1, bind into second command

Semantics of sequencing

$$(\!| c_1 \ ; \ c_2 |\!) m \triangleq bind((\!| c_1 |\!) m, (\!| c_2 |\!))$$

Semantics of commands: conditionals

Intuition

- Input: memory m
- If guard is true in m run c_1, else run c_2
- Note: m is a memory, not a distribution!

Semantics of commands: conditionals

Intuition

- Input: memory m
- If guard is true in m run c_1, else run c_2
- Note: m is a memory, not a distribution!

Semantics of conditionals

$$(\!|\text{if } e \text{ then } c_1 \text{ else } c_2|\!)m \triangleq \begin{cases} (\!|c_1|\!)m & : [\![e]\!]m = tt \\ (\!|c_2|\!)m & : [\![e]\!]m = f\!f \end{cases}$$

Semantics of loops: first try

Intuition

- Input: memory m
- Idea: while e do c should be sequence of if-then-else:

$$(\text{if } e \text{ then } c); \cdots ; (\text{if } e \text{ then } c)$$

Semantics of loops: first try

Intuition

- Input: memory m
- Idea: while e do c should be sequence of if-then-else:

$$(\text{if } e \text{ then } c); \cdots ; (\text{if } e \text{ then } c)$$

- Define loop semantics as limit?

$$(\!|\text{while } e \text{ do } c|\!)m \stackrel{?}{=} \lim_{n\to\infty} (\!|(\text{if } e \text{ then } c)^n|\!)m$$

Semantics of loops: first try

Intuition

- Input: memory m
- Idea: $\text{while } e \text{ do } c$ should be sequence of if-then-else:

$$(\text{if } e \text{ then } c); \cdots ; (\text{if } e \text{ then } c)$$

- Define loop semantics as limit?

$$(\!|\text{while } e \text{ do } c|\!) m \stackrel{?}{=} \lim_{n \to \infty} (\!|(\text{if } e \text{ then } c)^n|\!) m$$

What does this limit mean?

- Say $\mu_n \triangleq (\!|(\text{if } e \text{ then } c)^n|\!) m$
- Each μ_n is a distribution in $\text{Distr}(\mathcal{M})$. Does limit exist?

Intuitive loop semantics: limit may not exist!

Simple example: flipper

$$\text{while } tt \text{ do if } x \text{ then } x \leftarrow \mathit{ff} \text{ else } x \leftarrow tt$$

What does this program do?

Intuitive loop semantics: limit may not exist!

Simple example: flipper

$$\text{while } tt \text{ do if } x \text{ then } x \leftarrow \mathit{ff} \text{ else } x \leftarrow tt$$

What does this program do?

Repeatedly changes x to tt and ff

- Suppose input m has $m(x) = tt$
- Can verify: $\mu_n = \llparenthesis(\text{if } e \text{ then } c)^n\rrparenthesis m$ has all mass on m for even n, and all mass on $m[x \mapsto \mathit{ff}]$ for odd n
- Oscillates: no sensible limit!

Semantics of loops: approximants

Problem with the flipper example: loop not terminating

- Idea: only “count” probability mass that has terminated
- Why? Once loop terminates, it is always terminated
- Terminated states can’t oscillate: values remain constant

Semantics of loops: approximants

Problem with the flipper example: loop not terminating

- Idea: only "count" probability mass that has terminated
- Why? Once loop terminates, it is always terminated
- Terminated states can't oscillate: values remain constant

More formally...

- For $\mu \in \text{Distr}(\mathcal{M})$, define:

$$\mu[e](m) \triangleq \begin{cases} \mu(m) & : [\![e]\!]m = tt \\ 0 & : \text{otherwise} \end{cases}$$

- Erase weight of memories where $e = f\!f$ (not conditioning)

Semantics of loops: limit of approximants

Loop approximants

Idea: mass that has terminated after n iterations

$$\mu_n \triangleq (\llparenthesis(\text{if } e \text{ then } c)^n\rrparenthesis m)[\neg e]$$

Sub-distributions μ_n are increasing in n: for any m',

$$\mu_n(m') \leq \mu_{n+1}(m').$$

Thus limit exists!

Semantics of loops: limit of approximants

Loop approximants

Idea: mass that has terminated after n iterations

$$\mu_n \triangleq (\llparenthesis(\text{if } e \text{ then } c)^n\rrparenthesis m)[\neg e]$$

Sub-distributions μ_n are increasing in n: for any m',

$$\mu_n(m') \leq \mu_{n+1}(m').$$

Thus limit exists!

