

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

Atomic Formulas

Equalities

- $e = e'$ holds in s iff all variables $FV(e, e') \subseteq \text{dom}(s)$, and e is equal to e' with probability 1 in s

Atomic Formulas

Equalities

- $e = e'$ holds in s iff all variables $FV(e, e') \subseteq \text{dom}(s)$, and e is equal to e' with probability 1 in s

Distribution laws

- $[e]$ holds in s iff all variables in $FV(e) \subseteq \text{dom}(s)$
- $\text{Unif}_S[e]$ holds in s iff $FV(e) \subseteq \text{dom}(s)$, and e is uniformly distributed on S (e.g., $S = \mathbb{B}$ is fair coin flip)

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\mu([x \mapsto tt, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto tt, y \mapsto ff]) = 1/4$$

$$\mu([x \mapsto ff, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto ff, y \mapsto ff]) = 1/4$$

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\mu([x \mapsto tt, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto tt, y \mapsto ff]) = 1/4$$

$$\mu([x \mapsto ff, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto ff, y \mapsto ff]) = 1/4$$

Then: μ satisfies $\text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$. Why?

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\mu([x \mapsto tt, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto tt, y \mapsto ff]) = 1/4$$

$$\mu([x \mapsto ff, y \mapsto tt]) = 1/4 \quad \mu([x \mapsto ff, y \mapsto ff]) = 1/4$$

Then: μ satisfies $\text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{aligned} \mu_x &\triangleq \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y &\triangleq \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{aligned}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

$x \neq y$ disjoint

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

Then: μ satisfies $\text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

Then: μ satisfies $\text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

- Next, $\mu_x \models \text{Unif}_{\mathbb{B}}[x]$ and $\mu_y \models \text{Unif}_{\mathbb{B}}[y]$

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

Then: μ satisfies $\mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

- Next, $\mu_x \models \mathbf{Unif}_{\mathbb{B}}[x]$ and $\mu_y \models \mathbf{Unif}_{\mathbb{B}}[y]$

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

Then: μ satisfies $\mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

- Next, $\mu_x \models \mathbf{Unif}_{\mathbb{B}}[x]$ and $\mu_y \models \mathbf{Unif}_{\mathbb{B}}[y]$

Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

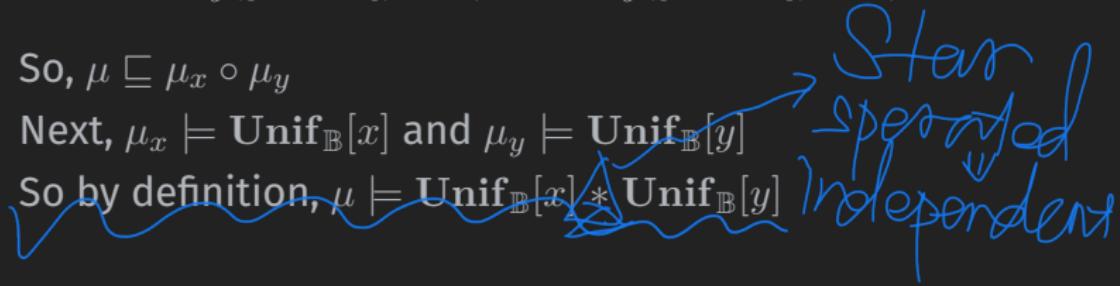
Then: μ satisfies $\text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

- Next, $\mu_x \models \text{Unif}_{\mathbb{B}}[x]$ and $\mu_y \models \text{Unif}_{\mathbb{B}}[y]$
- So by definition, $\mu \models \text{Unif}_{\mathbb{B}}[x] * \text{Unif}_{\mathbb{B}}[y]$



Example: Distribution Assertions

Suppose μ has two variables x, y , indep. fair coin flips

$$\begin{array}{ll} \mu([x \mapsto tt, y \mapsto tt]) = 1/4 & \mu([x \mapsto tt, y \mapsto ff]) = 1/4 \\ \mu([x \mapsto ff, y \mapsto tt]) = 1/4 & \mu([x \mapsto ff, y \mapsto ff]) = 1/4 \end{array}$$

Then: μ satisfies $\mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y]$. Why?

- We can decompose $\mu = \mu_x \otimes \mu_y$, where:

$$\begin{array}{ll} \mu_x([x \mapsto tt]) \triangleq 1/2 & \mu_x([x \mapsto ff]) \triangleq 1/2 \\ \mu_y([y \mapsto tt]) \triangleq 1/2 & \mu_y([y \mapsto ff]) \triangleq 1/2 \end{array}$$

So, $\mu \sqsubseteq \mu_x \circ \mu_y$

- Next, $\mu_x \models \mathbf{Unif}_{\mathbb{B}}[x]$ and $\mu_y \models \mathbf{Unif}_{\mathbb{B}}[y]$
- So by definition, $\mu \models \mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y]$

Example Axioms

Example Axioms

Equality and distributions

- $x = y \wedge \mathbf{Unif}_{\mathbb{B}}[x] \rightarrow \mathbf{Unif}_{\mathbb{B}}[y]$

Example Axioms

Equality and distributions

- $x = y \wedge \mathbf{Unif}_{\mathbb{B}}[x] \rightarrow \mathbf{Unif}_{\mathbb{B}}[y]$

Uniformity and products

- $\mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y] \rightarrow \mathbf{Unif}_{\mathbb{B} \times \mathbb{B}}[x, y]$

Example Axioms

Equality and distributions

- $x = y \wedge \mathbf{Unif}_{\mathbb{B}}[x] \rightarrow \mathbf{Unif}_{\mathbb{B}}[y]$

Uniformity and products

- $\mathbf{Unif}_{\mathbb{B}}[x] * \mathbf{Unif}_{\mathbb{B}}[y] \rightarrow \mathbf{Unif}_{\mathbb{B} \times \mathbb{B}}[x, y]$

Uniformity and exclusive-or (\oplus)

- $\mathbf{Unif}_{\mathbb{B}}[x] * [y] \wedge z = x \oplus y \rightarrow \mathbf{Unif}_{\mathbb{B}}[z] * [y]$

A Probabilistic Separation Logic

Program Logic Judgments in PSL

P and Q from probabilistic BI, c a probabilistic program

$$\{P\} \ c \ \{Q\}$$

Program Logic Judgments in PSL

P and Q from probabilistic BI, c a probabilistic program

$$\{P\} \ c \ \{Q\}$$

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $\llbracket c \rrbracket s$ satisfies the post-condition $\llbracket c \rrbracket s \models Q$.

