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?

\[
\begin{align*}
r & \leftarrow 0; \\
\text{while } r < 4 \text{ do} \\
& \quad r \leftarrow \text{Roll}
\end{align*}
\]
Can you guess what this program does?

\[
\begin{align*}
r & \leftarrow 0; \\
& \text{while } r < 4 \text{ do} \\
& \quad r \leftarrow \text{Roll}
\end{align*}
\]

Uniform sample from \(\{4, 5, 6\}\)

- Start with dice roll, condition on \(r \geq 4\)
More formally: \texttt{PWHILE} expressions

Grammar of boolean and numeric expressions

$$\mathcal{E} \ni \mathcal{e} := x \in \mathcal{X}$$

$$\mid b \in \mathbb{B} \mid \mathcal{E} > \mathcal{E} \mid \mathcal{E} = \mathcal{E}$$

$$\mid n \in \mathbb{N} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} \cdot \mathcal{E}$$

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**

\[
D \varepsilon \ni d ::= \text{Flip} \quad \text{(fair coin flip)} \\
| \text{Flip}(p) \quad \text{($p$-biased coin flip, $p \in [0, 1]$)} \\
| \text{Roll} \quad \text{(fair dice roll)}
\]

“Built-in” or “primitive” distributions

- Distributions can be extended if needed
- “Mathematically standard” distributions
- Distributions that can be sampled from in hardware
More formally: \texttt{PWHILE} commands

**Grammar of commands**

\[ C \
i C := \text{skip} \quad \text{(do nothing)} \\
\quad | \quad X \leftarrow E \quad \text{(assignment)} \\
\quad | \quad X \leftarrow \$ \text{DE} \quad \text{(sampling)} \\
\quad | \quad C ; C \quad \text{(sequencing)} \\
\quad | \quad \text{if } E \text{ then } C \text{ else } C \quad \text{(if-then-else)} \\
\quad | \quad \text{while } E \text{ do } C \quad \text{(while-loop)} \]

**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:

$$m \in \mathcal{M} \triangleq \mathcal{X} \rightarrow \mathcal{V}$$
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:

\[
\llbracket - \rrbracket : \mathcal{E} \rightarrow \mathcal{M} \rightarrow \mathcal{V}
\]

For example:

- Expression \( x + 1 \)
- Memory \( m \) with \( m(x) = 3 \)
- \( \llbracket x + 1 \rrbracket m \triangleq \llbracket x \rrbracket m + \llbracket 1 \rrbracket m \triangleq m(x) + 1 = 3 + 1 = 4 \)
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:

\[ [\_] : \mathcal{DE} \rightarrow \text{Distr}(\mathcal{V}) \]

For example:

- D-expression Flip
- \([\text{Flip}] \triangleq \mu \in \text{Distr}(\mathcal{B}), \text{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?
First choice:
1. Command takes a memory as input, or:
2. Command takes a distribution over memories as input?

This lecture: monadic semantics

\[
(\|\!\|) : C \to \mathcal{M} \to \text{Distr}(\mathcal{M})
\]

Command: input memory to output distribution over memories.
The simplest possible distribution

Dirac distribution: Probability 1 of producing a particular element, and probability 0 of producing anything else.
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 $\text{unit}(a) \in \text{Distr}(A)$ is defined to be:

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

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**

$$(\text{skip}) m \triangleq \text{unit}(m)$$
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:

$$(\langle x \leftarrow e \rangle m \triangleq \text{unit}(m[x \mapsto v]))$$
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 \( \text{map}(f) : \text{Distr}(A) \to \text{Distr}(B) \) takes \( \mu \in \text{Distr}(A) \) to:

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

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:

\[
\langle x \leftarrow d \rangle m \triangleq \text{map}(f)([d])
\]
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 $A \rightarrow \text{Distr}(B)$.