Finally: define loop semantics

$$\llparenthesis \text{while } c \text{ do } e \rrparenthesis m \triangleq \lim_{n \to \infty} \mu_n$$

Semantics of loops: example

Consider this loop:

$$\begin{array}{l} \text{while } \neg stop \text{ do} \\ \quad t \leftarrow t + 1; \\ \quad stop \xleftarrow{\$} \mathbf{Flip}(1/4) \end{array}$$

Suppose input memory m has $m(t) = 0, m(stop) = \mathit{ff}$

Semantics of loops: example

Consider this loop:

$$\begin{array}{l} \text{while } \neg \mathit{stop} \text{ do} \\ \quad t \leftarrow t + 1; \\ \quad \mathit{stop} \xleftarrow{\$} \mathbf{Flip}(1/4) \end{array}$$

Suppose input memory m has $m(t) = 0, m(\mathit{stop}) = \mathit{ff}$

- After 1 iters: terminates with prob. $1/4$ with $t = 1$
- After 2 iters: terminates with prob. $3/4 \cdot 1/4$ with $t = 2$
- After n iters: terminates with prob. $(3/4)^{n-1} \cdot 1/4$ with $t = n$

Thus approximants are:

$$\mu_n(\llbracket t = k \rrbracket) = (3/4)^{k-1} \cdot 1/4$$

for $k = 1, \ldots, n$. Taking limit as $n \to \infty$ gives loop semantics.

Reasoning about pWHILE Programs

Weakest Pre-Expectation Calculus

Standard programs: Weakest Pre-conditions (Dijkstra)

Given a program and a post-condition, find pre-condition

- Given: program c and post-condition Q
- Find $wp(c, Q)$: general pre-condition that ensures Q holds

To check Q on output, check $wp(c, Q)$ on input

- If input state m satisfies $wp(c, Q)$, then $[\![c]\!]m$ satisfies Q

Example: Weakest Pre-conditions

Example

- Program: $x \leftarrow y$
- Post-condition: $x > 0$

What is the wp?

Example: Weakest Pre-conditions

Example

- Program: $x \leftarrow y$
- Post-condition: $x > 0$

What is the wp?

Answer: $wp(x \leftarrow y, x > 0) = (y > 0)$

Why?

Condition $y > 0$ is the least we need to ensure that $x > 0$ holds after running $x \leftarrow y$.

Example: Weakest Pre-conditions

Example

- Program: $x \leftarrow y; x \leftarrow x + 1$
- Post-condition: $x > 0$

What is the wp?

Example: Weakest Pre-conditions

Example

- Program: $x \leftarrow y; x \leftarrow x + 1$
- Post-condition: $x > 0$

What is the wp?

Answer: $wp(x \leftarrow y; x \leftarrow x + 1, x > 0) = (y > -1)$

Why?

Condition $y > -1$ is the least we need to ensure that $x > 0$ holds after running $x \leftarrow y; x \leftarrow x + 1$.

Example: Weakest Pre-conditions

Example

- Program: if $z > 0$ then $x \leftarrow y; x \leftarrow x + 1$ else $x \leftarrow 5$
- Post-condition: $x > 0$

What is the wp?

Example: Weakest Pre-conditions

Example

- Program: if $z > 0$ then $x \leftarrow y; x \leftarrow x + 1$ else $x \leftarrow 5$
- Post-condition: $x > 0$

What is the wp?

Possible to work out by hand, but getting a bit cumbersome...

How to make computing WP easier?

Idea: compute WP compositionally

- WP of complex command defined in terms of WP for sub-commands

Benefits

- Simplify computation of WP for complicated programs
- WP can be computed "mechanically" (and automatically)

WP Calculus: Skip

Intuition

- Program: skip
- Post-condition: Q
- To ensure Q holds after, Q must hold before

WP Calculus: Skip

Intuition

- Program: skip
- Post-condition: Q
- To ensure Q holds after, Q must hold before

WP for Skip

$$wp(\mathsf{skip}, Q) = Q$$

WP Calculus: Assignment

Intuition

- Program: $x \leftarrow e$
- Post-condition: Q
- To ensure Q holds after, Q with $x \mapsto e$ must hold before

WP Calculus: Assignment

Intuition

- Program: $x \leftarrow e$
- Post-condition: Q
- To ensure Q holds after, Q with $x \mapsto e$ must hold before

WP for Assignment

$$wp(x \leftarrow e, Q) = Q[x \mapsto e]$$

WP Calculus: Assignment

Intuition

- Program: $x \leftarrow e$
- Post-condition: Q
- To ensure Q holds after, Q with $x \mapsto e$ must hold before

WP for Assignment

$$wp(x \leftarrow e, Q) = Q[x \mapsto e]$$

Brief check

$$wp(x \leftarrow x + 1, x > 0) = (x + 1 > 0) = (x > -1)$$

WP Calculus: Sequencing

Intuition

- Program: $c_1 \; ; c_2$
- Post-condition: Q
- To ensure Q holds after c_2, $wp(c_2, Q)$ must hold after c_1
- To ensure $wp(c_2, Q)$ holds after c_1, compute another wp

