# Verifying Probabilistic Programs with Separation Logic Part 1

**Joseph Tassarotti**
New York University

Based on joint work with: Alejandro Aguirre, Philipp G. Haselwarter, Kwing Hei-Li, Markus de Medeiros, Puming Liu, Alex Bai, Simon Oddershede Gregersen, and Lars Birkedal

**Course page:** clutch-project.org/epit2026.html
**Rocq code:** github.com/logsem/clutch/tree/epit2026

# Randomness is an Essential Tool

**Cryptography**

Randomness for keys, nonces, hashes, pseudorandom functions, ...

# Randomness is an Essential Tool

## Cryptography

Randomness for keys, nonces, hashes, pseudorandom functions, ...

## Privacy

Differential privacy adds randomized noise: Laplace & Gaussian mechanisms, randomized response.

# Randomness is an Essential Tool

## Cryptography

Randomness for keys, nonces, hashes, pseudorandom functions, ...

## Privacy

Differential privacy adds randomized noise: Laplace & Gaussian mechanisms, randomized response.

## Data structures & algorithms

**Simpler and faster** algorithms: hashing, Bloom filters, sketches, skip lists, Miller–Rabin, ...

# Randomness is an Essential Tool

## Cryptography

Randomness for keys, nonces, hashes, pseudorandom functions, ...

## Privacy

Differential privacy adds randomized noise: Laplace & Gaussian mechanisms, randomized response.

## Data structures & algorithms

**Simpler and faster** algorithms: hashing, Bloom filters, sketches, skip lists, Miller–Rabin, ...

## Machine learning

Stochastic Gradient Descent, Markov Chain Monte Carlo, ...

# Randomness is an Essential Tool

## Cryptography

Randomness for keys, nonces, hashes, pseudorandom functions, ...

## Privacy

Differential privacy adds randomized noise: Laplace & Gaussian mechanisms, randomized response.

## Data structures & algorithms

**Simpler and faster** algorithms: hashing, Bloom filters, sketches, skip lists, Miller–Rabin, ...

## Machine learning

Stochastic Gradient Descent, Markov Chain Monte Carlo, ...

**Program verification techniques need to handle randomness.**

# Probabilistic Verification: Many Successes!

- **Cryptographic security:**
  - CertiCrypt / EasyCrypt: OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC
  - FCF: ElGamal, HMAC, searchable symmetric encryption
  - SSProve: state-separating proofs of KEM-DEM, PRF-based encryption

# Probabilistic Verification: Many Successes!

- **Cryptographic security:**
  - CertiCrypt / EasyCrypt: OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC
  - FCF: ElGamal, HMAC, searchable symmetric encryption
  - SSProve: state-separating proofs of KEM-DEM, PRF-based encryption
- **Differential privacy:**
  - apRHL: Laplace & Gaussian mechanisms, Sparse Vector, Report Noisy Max, Above Threshold
  - Fuzz / DFuzz / Duet: linear/dependent types for DP
  - LightDP: automated proofs (Sparse Vector, Report Noisy Max, ...)

# Probabilistic Verification: Many Successes!

- **Cryptographic security:**
  - CertiCrypt / EasyCrypt: OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC
  - FCF: ElGamal, HMAC, searchable symmetric encryption
  - SSProve: state-separating proofs of KEM-DEM, PRF-based encryption
- **Differential privacy:**
  - apRHL: Laplace & Gaussian mechanisms, Sparse Vector, Report Noisy Max, Above Threshold
  - Fuzz / DFuzz / Duet: linear/dependent types for DP
  - LightDP: automated proofs (Sparse Vector, Report Noisy Max, ...)
- **Randomized algorithms:**
  - Union-bound logic (aHL): upper bound error probabilities
  - Ellora: polynomial identity testing, random walks, concentration bounds
  - Caesar / HeyVL: weakest pre-expectation reasoning for pGCL