Program Logic Judgments in PSL

P and Q from probabilistic BI, c a probabilistic program

$$\{P\} \ c \ \{Q\}$$

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $\llbracket c \rrbracket s$ satisfies the post-condition $\llbracket c \rrbracket s \models Q$.

Perfectly fits the ~~transformer~~ semantics for PWHILE

Under transformer semantics:

- P describes: a distribution over memories (input)
- Q describes: a distribution over memories (output)

Under monadic semantics: mismatch!

- P describes: a distribution over memories
- But input to program: a single memory

How do we prove these judgments?

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $\llbracket c \rrbracket s$ satisfies the post-condition $\llbracket c \rrbracket s \models Q$.

How do we prove these judgments?

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $\llbracket c \rrbracket s$ satisfies the post-condition $\llbracket c \rrbracket s \models Q$.

Proving validity directly is difficult

- ▶ Must unfold definition of $\llbracket c \rrbracket$ as a function
- ▶ Then prove property of function by working with definition

How do we prove these judgments?

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $\llbracket c \rrbracket s$ satisfies the post-condition $\llbracket c \rrbracket s \models Q$.

Proving validity directly is difficult

- ▶ Must unfold definition of $\llbracket c \rrbracket$ as a function
- ▶ Then prove property of function by working with definition

Things that would make proving judgments easier:

- ▶ Compositionality: prove property of bigger program by combining proofs of properties of sub-programs
- ▶ Avoid unfolding definition of program semantics

Solution: define a set of proof rules (a proof system)

Each proof rule look like:

$$\frac{\{P_1\} c_1 \{Q_1\} \quad \cdots \quad \{P_n\} c_n \{Q_n\}}{\{P\} c \{Q\}} \text{ RULENAME}$$

Solution: define a set of proof rules (a proof system)

Each proof rule look like:

$$\frac{\{P_1\} c_1 \{Q_1\} \quad \cdots \quad \{P_n\} c_n \{Q_n\}}{\{P\} c \{Q\}} \text{ RULENAME}$$

Proof rules mean:

- To prove $\{P\} c \{Q\}$
- We just have to prove $\{P_1\} c_1 \{Q_1\}, \dots, \{P_n\} c_n \{Q_n\}$

Solution: define a set of proof rules (a proof system)

Each proof rule look like:

$$\frac{\{P_1\} c_1 \{Q_1\} \quad \dots \quad \{P_n\} c_n \{Q_n\}}{\{P\} c \{Q\}} \text{ RULENAME}$$

Proof rules mean:

- To prove $\{P\} c \{Q\}$
- We just have to prove $\{P_1\} c_1 \{Q_1\}, \dots, \{P_n\} c_n \{Q_n\}$

Why do proof rules help?

- Programs c_1, \dots, c_n are smaller/simpler than c
- If c can't be broken down, no premises ($n = 0$)

The Proof System of PSL

Basic Rules

Basic Proof Rules in PSL: Assignment

Assignment Rule

$$\frac{x \notin FV(e)}{\{\top\} x \leftarrow e \{x = e\}} \text{ASSN}$$

Basic Proof Rules in PSL: Assignment

Assignment Rule

$$\frac{x \notin FV(e)}{\{\top\} x \leftarrow e \{x = e\}} \text{ AssN}$$

How to read this rule?

From any initial distribution, running $x \leftarrow e$ will lead to a distribution where x equals e with probability 1 (assuming x doesn't appear in e).

Basic Proof Rules in PSL: Sampling

Sampling Rule

$$\frac{}{\{\top\} x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}} \text{ SAMP}$$

Basic Proof Rules in PSL: Sampling

Sampling Rule

$$\frac{\{ \top \} \quad x \xleftarrow{\$} \text{Flip} \quad \{ \text{Unif}_{\mathbb{B}}[x] \}}{\{ \top \}}$$

How to read this rule?

From any initial distribution, running $x \xleftarrow{\$} \text{Flip}$ will lead to a distribution where x is a uniformly distributed Boolean.

$[x]$ holds in s

Basic Proof Rules in PSL: Sequencing

Sequencing Rule

$$\frac{\{P\} c_1 \{Q\} \quad \{Q\} c_2 \{R\}}{\underbrace{\{P\} c_1 ; c_2 \{R\}}_{\text{SEQ}}} \text{SEQ}$$

Basic Proof Rules in PSL: Sequencing

Sequencing Rule

$$\frac{\{P\} c_1 \{Q\} \quad \{Q\} c_2 \{R\}}{\{P\} c_1 ; c_2 \{R\}} \text{SEQ}$$

How to read this rule?

- ▶ If: from any distribution satisfying P , running c_1 leads to a distribution satisfying R
- ▶ If: from any distribution satisfying R , running c_2 leads to a distribution satisfying Q
- ▶ Then: from any distribution satisfying P , running $c_1 ; c_2$ leads to a distribution satisfying Q

The Proof System of PSL

Conditional Rule

Conditional Rule: first try

Does this rule work?

$$\frac{\{e = tt \wedge P\} \ c \ \{Q\} \quad \{e = ff \wedge P\} \ c' \ \{Q\}}{\{P\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND?}$$

Rule COND? is not sound!

Rule COND? is not sound!

Take P to be $\mathbf{Unif}_{\mathbb{B}}[e]$ and Q to be \perp :

$$\frac{\{e = tt \wedge \mathbf{Unif}_{\mathbb{B}}[e]\} c \{\perp\} \quad \{e = ff \wedge \mathbf{Unif}_{\mathbb{B}}[e]\} c' \{\perp\}}{\{\mathbf{Unif}_{\mathbb{B}}[e]\} \text{ if } e \text{ then } c \text{ else } c' \{\perp\}} \text{COND?}$$

Rule COND? is not sound!