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

$$\text{bind}(\mu, f)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$$
Semantics of commands: sequencing

Intuition

- Input: memory $m$
- Run first command, get distribution $\mu_1$
- Sample $m'$ from $\mu_1$, bind into second command
Semantics of commands: sequencing

Intuition

- Input: memory $m$
- Run first command, get distribution $\mu_1$
- Sample $m'$ from $\mu_1$, bind into second command

Semantics of sequencing

$$(\langle c_1 ; c_2 \rangle m) \triangleq bind((\langle c_1 \rangle m, \langle c_2 \rangle m, \langle c_2 \rangle m))$$
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

$$\langle \text{if } e \text{ then } c_1 \text{ else } c_2 \rangle m \triangleq \begin{cases} \langle c_1 \rangle m & : [e]m = tt \\ \langle c_2 \rangle m & : [e]m = ff \end{cases}$$
Intuition

- Input: memory \( m \)
- Idea: \( \textbf{while } e \textbf{ do } c \) should be sequence of \texttt{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?

$$\text{(while } e \text{ do } c)\left<m \right> = \lim_{n \to \infty} \text{((if } e \text{ then } c)^n\right|m}$$
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?

$$\langle\text{while } e \text{ do } c\rangle m \overset{?}{=} \lim_{n \to \infty} \langle(\text{if } e \text{ then } c)^n\rangle m$$

What does this limit mean?

- Say $\mu_n \triangleq \langle(\text{if } e \text{ then } c)^n\rangle m$
- Each $\mu_n$ is a distribution in $\text{Distr}(\mathcal{M})$. Does limit exist?
Intuitive loop semantics: limit may not exist!

Simple example: flipper

```
while tt do if x then x ← ff else x ← tt
```

What does this program do?
Simple example: flipper

```
while tt do if x then x ← ff else x ← tt
```

What does this program do?

Repeatedly changes $x$ to $tt$ and $ff$

- Suppose input $m$ has $m(x) = tt$
- Can verify: $\mu_n = \|(\text{if } e \text{ then } c)^n\|_m$ has all mass on $m$ for even $n$, and all mass on $m[x ← ff]$ for odd $n$
- Oscillates: no sensible limit!
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 $e = ff$ (not conditioning)
Loop approximants

Idea: mass that has terminated after \( n \) iterations

\[ \mu_n \triangleq (\langle \langle \text{if } e \text{ then } c \rangle^n \rangle m) [\neg e] \]

Sub-distributions \( \mu_n \) 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

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

Sub-distributions \( \mu_n \) are increasing in \( n \): for any \( m' \),

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

Thus limit exists!

Finally: define loop semantics

\[
\langle \text{while } c \text{ do } e \rangle m \triangleq \lim_{n \to \infty} \mu_n
\]
Semantics of loops: example

Consider this loop:

```plaintext
while ¬stop do
    t ← t + 1;
    stop ← Flip(1/4)
```

Suppose input memory $m$ has $m(t) = 0, m(stop) = ff$
Semantics of loops: example

Consider this loop:

```
while ¬stop do
    t ← t + 1;
    stop ← Flip(1/4)
```

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

- 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:

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

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?

\[ \text{wp}(x \Omega y, x > 0) = (y > 0) \]

Why?
Condition \( y > 0 \) is the least we need to ensure that \( x > 0 \) holds after running \( x \Omega y \).
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

$$wp(\text{skip}, Q) = Q$$
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 $e = \text{tt}$, and $wp(c_2, Q)$ must hold before if $e = \text{ff}$
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 $e = ff$

WP for Conditionals

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

$$wp(\text{if } z > 0 \text{ then } x \leftarrow y; x \leftarrow x + 1 \text{ else } x \leftarrow 5, x > 0)$$

$$= (z > 0 \rightarrow wp(x \leftarrow y; x \leftarrow x + 1, x > 0))$$
$$\quad \land (z \leq 0 \rightarrow wp(x \leftarrow 5, x > 0))$$

$$= (z > 0 \rightarrow wp(x \leftarrow y; x > -1)) \land (z \leq 0 \rightarrow 5 > 0))$$

$$= (z > 0 \rightarrow y > -1)$$
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 $Q$

If we know $I$ satisfying the invariant conditions...

$$I \land \neg e \land wp((c, I))$$

then we are done: $P \land wp(\text{while } e \text{ do } c, Q)$

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

Invariant conditions are easy to check.
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...