# Limitations... and A Divide

## Probabilistic Verification

- small programs
- restricted languages
- often not compositional

# Limitations... and A Divide

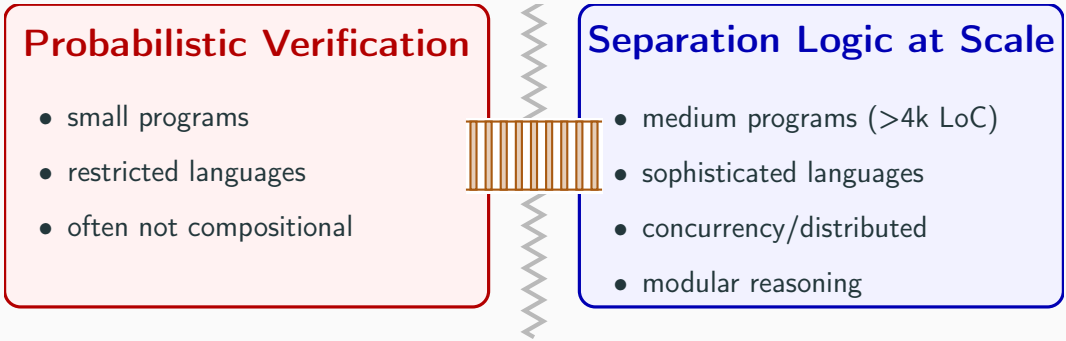
## Probabilistic Verification

- small programs
- restricted languages
- often not compositional

## Separation Logic at Scale

- medium programs (>4k LoC)
- sophisticated languages
- concurrency/distributed
- modular reasoning

# Limitations... and A Divide




**Can we bridge these two worlds?**

# Clutch Project: Probabilistic Reasoning in Separation Logic

**Approach:** Extend higher-order separation logic with reasoning principles from probabilistic logics.

- **Relational Reasoning**
- **Differential Privacy**
- **Expected Runtime**
- **Error Bounding**

All results mechanized in Rocq proof assistant.

# Clutch Project: Probabilistic Reasoning in Separation Logic

**Approach:** Extend higher-order separation logic with reasoning principles from probabilistic logics.

- Relational Reasoning
- Differential Privacy
- Expected Runtime
- **Error Bounding**

All results mechanized in Rocq proof assistant.

# Clutch Project: Probabilistic Reasoning in Separation Logic

**Approach:** Extend higher-order separation logic with reasoning principles from probabilistic logics.

- Relational Reasoning
- Differential Privacy
- Expected Runtime
- **Error Bounding**

All results mechanized in Rocq proof assistant.

**Note:** There is a **radically different** approach in other work, will discuss later.

# Background:
# Modern Separation Logic

# The Iris Framework

**Iris** [POPL15, JFP18] is a higher-order concurrent separation logic framework, mechanized in Rocq. Used for many projects:

- **RustBelt**: safety of the Rust type system and unsafe libraries
- **Perennial**: crash-safe concurrent storage systems
- **ReLoC**: contextual refinement of higher-order concurrent programs
- **Actris**: session-typed message-passing concurrency
- **Aneris**, **Grove**: distributed systems
- ...

The logics I will describe all build on this framework and retain its expressive features.

$$\{P\}\ e\ \{Q\}$$

If the program state initially satisfies $P$, then running $e$ from this state is safe, and after running $e$, the resulting state satisfies $Q$.

- $P$ is the precondition.
- $Q$ is the postcondition.
- $e$ is the program being verified.

# Separation Logic

Separation Logic adds "separating conjunction" connective $*$ to Hoare logic:

$$P * Q$$

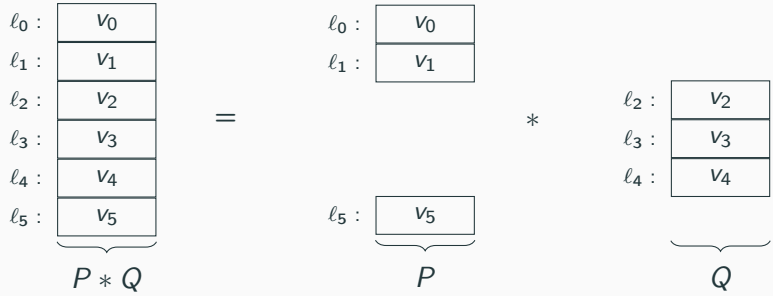
state can be split into disjoint pieces satisfying $P$ and $Q$ respectively

# Separation Logic

Separation Logic adds "separating conjunction" connective $*$ to Hoare logic:

$P * Q$

state can be split into disjoint pieces satisfying $P$ and $Q$ respectively



# Points-to Assertion and Rules

The points-to assertion $\ell \mapsto v$ says that location $\ell$ stores value $v$, and confers ownership of $\ell$.

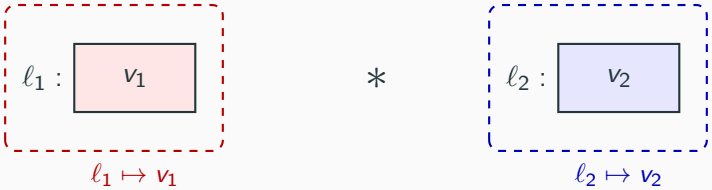
To read or write $\ell$, the precondition must contain $\ell \mapsto -$.