Take P to be $\mathbf{Unif}_{\mathbb{B}}[e]$ and Q to be \perp :

$$\frac{\{e = tt \wedge \mathbf{Unif}_{\mathbb{B}}[e]\} c \{\perp\} \quad \{e = ff \wedge \mathbf{Unif}_{\mathbb{B}}[e]\} c' \{\perp\}}{\{\mathbf{Unif}_{\mathbb{B}}[e]\} \text{ if } e \text{ then } c \text{ else } c' \{\perp\}} \text{ COND?}$$

Premises are valid...

There is no distribution satisfying $e = tt \wedge \mathbf{Unif}_{\mathbb{B}}[e]$ or $e = ff \wedge \mathbf{Unif}_{\mathbb{B}}[e]$, so pre-conditions are \perp and the premises are trivially valid.

Rule COND? is not sound!

Take P to be $\text{Unif}_{\mathbb{B}}[e]$ and Q to be \perp :

$$\frac{\{e = tt \wedge \text{Unif}_{\mathbb{B}}[e]\} c \{\perp\} \quad \{e = ff \wedge \text{Unif}_{\mathbb{B}}[e]\} c' \{\perp\}}{\{\text{Unif}_{\mathbb{B}}[e]\} \text{ if } e \text{ then } c \text{ else } c' \{\perp\}} \text{ COND?}$$

$\xrightarrow{\text{input } u}$
 M conditional on $e = tt$. might not exist

Premises are valid...

There is no distribution satisfying $e = tt \wedge \text{Unif}_{\mathbb{B}}[e]$ or $e = ff \wedge \text{Unif}_{\mathbb{B}}[e]$, so pre-conditions are \perp and the premises are trivially valid.

But the conclusion is not!

It is not the case that if $\text{Unif}_{\mathbb{B}}[e]$ in the input distribution, then running $\text{if } e \text{ then } c \text{ else } c'$ will lead to an impossible output distribution!

What went wrong?

The broken rule

$$\frac{\{e = tt \wedge P\} \ c \ \{Q\} \quad \{e = ff \wedge P\} \ c' \ \{Q\}}{\{P\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND?}$$

What went wrong?

The broken rule

$$\frac{\{e = tt \wedge P\} c \{Q\} \quad \{e = ff \wedge P\} c' \{Q\}}{\{P\} \text{ if } e \text{ then } c \text{ else } c' \{Q\}} \text{ COND?}$$

The problem: conditioning

- We assume: P holds in input distribution μ
- Inputs to branches: μ conditioned on $e = tt$ and $e = ff$
- But: P might not hold on conditional distributions!

Conditional Rule: second try

Does this rule work?

$$\frac{\{e = tt * P\} \ c \ \{Q\} \quad \{e = ff * P\} \ c' \ \{Q\}}{\{[e] * \textcolor{green}{P}\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{COND??}$$

Conditional Rule: second try

Does this rule work?

$$\frac{\{e = tt * P\} \ c \ \{Q\} \quad \{e = ff * P\} \ c' \ \{Q\}}{\{[e] * \textcolor{green}{P}\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND??}$$

Previous counterexample fails

If we take P to be $\mathbf{Unif}_{\mathbb{B}}[e]$, then $[e] * \mathbf{Unif}_{\mathbb{B}}[e]$ is false, and the conclusion is trivially valid.

But rule COND?? is still not sound!

But rule COND?? is still not sound!

Consider this proof

$$\frac{\{e = tt * \top\} x \leftarrow e \{[x] * [e]\} \quad \{e = ff * P\} x \leftarrow e \{[x] * [e]\}}{\{[e] * \top\} \text{if } e \text{ then } x \leftarrow e \text{ else } x \leftarrow e \{[x] * [e]\}}$$
 COND??

But rule COND?? is still not sound!

Consider this proof

$$\frac{\{e = tt * \top\} x \leftarrow e \{[x] * [e]\} \quad \{e = ff * P\} x \leftarrow e \{[x] * [e]\}}{\{[e] * \top\} \text{if } e \text{ then } x \leftarrow e \text{ else } x \leftarrow e \{[x] * [e]\}} \text{ COND??}$$

Premises are valid...

In the output of each branch, x and e are independent since e is deterministic.

But rule COND?? is still not sound!

Consider this proof

$$\frac{\{e = tt * \top\} x \leftarrow e \{[x] * [e]\} \quad \{e = ff * P\} x \leftarrow e \{[x] * [e]\}}{\{[e] * \top\} \text{if } e \text{ then } x \leftarrow e \text{ else } x \leftarrow e \{[x] * [e]\}} \text{ COND??}$$

Premises are valid...

In the output of each branch, x and e are independent since e is deterministic.

But the conclusion is not!

In the output of the conditional, x and e are clearly not always independent: they are equal, and they might be randomized!

What went wrong?

The broken rule

$$\frac{\{e = tt * P\} \ c \ \{Q\} \quad \{e = ff * P\} \ c' \ \{Q\}}{\{[e] * P\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND??}$$

What went wrong?

The broken rule

$$\frac{\{e = tt * P\} \ c \ \{Q\} \quad \{e = ff * P\} \ c' \ \{Q\}}{\{[e] * P\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND??}$$

The problem: mixing

- ▶ Suppose: Q holds in the outputs of both branches
- ▶ The output of the conditional is a **convex combination** of the branch outputs
- ▶ But: Q might not hold in the convex combination!

Conditional Rule in PSL

Fixed rule

$$\frac{\begin{array}{c} \{e = tt * P\} c \{Q\} \\ \{e = ff * P\} c' \{Q\} \\ Q \text{ is closed under mixtures (CM)} \end{array}}{\{[e] * P\} \text{ if } e \text{ then } c \text{ else } c' \{Q\}} \text{ COND}$$

Conditional Rule in PSL

Fixed rule

$$\frac{\begin{array}{c} \{e = tt * P\} c \{Q\} \\ \{e = ff * P\} c' \{Q\} \\ Q \text{ is closed under mixtures (CM)} \end{array}}{\{[e] * P\} \text{ if } e \text{ then } c \text{ else } c' \{Q\}} \text{ COND}$$

Pre-conditions

- ▶ Inputs to branches derived from **conditioning on e**
- ▶ Independence ensures that P holds after conditioning

Conditional Rule in PSL

Fixed rule

$$\frac{\begin{array}{c} \{e = tt * P\} \ c \ \{Q\} \\ \{e = ff * P\} \ c' \ \{Q\} \\ Q \text{ is closed under mixtures (CM)} \end{array}}{\{[e] * P\} \text{ if } e \text{ then } c \text{ else } c' \ \{Q\}} \text{ COND}$$

Pre-conditions

- ▶ Inputs to branches derived from **conditioning** on e
- ▶ Independence ensures that P holds after conditioning

Post-conditions

- ▶ Not all post-conditions Q can be soundly combined
- ▶ “Closed under mixtures” needed for soundness

CM properties: Closed under Mixtures

An assertion Q is CM if it satisfies:

If $\mu_1 \models Q$ and $\mu_2 \models Q$, then $\mu_1 \oplus_p \mu_2 \models Q$ for any $p \in [0, 1]$.