- \( P \rightarrow I \)
- \( I \land \neg e \rightarrow Q \)
- \( I \land e \rightarrow \text{wp}(c, I) \)

then we are done:

\[
P \rightarrow \text{wp}(\text{while } e \text{ do } c, Q)
\]
WP for loops: invariant rule

Setup

- Program \( \text{while } e \text{ do } c \)
- Pre-condition \( P \), post-condition \( Q \)

If we know \( I \) satisfying the invariant conditions...

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

then we are done:

\[ P \rightarrow \wp(\text{while } e \text{ do } c, Q) \]

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:** $P = (n \% 2 = 0 \land n \geq 0)$ (*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: $P = (n \% 2 = 0 \land n \geq 0)$ (n is even)
- Post-condition: $Q = (n = 0)$

Invariant:

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

Check these invariant conditions:

- $P \rightarrow I$
- $I \land \neg e \rightarrow Q$
- $I \land e \rightarrow wp(c, I)$
Generalizing Weakest Preconditions to Probabilistic Programs
Idea: generalize predicates to expectations

“Real-valued” version of predicates

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

Example: numeric expression
- If $x, y, z$ are numeric, then they are all expectations
- Also expressions like $x + y, x \land y, ...$

Example: indicator function
- If $P$ is a (binary) predicate, then the indicator function is:
  
  $$I_1 : P(m) = t \Rightarrow 1$$
  $$I_0 : P(m) = f \Rightarrow 0$$

Turns a predicate into an expectation
Idea: generalize predicates to expectations

“Real-valued” version of predicates

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

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

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 $\mu$ is the average of $E$ over $\mu$. 
Example: encoding a probability as an expectation

Suppose that:

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

Then we have:
The probability of $x = y$ in $\mu$ is the average of $E$ over $\mu$. 
Example: encoding an average as an expectation

Suppose that:
- $\mu$ is a distribution over memories
- $E$ is the expectation $t$, where $t$ is the running time

Then we have:
The average running time in $\mu$ is the average of $E$ over $\mu$. 
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 $\langle c \rangle m$ is exactly $wpe(c, E)(m)$. 
Key property satisfied by $wpe$

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

$$wpe(c, E)(m) = \mathbb{E}_{m' \sim \langle c \rangle m}[E(m')]$$
Tailored to the **monadic** semantics for **PWHILE**

**Key property satisfied by** \(wpe\)

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

\[
wpe(c, E)(m) = \mathbb{E}_{m' \sim \langle c \rangle m}[E(m')]
\]

**Expectation evaluated on input**

- Input is a single memory \(m\)
- Evaluate expectation on the memory
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: $z \xleftarrow{\$} \text{Flip}(p)$
- Expectation: $[z]

What is the wpe?
Example: Reasoning with Weakest Pre-expectation

Example
- Program: $z \overset{\$}{\leftarrow} \text{Flip}(p)$
- Expectation: $[z]$

What is the wpe?
Answer: $wpe(z \overset{\$}{\leftarrow} \text{Flip}(p), [z]) = p$

Why?
Average value of $[z]$ after running $z \overset{\$}{\leftarrow} \text{Flip}(p)$ is the probability that $z = tt$, which is $p$. 
Example: Reasoning with Weakest Pre-expectation

Example

- Program: $x \triangleleft \text{Roll}; y \triangleleft \text{Roll}$
- Expectation: $x + y$

What is the wpe?

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

Why? Already not so easy to see...

Average value of $x + y$ after running $x \triangleleft \text{Roll}; y \triangleleft \text{Roll}$ is the average value of $x$ plus the average value of $y$, which is $3.5 + 3.5 = 7$.5.
Example: Reasoning with Weakest Pre-expectation

Example

- Program: $x \leftarrow \text{Roll}; y \leftarrow \text{Roll}$
- Expectation: $x + y$

What is the wpe?

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

Why? Already not so easy to see...

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