- **Allocation:** $\{\mathsf{True}\}\ \mathsf{ref}\ v\ \{\ell.\, \ell \mapsto v\}$
- **Read:** $\{\ell \mapsto v\}\ !\ell\ \{x.\, x = v * \ell \mapsto v\}$
- **Write:** $\{\ell \mapsto v\}\ \ell := w\ \{\ell \mapsto w\}$

# Disjointness from Points-to

Since $*$ requires splitting into disjoint heap pieces:

$$\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \neq \ell_2$$




A single cell cannot belong to both pieces, so $\ell_1$ and $\ell_2$ must differ.

# A Substructural Logic

This disjointness also makes the logic **substructural**.

# A Substructural Logic

This disjointness also makes the logic **substructural**.

In classical logic, hypotheses can always be copied:

$$P \vdash P \wedge P$$

# A Substructural Logic

This disjointness also makes the logic **substructural**.

In classical logic, hypotheses can always be copied:

$$P \vdash P \land P$$

In separation logic, ownership cannot be duplicated:

$$\ell \mapsto v \not\vdash \ell \mapsto v * \ell \mapsto v$$

(Otherwise we could "own" the same location twice.)

# Affine Separation Logic

Iris is an **affine** separation logic:

$$P * Q \vdash P$$

# Affine Separation Logic

Iris is an **affine** separation logic:

$$P * Q \vdash P$$

Early separation logics were non-affine. Goal was to prove absence of memory-leaks.

# The Persistent Modality

Some assertions do not describe exclusive ownership — they can be freely duplicated.

# The Persistent Modality

Some assertions do not describe exclusive ownership — they can be freely duplicated.

Iris uses the persistent modality $\square P$ to mark such propositions:

$$\square P \vdash \square P * \square P \qquad \square P \vdash P$$

# The Persistent Modality

Some assertions do not describe exclusive ownership — they can be freely duplicated.

Iris uses the persistent modality $\square P$ to mark such propositions:

$$\square P \vdash \square P * \square P \qquad \square P \vdash P$$

Examples of persistent assertions:

- Pure mathematical facts (e.g., $n > 0$)
- Hoare triples $\{P\}\, e\, \{Q\}$ — specifications can be reused freely

# The Frame Rule

Because modifying a location requires ownership, a program cannot modify outside of its precondition.

So specifications can be extended with any disjoint piece of state $R$:

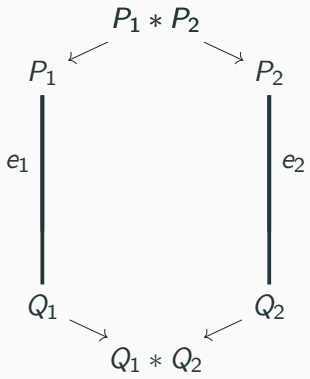
$$\frac{\{P\}\ e\ \{Q\}}{\{P * R\}\ e\ \{Q * R\}}$$

Premise implies $e$ touches only state described by $P$, so the frame $R$ is unchanged.

# Concurrent Composition through Ownership

Separation enables thread-local reasoning about **concurrent composition**:

$$\frac{\{P_1\}\ e_1\ \{Q_1\} \qquad \{P_2\}\ e_2\ \{Q_2\}}{\{P_1 * P_2\}\ (e_1\ \|\ e_2)\ \{Q_1 * Q_2\}}$$



# The Magic Wand

The magic wand $P \mathrel{-\!\!*} Q$ (separating implication) intuitively says: "given a disjoint resource satisfying $P$, the combined resource satisfies $Q$".

# The Magic Wand

The magic wand $P \mathrel{-\!\!*} Q$ (separating implication) intuitively says: "given a disjoint resource satisfying $P$, the combined resource satisfies $Q$".

Modus ponens:

$$P * (P \mathrel{-\!\!*} Q) \vdash Q$$

# The Magic Wand

The magic wand $P \mathrel{-\!\!*} Q$ (separating implication) intuitively says: "given a disjoint resource satisfying $P$, the combined resource satisfies $Q$".