CM properties: Closed under Mixtures

An assertion Q is CM if it satisfies:

If $\mu_1 \models Q$ and $\mu_2 \models Q$, then $\mu_1 \oplus_p \mu_2 \models Q$ for any $p \in [0, 1]$.

Examples of CM assertions

- ▶ $x = e$
- ▶ $\mathbf{Unif}_{\mathbb{B}}[x]$

CM properties: Closed under Mixtures

An assertion Q is CM if it satisfies:

If $\mu_1 \models Q$ and $\mu_2 \models Q$, then $\mu_1 \oplus_p \mu_2 \models Q$ for any $p \in [0, 1]$.

Examples of CM assertions

- ▶ $x = e$
- ▶ $\mathbf{Unif}_{\mathbb{B}}[x]$

Examples of non-CM assertions

- ▶ $[x] * [y]$
- ▶ $x = 1 \vee x = 2$

Example: using the conditional rule

Consider the program:

`if x then $z \leftarrow \neg y$ else $z \leftarrow y$`

If x is true, negate y and store in z . Otherwise store y into z .

Example: using the conditional rule ✓

Consider the program:

if x then $z \leftarrow \neg y$ else $z \leftarrow y$

If x is true, negate y and store in z . Otherwise store y into z .

Using the conditional rule:

$$\{x = tt * \mathbf{Unif}_{\mathbb{B}}[y]\} z \leftarrow \neg y \{\mathbf{Unif}_{\mathbb{B}}[z]\}$$

$$\{x = ff * \mathbf{Unif}_{\mathbb{B}}[y]\} z \leftarrow y \{\mathbf{Unif}_{\mathbb{B}}[z]\}$$

$\mathbf{Unif}_{\mathbb{B}}[z]$ is closed under mixtures (CM)

$$\frac{}{\{[x] * \mathbf{Unif}_{\mathbb{B}}[y]\} \text{ if } x \text{ then } z \leftarrow \neg y \text{ else } z \leftarrow y \{\mathbf{Unif}_{\mathbb{B}}[z]\}} \text{ COND}$$

The Proof System of PSL

Frame Rule

The Frame Rule in SL

Properties about unmodified heaps are preserved

$$\frac{\{P\} \ c \ \{Q\} \quad c \text{ doesn't modify } FV(R)}{\{P * R\} \ c \ \{Q * R\}} \text{ FRAME}$$

The Frame Rule in SL

Properties about unmodified heaps are preserved

$$\frac{\{P\} \ c \ \{Q\} \quad c \text{ doesn't modify } FV(R)}{\{P * R\} \ c \ \{Q * R\}} \text{ FRAME}$$

So-called “local reasoning” in SL

- ▶ Only need to reason about part of heap used by c
- ▶ Note: **doesn't hold** if $*$ replaced by \wedge , due to aliasing!

Why is the Frame rule important?

Why is the Frame rule important?

In SL: simplify reasoning

- ▶ Program c may only modify a small part of the heap
- ▶ Rest of heap may be complicated (linked lists, trees, etc.)
- ▶ Automatically preserve any assertion about rest of heap, as long as rest of heap is separate from what c touches

Why is the Frame rule important?

In SL: simplify reasoning

- ▶ Program c may only modify a small part of the heap
- ▶ Rest of heap may be complicated (linked lists, trees, etc.)
- ▶ Automatically preserve any assertion about rest of heap, as long as rest of heap is separate from what c touches

In PSL: preserve independence

- ▶ Assume: in input, variable x is independent of what c uses
- ▶ Conclude: in output, x is independent of what c touches

The Frame Rule in PSL

The rule

$$\frac{\{P\} c \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\} c \{Q * R\}} \text{ FRAME}$$

Side conditions

The Frame Rule in PSL

The rule

$$\frac{\{P\} c \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\} c \{Q * R\}} \text{ FRAME}$$

Side conditions

1. Variables in R are not modified

The Frame Rule in PSL

The rule

$$\frac{\{P\} c \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\} c \{Q * R\}} \text{ FRAME}$$

Side conditions

1. Variables in R are not modified
2. P describes all variables that might be read

The Frame Rule in PSL

The rule

$$\frac{\{P\} c \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\} c \{Q * R\}} \text{ FRAME}$$

Side conditions

1. Variables in R are not modified
2. P describes all variables that might be read
3. Everything in Q is freshly written, or in P

The Frame Rule in PSL



The rule

$$\frac{\{P\} c \{Q\} \quad FV(R) \cap MV(c) = \emptyset \quad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\} c \{Q * R\}} \text{ FRAME}$$

Side conditions

1. Variables in R are not modified

2. P describes all variables that might be read

3. Everything in Q is freshly written, or in P

Variables in the Q were independent of R , or

or are newly independent of R

preserve independent from P

newly freshly written.