WP Calculus: Sequencing

Intuition

- Program: $c_1 \; ; c_2$
- Post-condition: Q
- To ensure Q holds after c_2, $wp(c_2, Q)$ must hold after c_1
- To ensure $wp(c_2, Q)$ holds after c_1, compute another wp

WP for Sequencing

$$wp(c_1 \; ; c_2, Q) = wp(c_1, wp(c_2, Q))$$

WP Calculus: Conditionals

Intuition

- Program: if e then c_1 else c_2
- Post-condition: Q
- To ensure Q holds after, $wp(c_1, Q)$ must hold before if $e = tt$, and $wp(c_2, Q)$ must hold before if $e = ff$

WP Calculus: Conditionals

Intuition

- Program: if e then c_1 else c_2
- Post-condition: Q
- To ensure Q holds after, $wp(c_1, Q)$ must hold before if $e = tt$, and $wp(c_2, Q)$ must hold before if $e = \mathit{ff}$

WP for Conditionals

$$wp(\text{if } e \text{ then } c_1 \text{ else } c_2, Q) = (e \to wp(c_1, Q)) \land (\neg e \to wp(c_2, Q))$$

Example: using the WP calculus

Example

- Program: if $z > 0$ then $x \leftarrow y; x \leftarrow x + 1$ else $x \leftarrow 5$
- Post-condition: $x > 0$

Example: using the WP calculus

Example

- Program: if $z > 0$ then $x \leftarrow y; x \leftarrow x + 1$ else $x \leftarrow 5$
- Post-condition: $x > 0$

What is the wp? A bit ugly, but entirely mechanical:

$$
\begin{aligned}
& wp(\text{if } z > 0 \text{ then } x \leftarrow y; x \leftarrow x + 1 \text{ else } x \leftarrow 5, x > 0) \\
&= (z > 0 \rightarrow wp(x \leftarrow y; x \leftarrow x + 1, x > 0)) \\
&\qquad \wedge (z \leq 0 \rightarrow wp(x \leftarrow 5, x > 0)) \\
&= (z > 0 \rightarrow wp(x \leftarrow y; x > -1)) \wedge (z \leq 0 \rightarrow 5 > 0)) \\
&= (z > 0 \rightarrow y > -1)
\end{aligned}
$$

What is WP for loops?

Problem: WP for loops is not easy to compute

- Defined in terms of a least fixed-point
- Might have to unroll loop arbitrarily far to compute wp

What is WP for loops?

Problem: WP for loops is not easy to compute

- Defined in terms of a least fixed-point
- Might have to unroll loop arbitrarily far to compute wp

Idea: we often don't need to compute WP for loops

- Just want to know: does P imply $wp(\text{while } e \text{ do } c, Q)$?
- Use simpler, sufficient conditions to prove this implication

WP for loops: invariant rule

Setup

- Program while e do c
- Pre-condition P, post-condition Q

WP for loops: invariant rule

Setup

- Program while e do c
- Pre-condition P, post-condition Q

If we know I satisfying the invariant conditions...

- $P \to I$
- $I \land \neg e \to Q$
- $I \land e \to wp(c, I)$

then we are done:

$$P \to wp(\text{while } e \text{ do } c, Q)$$

WP for loops: invariant rule

Setup

- Program while e do c
- Pre-condition P, post-condition Q

If we know I satisfying the invariant conditions...

- $P \to I$
- $I \land \neg e \to Q$
- $I \land e \to wp(c, I)$

then we are done:

$$P \to wp(\text{while } e \text{ do } c, Q)$$

What's the catch? Need to magically find an invariant I

- Invariant conditions are easy to check

Example: using the invariant rule

Example

- Program: while $n > 0$ do $n \leftarrow n - 2$
- Pre-condition: $P = (n\%2 = 0 \wedge n \geq 0)$ (n is even)
- Post-condition: $Q = (n = 0)$

Example: using the invariant rule

Example

- Program: while $n > 0$ do $n \leftarrow n - 2$
- Pre-condition: $P = (n\%2 = 0 \land n \geq 0)$ (n is even)
- Post-condition: $Q = (n = 0)$

Invariant:

$$I = (n > 0 \rightarrow n\%2 = 0) \land (n \leq 0 \rightarrow n = 0)$$

Example: using the invariant rule

Example

- Program: while $n > 0$ do $n \leftarrow n - 2$
- Pre-condition: $P = (n\%2 = 0 \wedge n \geq 0)$ (n is even)
- Post-condition: $Q = (n = 0)$

Invariant:

$$I = (n > 0 \rightarrow n\%2 = 0) \wedge (n \leq 0 \rightarrow n = 0)$$

Check these invariant conditions:

- $P \rightarrow I$
- $I \wedge \neg e \rightarrow Q$
- $I \wedge e \rightarrow wp(c, I)$