Modus ponens:

$$P * (P \mathrel{-\!\!*} Q) \vdash Q$$

Adjoint to $*$:

$$P * Q \vdash R \quad \iff \quad Q \vdash P \mathrel{-\!\!*} R$$

# Hoare Triples via Weakest Preconditions

In Iris, the Hoare triple is a derived notion. The primitive is the weakest precondition modality $\mathsf{wp}\ e\ \{\Phi\}$.

# Hoare Triples via Weakest Preconditions

In Iris, the Hoare triple is a derived notion. The primitive is the weakest precondition modality $\mathsf{wp}\ e\ \{\Phi\}$.

Intuitively:

$$\mathsf{wp}\ e\ \{\Phi\} \quad \equiv \quad \text{“running } e \text{ is safe and any result } v \text{ satisfies } \Phi(v)\text{”}$$

# Hoare Triples via Weakest Preconditions

In Iris, the Hoare triple is a derived notion. The primitive is the weakest precondition modality $\mathsf{wp}\, e\, \{\Phi\}$.

Intuitively:

$$\mathsf{wp}\, e\, \{\Phi\} \quad \equiv \quad \text{“running } e \text{ is safe and any result } v \text{ satisfies } \Phi(v)\text{”}$$

Hoare triples are then defined by:

$$\{P\}\, e\, \{\Phi\} \quad \triangleq \quad \square\big(P \mathrel{-\!\!*} \mathsf{wp}\, e\, \{\Phi\}\big)$$

# Hoare Triples via Weakest Preconditions

In Iris, the Hoare triple is a derived notion. The primitive is the weakest precondition modality $\mathsf{wp}\, e\, \{\Phi\}$.

Intuitively:

$$\mathsf{wp}\, e\, \{\Phi\} \quad \equiv \quad \text{“running } e \text{ is safe and any result } v \text{ satisfies } \Phi(v)\text{”}$$

Hoare triples are then defined by:

$$\{P\}\, e\, \{\Phi\} \quad \triangleq \quad \square\big(P \mathrel{-\!\!*} \mathsf{wp}\, e\, \{\Phi\}\big)$$

# Why Reason with wp $e\,\{\Phi\}$ Directly?

Proofs are carried out at the level of wp $e\,\{\Phi\}$ rather than triples.

# Why Reason with wp $e\,\{\Phi\}$ Directly?

Proofs are carried out at the level of wp $e\,\{\Phi\}$ rather than triples.

- **More primitive:** theoretically cleaner to work with more "basic" notion.
- **Mechanization:** wp $e\,\{\Phi\}$ goal is easier to manipulate with tactics in Rocq.
- **Cleaner rules:** expressions decompose cleanly through evaluation contexts $K[\cdot]$:

$$\text{wp } K[e]\,\{\Phi\} \;\dashv\vdash\; \text{wp } e\,\{v.\,\text{wp } K[v]\,\{\Phi\}\}$$

# The Later Modality and Löb Induction

Iris features a later modality $\triangleright P$, intuitively meaning "$P$ holds one step later".

# The Later Modality and Löb Induction

Iris features a later modality $\triangleright P$, intuitively meaning "$P$ holds one step later".

Some basic rules:

$$P \vdash \triangleright P \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \triangleright \Box P \dashv\vdash \Box \triangleright P$$

# The Later Modality and Löb Induction

Iris features a later modality $\triangleright P$, intuitively meaning "$P$ holds one step later".

Some basic rules:

$$P \vdash \triangleright P \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \triangleright \Box P \dashv\vdash \Box \triangleright P$$

Löb induction:

$$(\triangleright P \mathrel{-\!\!*} P) \vdash P$$

If we can prove $P$ assuming $P$ holds *later*, then $P$ holds.

# The Later Modality and Löb Induction

Iris features a later modality $\triangleright P$, intuitively meaning "$P$ holds one step later".

Some basic rules:

$$P \vdash \triangleright P \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \triangleright \Box P \dashv\vdash \Box \triangleright P$$

Löb induction:

$$(\triangleright P \mathrel{-\!\!*} P) \vdash P$$

If we can prove $P$ assuming $P$ holds *later*, then $P$ holds.

Used to reason about recursive programs: each program step "discharges" a $\triangleright$, allowing us to use the IH.


# Rocq Exercises

# Probabilistic Preliminaries

# Probability distributions

A (sub)-distribution over a countable type $A$ is a map $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$.