Example: Deriving a Better Sampling Rule

Original sampling rule:

$$\frac{\{ \top \} \ x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}}{\{ \top \} \ x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}} \text{ SAMP}$$

Frame rule:

$$\frac{\begin{array}{c} \{P\} \ c \ \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c) \end{array}}{\{P * R\} \ c \ \{Q * R\}} \text{ FRAME}$$

Example: Deriving a Better Sampling Rule

Original sampling rule:

$$\frac{\{\top\} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}}{\{\top\} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}} \text{ SAMP}$$

Frame rule:

$$\frac{\begin{array}{c} \{P\} \; c \; \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c) \end{array}}{\{P * R\} \; c \; \{Q * R\}} \text{ FRAME}$$

Can derive:

$$\frac{x \notin FV(R)}{\{R\} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x] * R\}} \text{ SAMP*}$$

Example: Deriving a Better Sampling Rule

Original sampling rule:

$$\frac{\{ \top \} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}}{\{ \top \} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x]\}} \text{ SAMP}$$

Frame rule:

$$\frac{\begin{array}{c} \{P\} \; c \; \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\ \models P \rightarrow [RV(c)] \quad FV(Q) \subseteq RV(c) \cup WV(c) \end{array}}{\{P * R\} \; c \; \{Q * R\}} \text{ FRAME}$$

Can derive:

$$\frac{x \notin FV(R)}{\{R\} \quad x \xleftarrow{\$} \text{Flip } \{\text{Unif}_{\mathbb{B}}[x] * R\}} \text{ SAMP*}$$

Intuitively: fresh random sample is independent of everything

A Probabilistic Separation Logic

Soundness Theorem



Proof rules can only show valid judgments

variables matter for P

Theorem

✓ If $\{P\} \vdash \{Q\}$ is derivable via the proof rules, then $\{P\} \vdash \{Q\}$ is a valid judgment: for all initial distributions μ , if $\mu \models P$ then $\llbracket c \rrbracket \mu \models Q$. Validity

Key property for soundness: restriction

Let P be any formula of probabilistic BI, and suppose that

$s \models P$. Then there exists $s' \sqsubseteq s$ such that $s' \models P$ and $\text{dom}(s') = \text{dom}(s) \cap FV(P)$.

restrict s into the

$FV(P)$

Intuition

- The only variables that “matter” for P are $FV(P)$
- Tricky for implications; proof “glues” distributions

Verifying an Example

One-Time-Pad (OTP)

Possibly the simplest encryption scheme

- ▶ Input: a message $m \in \mathbb{B}$
- ▶ Output: a ciphertext $c \in \mathbb{B}$
- ▶ Idea: encrypt by taking xor with a uniformly random key k

One-Time-Pad (OTP)

Possibly the simplest encryption scheme

- ▶ Input: a message $m \in \mathbb{B}$
- ▶ Output: a ciphertext $c \in \mathbb{B}$
- ▶ Idea: encrypt by taking xor with a uniformly random key k

The encoding program:

$$\begin{aligned} k &\xleftarrow{\$} \textbf{Flip}_9 \\ c &\leftarrow k \oplus m \end{aligned}$$

How to Formalize Security?

How to Formalize Security?

Method 1: Uniformity

- ▶ Show that c is uniformly distributed
- ▶ Always the same, no matter what the message m is

How to Formalize Security?

Method 1: Uniformity

- ▶ Show that c is uniformly distributed
- ▶ Always the same, no matter what the message m is

Method 2: **input-output independence**

- ▶ Assume that m is drawn from some (unknown) distribution
- ▶ Show that c and m are **independent**

Proving Input-Output Independence for OTP in PSL

$$k \xleftarrow{\$} \mathbf{Flip};$$

$$c \leftarrow k \oplus m$$

Proving Input-Output Independence for OTP in PSL

$\{[m]\}$ assumption

$k \xleftarrow{\$} \text{Flip};$

$c \leftarrow k \oplus m$

Proving Input-Output Independence for OTP in PSL

$\{[m]\}$ assumption

$k \xleftarrow{\$} \text{Flip}_{\mathbb{B}}$

$\{[m] * \text{Unif}_{\mathbb{B}}[k]\}$ [SAMP*]

$c \leftarrow k \oplus m$

Proving Input-Output Independence for OTP in PSL

$\{[m]\}$ assumption

$k \xleftarrow{\$} \mathbf{Flip};$

$\{[m] * \mathbf{Unif}_{\mathbb{B}}[k]\}$ [SAMP*]

$c \leftarrow k \oplus m$

$\{[m] * \mathbf{Unif}_{\mathbb{B}}[k] \wedge c = k \oplus m\}$ [ASSN*]

Proving Input-Output Independence for OTP in PSL

$\{[m]\}$

assumption

$k \xleftarrow{\$} \mathbf{Flip};$

$\{[m] * \mathbf{Unif}_{\mathbb{B}}[k]\}$

[SAMP*]

$c \leftarrow k \oplus m$

$\{[m] * \mathbf{Unif}_{\mathbb{B}}[k] \wedge c = k \oplus m\}$

[ASSN*]

$\{[m] * \mathbf{Unif}_{\mathbb{B}}[c]\}$

XOR axiom

PSL: references and further reading

The original paper on probabilistic semantics

Kozen. Semantics of Probabilistic Programs. FOCS 1980.

Unifying survey on Bunched Implications

Docherty. Bunched Logics: A Uniform Approach. PhD Thesis (UCL), 2019.

A Probabilistic Separation Logic (POPL20)

- ▶ Extensions to PSL: deterministic variables, loops, etc.
- ▶ Many examples from cryptography, security of ORAM
- ▶ <https://arxiv.org/abs/1907.10708>

A Bunched Logic for Conditional Independence (LICS21)

- ▶ A BI-style logic called DIBI for conditional independence
- ▶ A separation logic (CPSL) based on DIBI
- ▶ <https://arxiv.org/abs/2008.09231>

Reasoning about Probabilistic Programs Higher-Order Languages

So far: reasoning about PWHILE programs

First part

- ▶ Monadic semantics: $\langle\!\langle c \rangle\!\rangle : \mathcal{M} \rightarrow \text{Distr}(\mathcal{M})$
- ▶ Verification method: weakest pre-expectations (*wpe*)

Second part

- ▶ Transformer semantics: $\llbracket c \rrbracket : \text{Distr}(\mathcal{M}) \rightarrow \text{Distr}(\mathcal{M})$
- ▶ Verification method: probabilistic separation logic (PSL)

Today: probabilistic higher-order programs

What's missing from PWHILE?

- ▶ First-order programs only
- ▶ That is: can't pass functions to other functions

This is OPLSS: where are the functions?

- ▶ How about probabilistic functional languages?
- ▶ What do the type systems look like?