Generalizing Weakest Preconditions
to Probabilistic Programs

Idea: generalize predicates to expectations

“Real-valued” version of predicates

- Predicate: $P : \mathcal{M} \to \mathbb{B}$
- Expectation: $E : \mathcal{M} \to \mathbb{R}^+$

Idea: generalize predicates to expectations

"Real-valued" version of predicates

- Predicate: $P : \mathcal{M} \to \mathbb{B}$
- Expectation: $E : \mathcal{M} \to \mathbb{R}^+$

Example: numeric expression

- If x, y, z are numeric, then they are all expectations
- Also expressions like $x + y, x \cdot y, \ldots$

Idea: generalize predicates to expectations

"Real-valued" version of predicates

- Predicate: $P : \mathcal{M} \to \mathbb{B}$
- Expectation: $E : \mathcal{M} \to \mathbb{R}^+$

Example: numeric expression

- If x, y, z are numeric, then they are all expectations
- Also expressions like $x + y, x \cdot y, \ldots$

Example: indicator function

- If P is a (binary) predicate, then the indicator function is:

$$[P](m) = \begin{cases} 1 & : P(m) = tt \\ 0 & : P(m) = ff \end{cases}$$

- Turns a predicate into an expectation

What do expectations “mean” in a probabilistic state?

Intuition

- The “value” of a predicate P in a memory m is $[P](m)$: 0 if false, and 1 if true.
- The “value” of an expectation E in a distribution over memories μ is the average of E over μ.

Example: encoding a probability as an expectation

Suppose that:

- μ is a distribution over memories
- E is the expectation $[x = y]$

Then we have:

The probability of $x = y$ in μ is the average of E over μ.

Example: encoding an average as an expectation

Suppose that:

- μ is a distribution over memories
- E is the expectation t, where t is the running time

Then we have:

The average running time in μ is the average of E over μ.

Weakest pre-expectation (Morgan and McIver)

Looks similar to weakest pre-conditions

- Given: probabilistic program c and expectation E
- Find $wpe(c, E)$: an expectation that computes the average value of E in the output distribution after running c

To find average value of E after, evaluate $wpe(c, E)$

- For any input state m, the average value of E in the output distribution $(\!|c|\!)m$ is exactly $wpe(c, E)(m)$.

Tailored to the monadic semantics for pWhile

Key property satisfied by wpe

For any program c, expectation E, and input memory m:

$$wpe(c, E)(m) = \mathbb{E}_{m' \sim (\!| c |\!) m}[E(m')]$$

Tailored to the monadic semantics for pWhile

Key property satisfied by wpe

For any program c, expectation E, and input memory m:

$$wpe(c, E)(m) = \mathbb{E}_{m' \sim (\!|c|\!) m}[E(m')]$$

Expectation evaluated on input

- Input is a single memory m
- Evaluate expectation on the memory

Tailored to the monadic semantics for pWHILE

Key property satisfied by wpe

For any program c, expectation E, and input memory m:

$$wpe(c, E)(m) = \mathbb{E}_{m' \sim (\!|c|\!)m}[E(m')]$$

Expectation evaluated on input

- Input is a single memory m
- Evaluate expectation on the memory

Expectation evaluated on output

- Output is a distribution over memories $(\!|c|\!)m$
- Average the expectation over the output distribution

Example: Reasoning with Weakest Pre-expectation

Example

- Program: $z \xleftarrow{\$} \mathbf{Flip}(p)$
- Expectation: $[z]$

What is the wpe?

Example: Reasoning with Weakest Pre-expectation

Example

- Program: $z \xleftarrow{\$} \mathbf{Flip}(p)$
- Expectation: $[z]$

What is the wpe?

Answer: $wpe(z \xleftarrow{\$} \mathbf{Flip}(p), [z]) = p$

Why?

Average value of $[z]$ after running $z \xleftarrow{\$} \mathbf{Flip}(p)$ is the probability that $z = tt$, which is p.

Example: Reasoning with Weakest Pre-expectation

Example

- Program: $x \xleftarrow{\$} \mathbf{Roll}; y \xleftarrow{\$} \mathbf{Roll}$
- Expectation: $x + y$

What is the wpe?

Example: Reasoning with Weakest Pre-expectation

Example

- Program: $x \xleftarrow{\$} \mathbf{Roll}; y \xleftarrow{\$} \mathbf{Roll}$
- Expectation: $x + y$

What is the wpe?

Answer: $wpe(x \xleftarrow{\$} \mathbf{Roll}; y \xleftarrow{\$} \mathbf{Roll}, x + y) = 7$

Why? Already not so easy to see...

Average value of $x + y$ after running $x \xleftarrow{\$} \mathbf{Roll}; y \xleftarrow{\$} \mathbf{Roll}$ is the average value of x plus the average value of y, which is $3.5 + 3.5 = 7$.