With a Probability Monad: A Simple Functional Language

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 $\text{unit}(a) \in \text{Distr}(A)$ 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.

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 $A \rightarrow \text{Distr}(B)$.

Distribution bind

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

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

Language: probabilistic monadic lambda calculus

Language grammar: core

$$\mathcal{E} \ni e ::= x \in \mathcal{X} \mid \lambda \mathcal{X}. \mathcal{E} \mid \mathcal{E} \mathcal{E} \mid \text{fix } \mathcal{X}. \lambda \mathcal{X}. \mathcal{E} \quad (\text{lambda calc.})$$

Language: probabilistic monadic lambda calculus

Language grammar: core

$\mathcal{E} \ni e ::= x \in \mathcal{X} \mid \lambda \mathcal{X}. \mathcal{E} \mid \mathcal{E} \mathcal{E} \mid \text{fix } \mathcal{X}. \lambda \mathcal{X}. \mathcal{E}$ (lambda calc.)

Language grammar: base types

$\mathcal{E} \ni e ::= \dots \mid b \in \mathbb{B} \mid \text{if } \mathcal{E} \text{ then } \mathcal{E} \text{ else } \mathcal{E}$ (booleans)
 $\mid n \in \mathbb{N} \mid \text{add}(\mathcal{E}, \mathcal{E})$ (numbers)

Language: probabilistic monadic lambda calculus

Language grammar: core

$$\mathcal{E} \ni e ::= x \in \mathcal{X} \mid \lambda \mathcal{X}. \mathcal{E} \mid \mathcal{E} \mathcal{E} \mid \text{fix } \mathcal{X}. \lambda \mathcal{X}. \mathcal{E} \quad (\text{lambda calc.})$$

Language grammar: base types

$$\begin{aligned} \mathcal{E} \ni e ::= & \cdots \mid b \in \mathbb{B} \mid \text{if } \mathcal{E} \text{ then } \mathcal{E} \text{ else } \mathcal{E} & (\text{booleans}) \\ & \mid n \in \mathbb{N} \mid \text{add}(\mathcal{E}, \mathcal{E}) & (\text{numbers}) \end{aligned}$$

Language grammar: probabilistic part

$$\begin{aligned} \mathcal{E} \ni e ::= & \cdots \mid \text{Flip} \mid \text{Roll} & (\text{distributions}) \\ & \mid \text{return}(\mathcal{E}) & (\text{unit}) \\ & \mid \text{sample } \mathcal{X} = \mathcal{E} \text{ in } \mathcal{E} & (\text{bind}) \end{aligned}$$

Example programs

Sum of two dice rolls

```
sample x = Roll in  
sample y = Roll in  
return(add(x, y))
```

Example programs

Sum of two dice rolls

```
sample  $x = \text{Roll}$  in  
sample  $y = \text{Roll}$  in  
return(add( $x, y$ ))
```

Geometric distribution

```
(fix  $geo.$   $\lambda n.$   
  sample  $stop = \text{Flip}$  in  
  if  $stop$  then return( $n$ ) else  $geo$  add( $n, 1$ )) 0
```

Operational semantics: setup

One-step reduction

The one-step relation $\rightarrow : \mathcal{CE} \rightarrow \text{SDistr}(\mathcal{CE})$ maps closed expressions to sub-distributions on closed expressions. Read:

$$e \rightarrow \mu$$

as “ e steps to sub-distribution μ on expressions in one step”.

Operational semantics: setup

One-step reduction

The one-step relation $\rightarrow : \mathcal{CE} \rightarrow \text{SDistr}(\mathcal{CE})$ maps closed expressions to sub-distributions on closed expressions. Read:

$$e \rightarrow \mu$$

as “ e steps to sub-distribution μ on expressions in one step”.

Multi-step reduction

For every $n \in \mathbb{N}$, the multi-step relation $\Rightarrow_n : \mathcal{CE} \rightarrow \text{SDistr}(\mathcal{CE})$ maps closed expressions to sub-distributions on closed expressions. Read:

$$e \Rightarrow_n \mu$$

as “ e steps to sub-distribution μ on values in exactly n steps”.

Operational semantics: non-probabilistic part

Standard call-by-value semantics

$(\lambda x. e) v \rightarrow unit(e[v/x])$

if tt then e else e' $\rightarrow unit(e)$

if ff then e else e' $\rightarrow unit(e')$

$(\text{fix } f. \lambda x. e) v \rightarrow unit(e[(\text{fix } f. \lambda x. e)/f][v/x])$

$\text{add}(n, n') \rightarrow unit(n + n')$

...

Operational semantics: primitive distributions

Notation

We write $\{v_1 : p_1, \dots, v_n : p_n\}$ or $\{v_i : p_i\}_{i \in I}$ for the distribution that produces v_i with probability p_i .

Step to distributions on values

Flip $\rightarrow \{tt : 1/2, ff : 1/2\}$

Roll $\rightarrow \{1 : 1/6, \dots, 6 : 1/6\}$

Operational semantics: unit and bind

Unit

$$\frac{e \rightarrow e'}{\text{return}(e) \rightarrow \text{return}(e')}$$

Bind

$$\frac{e \rightarrow \{v_i : p_i\}_{i \in I}}{\text{sample } x = e \text{ in } e' \rightarrow \sum_{i \in I} p_i \cdot e'[v_i/x]}$$

A Simple Probabilistic Type System

Types in our language

$$\begin{array}{lcl} \mathcal{T} \ni \tau := \mathbb{B} \mid \mathbb{N} & & \text{(base types)} \\ | \quad \mathcal{T} \rightarrow \mathcal{T} & & \text{(functions)} \\ | \text{ } \textcolor{blue}{\bigcirc} \mathcal{T} & & \text{(distributions)} \end{array}$$


Typing judgment basics

The main judgment

Let $e \in \mathcal{E}$, $\tau \in \mathcal{T}$, and Γ be a finite list of of bindings
 $x_1 : \tau_1, \dots, x_n : \tau_n$. Then the typing judgment is:

$$\Gamma \vdash e : \tau$$

Reading

If we substitute closed values v_1, \dots, v_n for variables x_1, \dots, x_n in e , then the result either reduces to $\text{unit}(v)$ if τ is non-probabilistic, or reduces to a sub-distribution over closed values if τ is probabilistic (of the form $\bigcirc\tau$).

Typing rules: variables and functions

Exactly the same as in lambda calculus

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ VAR}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x. e : \tau \rightarrow \tau'} \text{ LAM}$$

$$\frac{\Gamma \vdash e : \tau \rightarrow \tau' \quad \Gamma \vdash e' : \tau}{\Gamma \vdash e \ e' : \tau'} \text{ APP}$$

$$\frac{\Gamma, f : \tau \rightarrow \tau' \vdash \lambda x. e : \tau \rightarrow \tau'}{\Gamma \vdash \text{fix } f. \lambda x. e : \tau \rightarrow \tau'} \text{ FIX}$$

Typing rules: booleans and integers

Hopefully not too surprising

$$\frac{b = tt, ff}{\Gamma \vdash b : \mathbb{B}} \text{BOOL}$$

$$\frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathbb{N}} \text{NAT}$$

$$\frac{\begin{array}{c} \Gamma \vdash e : \mathbb{N} \\ \Gamma \vdash e' : \mathbb{N} \end{array}}{\Gamma \vdash \text{add}(e, e') : \mathbb{N}} \text{ADD}$$

Typing rules: primitive distributions

Assign distribution types

$$\frac{}{\Gamma \vdash \text{Flip} : \bigcirc \mathbb{B}} \text{FLIP}$$

$$\frac{}{\Gamma \vdash \text{Roll} : \bigcirc \mathbb{N}} \text{ROLL}$$

Typing rules: unit and bind

Unit

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{return}(e) : \bigcirc \tau} \text{RETURN}$$

Bind

$$\frac{\Gamma \vdash e : \bigcirc \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc \tau'} \text{SAMPLE}$$

What property do we want the types to ensure?

Non-probabilistic types

If $e \in \mathcal{CE}$ has non-probabilistic type τ , then e should reduce to $\text{unit}(v)$ with $v \in \mathcal{CV}$ of type τ , or loop forever.

Probabilistic types

If $e \in \mathcal{CE}$ has probabilistic type $\bigcirc\tau$, then e should reduce to $\mu \in \text{SDistr}(\mathcal{CV})$ where every element in the support of μ has type τ .

Monadic Type Systems: A Closer Look

What else can we do with a monadic type system?

So far: describe type of a distribution

If a program e has type \textcircled{N} , then:

- ▶ It evaluates to a sub-distribution over \mathbb{N} : samples drawn from the distribution will always be natural numbers.
- ▶ It never gets stuck (runtime error) during evaluation.

But what other properties can we handle?

- ▶ Produces a uniform distribution
- ▶ Produces a distribution that has probability $1/4$ of returning an even number
- ▶ ...

The key typing rule: SAMPLE

$$\frac{\Gamma \vdash e : \bigcirc \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc \tau'} \text{ SAMPLE}$$

The key typing rule: SAMPLE

$$\frac{\Gamma \vdash e : \bigcirc \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc \tau'} \text{ SAMPLE}$$

Let's unpack this rule

1. e is a distribution over τ

The key typing rule: SAMPLE

$$\frac{\Gamma \vdash e : \bigcirc \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc \tau'} \text{ SAMPLE}$$

Let's unpack this rule

1. e is a distribution over τ
2. Given a sample $x : \tau$, e' produces a distribution over τ'

The key typing rule: SAMPLE

$$\frac{\Gamma \vdash e : \bigcirc \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc \tau'} \text{ SAMPLE}$$

Let's unpack this rule

1. e is a distribution over τ
2. Given a sample $x : \tau$, e' produces a distribution over τ'
3. Sampling from e and plugging into e' : distribution over τ'

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Let's change the meaning of the distribution type

1. e is a distribution over τ satisfying P

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Let's change the meaning of the distribution type

1. e is a distribution over τ satisfying P
2. Given a sample $x : \tau$, e' produces a distribution over τ' satisfying Q

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Let's change the meaning of the distribution type

1. e is a distribution over τ satisfying P
2. Given a sample $x : \tau$, e' produces a distribution over τ' satisfying Q
3. Sampling from e and plugging into e' produces a distribution over τ' satisfying Q

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Let's change the meaning of the distribution type

1. e is a distribution over τ satisfying P
2. Given a sample $x : \tau$, e' produces a distribution over τ' satisfying Q
3. Sampling from e and plugging into e' produces a distribution over τ' satisfying Q

Generalizing the rule

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

Let's change the meaning of the distribution type

1. e is a distribution over τ satisfying P
2. Given a sample $x : \tau$, e' produces a distribution over τ' satisfying Q
3. Sampling from e and plugging into e' produces a distribution over τ' satisfying Q

For what distribution properties Q is this rule OK?

Does this remind you of something we have seen already?

CM properties: Closed under Mixtures

An assertion Q is CM if it satisfies:

If $\mu_1 \models Q$ and $\mu_2 \models Q$, then $\mu_1 \oplus_p \mu_2 \models Q$ for any $p \in [0, 1]$.

Examples of CM assertions

- ▶ $x = e$
- ▶ $\mathbf{Unif}_{\mathbb{B}}[x]$

Examples of non-CM assertions

- ▶ $[x] * [y]$
- ▶ $x = 1 \vee x = 2$

The main requirement: closed under mixtures (CM)

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

The main requirement: closed under mixtures (CM)

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

The property Q must be closed under mixtures (CM)

1. We have a bunch of distributions over τ' satisfying Q

The main requirement: closed under mixtures (CM)

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

The property Q must be closed under mixtures (CM)

1. We have a bunch of distributions over τ' satisfying Q
2. We are blending these distributions together

The main requirement: closed under mixtures (CM)

$$\frac{\Gamma \vdash e : P\tau \quad \Gamma, x : \tau \vdash e' : Q\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : Q\tau'} \text{ SAMPLEGEN}$$

The property Q must be closed under mixtures (CM)

1. We have a bunch of distributions over τ' satisfying Q
2. We are blending these distributions together
3. We want the resulting distribution to also satisfy Q

Example: monadic types for uniformity

Type of uniform distributions $U\tau$

Meaning: when τ is a finite type (e.g., \mathbb{B}), a program e has type $U\tau$ if it evaluates to the uniform distribution over τ without encountering any runtime errors.

Then the sampling rule is sound:

$$\frac{\Gamma \vdash e : \bigcirc\tau \quad \Gamma, x : \tau \vdash e' : U\tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : U\tau'} \text{ SAMPLEUNIF}$$

Monadic Type Systems: Generalizing to Graded Monads

From monads to graded monads

Instead of one monad, have a family of monads

- ▶ M is a monoid with a pre-order (e.g., $(\mathbb{R}, 0, +, \leq)$)
- ▶ Each monadic type has an index $\alpha \in M$

From monads to graded monads

Instead of one monad, have a family of monads

- ▶ M is a monoid with a pre-order (e.g., $(\mathbb{R}, 0, +, \leq)$)
- ▶ Each monadic type has an index $\alpha \in M$

Intuition

- ▶ Graded monads: different kinds of the same monad
- ▶ Smaller index: less information/weaker guarantee
- ▶ Index carries additional information “on the side”
- ▶ Indexes combine through the bind rule

Changes to the type system

New types

$$\mathcal{T} \ni \tau := \dots | \bigcirc_{\alpha} \tau \quad (\alpha \in M)$$

New typing rules

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{return}(e) : \bigcirc_0 \tau} \text{ GRETURN}$$

$$\frac{\Gamma \vdash e : \bigcirc_{\alpha} \tau \quad \Gamma, x : \tau \vdash e' : \bigcirc_{\beta} \tau'}{\Gamma \vdash \text{sample } x = e \text{ in } e' : \bigcirc_{\alpha+\beta} \tau'} \text{ GSAMPLE}$$

$$\frac{\Gamma \vdash e : \bigcirc_{\alpha} \quad \alpha \leq \beta}{\Gamma \vdash e : \bigcirc_{\beta}} \text{ GSUBTY}$$

Monadic types: references and further readings

Original papers on probabilistic monadic types

- ▶ Ramsey and Pfeffer. Stochastic lambda calculus and monads of probability distributions. POPL 2002.
- ▶ Park, Pfenning, and Thrun. A Probabilistic Language based upon Sampling Functions. POPL 2005.

Differential privacy typing

- ▶ Key ingredients: (bounded) linear types and a monad
- ▶ Reed and Pierce. Distance makes the types grow stronger: a calculus for differential privacy. ICFP 2010.

HOARE²: probabilistic relational properties by typing

- ▶ Key ingredients: Refinement types and a graded monad.
- ▶ Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy. POPL 2015.

Beyond Monadic Types: Two Representative Systems

Monadic type systems: the good and the bad

The good

- ▶ Clean separation between deterministic and randomized
- ▶ Always treat variables as values, not distributions

The bad

- ▶ Class of properties is limited
- ▶ All properties everywhere must be CM (cf. PSL)

Main features

- ▶ Makes τ and $\bigcirc\tau$ the same: no more monad!
- ▶ Call-by-value: sample when passing arguments to fn.

What kinds of properties can be expressed in types?

- ▶ No monad type, but let-binding rule is similar to SAMPLE
- ▶ Seems to need the CM condition

PCF_⊕: Reading the typing judgment

Judgments look like

$$x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau$$

Reading

For any well-typed closing substitution of values v_1, \dots, v_n for x_1, \dots, x_n , the expression e evaluates to distribution over τ .

Main features

- ▶ Makes τ and $\bigcirc\tau$ the same: no more monad!
- ▶ Call-by-name: functions can take distributions
- ▶ Let-binding construct used to force sampling

What kinds of properties can be expressed in types?

- ▶ Function calls don't force sampling
- ▶ Let-binding, if-then-else, all force sampling

PPCF: Reading the typing judgment

Judgments look like

$$x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau$$

Reading

For any well-typed closing substitution of distributions μ_1, \dots, μ_n for μ_1, \dots, μ_n , the expression e evaluates to some distribution over τ .

But note that μ_1, \dots, μ_n are entirely separate distributions: draws from μ_1, \dots, μ_n are always independent.

Many technical extensions

Richer distributions

- ▶ Continuous distributions
- ▶ Distributions over function spaces

Richer types

- ▶ Recursive types, linear types, ...

Richer language features

- ▶ Most notably: conditioning constructs (“observe”/“score”)

Higher-order programs: references and readings

Semantics

- ▶ Saheb-Djahromi. CPO's of Measures for Non-determinism. 1979.
- ▶ Jones and Plotkin. A Probabilistic Powerdomain of Evaluations. 1989.
- ▶ Heunen, Kammar, Staton, Yang. A Convenient Category for Higher-Order Probability Theory. 2017.

Type systems

- ▶ PCF \oplus : Dal Lago
(<https://doi.org/10.1017/9781108770750.005>)
- ▶ PPCF: Erhard, Pagani, Tasson. Measurable Cones and Stable, Measurable Functions. 2018.
- ▶ Darais, Sweet, Liu, Hicks. A language for probabilistically oblivious computation. POPL 2020.

Reasoning about Probabilistic Programs

Wrapping up

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

Main takeaways

There are multiple semantics for probabilistic programs

- ▶ We saw: monadic semantics, and transformer semantics
- ▶ Choice of semantics influences what verification is possible

Standard verification methods, to probabilistic programs

- ▶ Weakest pre-conditions to weakest pre-expectations
- ▶ Separation logic to Probabilistic separation logic
- ▶ Type systems, monads, ...

Verification currently better for imperative programs

- ▶ Wide variety of Hoare logics proving interesting properties
- ▶ Type systems for probabilistic programs: active research

Where to go next

More semantics

- ▶ Lots of recent research on categorical semantics (e.g., QBS)

Learn about conditioning

- ▶ Mostly implementation (hard), but recently verification too

Verifying specific properties

- ▶ Expected running time, probabilistic termination, ...

Interesting applications

- ▶ Cryptography, differential privacy, machine learning, ...

Read: Foundations of Probabilistic Programming

- ▶ Open-access book, 15 chapters by leading researchers

<https://doi.org/10.1017/9781108770750>

Reasoning about Probabilistic Programs

Oregon PL Summer School 2021

Justin Hsu
UW-Madison
Cornell University