# Probability distributions

A (sub)-distribution over a countable type $A$ is a map $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$.
We will sometimes use the notation:

$$\{a_1 \mapsto p_1,\ a_2 \mapsto p_2,\ \dots\} \qquad\qquad a_i \in A, p_i = \mu(a_i)$$

# Probability distributions

A (sub)-distribution over a countable type $A$ is a map $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$.
We will sometimes use the notation:

$$\{a_1 \mapsto p_1,\ a_2 \mapsto p_2,\ \dots\} \qquad\qquad a_i \in A, p_i = \mu(a_i)$$

Example: Outcomes of a die roll, $A = \{1, 2, 3, 4, 5, 6\}$ are described by

$$\{1 \mapsto \frac{1}{6},\ 2 \mapsto \frac{1}{6},\ \dots,\ 6 \mapsto \frac{1}{6}\}$$

# Probability distributions

Distributions have a convex combination operation. Suppose we have:

- Countable sets $I, A$
- A set of weights $\{p_i\}_{i \in I}$ s.t. $\sum_{i \in I} p_i \leq 1$
- For each $i \in I$, a distribution $\nu_i$ over $A$

# Probability distributions

Distributions have a convex combination operation. Suppose we have:

- Countable sets $I, A$
- A set of weights $\{p_i\}_{i \in I}$ s.t. $\sum_{i \in I} p_i \leq 1$
- For each $i \in I$, a distribution $\nu_i$ over $A$

Then we can combine them into a distribution $(\bigoplus_i p_i \cdot \nu_i) \in \mathcal{D}(A)$:

$$(\bigoplus_i p_i \cdot \nu_i)(a) \triangleq \sum_{i \in I} p_i \cdot \nu_i(a)$$

# Probability distributions

Distributions have a convex combination operation. Suppose we have:

- Countable sets $I, A$
- A set of weights $\{p_i\}_{i \in I}$ s.t. $\sum_{i \in I} p_i \leq 1$
- For each $i \in I$, a distribution $\nu_i$ over $A$

Then we can combine them into a distribution $(\bigoplus_i p_i \cdot \nu_i) \in \mathcal{D}(A)$:

$$(\bigoplus_i p_i \cdot \nu_i)(a) \triangleq \sum_{i \in I} p_i \cdot \nu_i(a)$$

Example: Suppose we flip a coin, if it's heads we roll 1D6, otherwise we roll 1D4

$$\nu_1 = \{1 \mapsto \tfrac{1}{6}, 2 \mapsto \tfrac{1}{6}, 3 \mapsto \tfrac{1}{6}, 4 \mapsto \tfrac{1}{6}, 5 \mapsto \tfrac{1}{6}, 6 \mapsto \tfrac{1}{6}\}$$

$$\nu_2 = \{1 \mapsto \tfrac{1}{4}, 2 \mapsto \tfrac{1}{4}, 3 \mapsto \tfrac{1}{4}, 4 \mapsto \tfrac{1}{4}\}$$

$$(1/2) \cdot \nu_1 \oplus (1/2) \cdot \nu_2 = \{1 \mapsto \tfrac{5}{24}, 2 \mapsto \tfrac{5}{24}, 3 \mapsto \tfrac{5}{24}, 4 \mapsto \tfrac{5}{24}, 5 \mapsto \tfrac{2}{24}, 6 \mapsto \tfrac{2}{24}\}$$

# The $\mathsf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ language

An ML-like language with higher-order (recursive) functions, higher-order state, impredicative polymorphism, ..., and probabilistic uniform sampling.

$$e \in \mathit{Expr} ::= \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \mathsf{fst}(e) \mid \mathsf{snd}(e) \mid \mathsf{ref}(e) \mid\ !\, e \mid e_1 \leftarrow e_2 \mid \lambda x.e \mid e_1\ e_2 \mid \ldots \mid \mathsf{rand}(e)$$

$\mathsf{rand}\, N$ evaluates to a uniform sample from $\{0, 1, \ldots, N\}$

# Operational semantics

We start from a probabilistic head step reduction hdStep: $Cfg \to \mathcal{D}(Cfg)$:

$$(\lambda x.e)\ v, \sigma \to^1_{\mathsf{h}} e[v/x], \sigma$$

$$\text{if true then } e_1 \text{ else } e_2, \sigma \to^1_{\mathsf{h}} e_1, \sigma$$

$$\text{if false then } e_1 \text{ else } e_2, \sigma \to^1_{\mathsf{h}} e_2, \sigma$$

$$!\ l, \sigma \to^1_{\mathsf{h}} \sigma(l), \sigma \qquad l \in \mathrm{dom}(\sigma)$$

$$\dots$$

$$\mathsf{flip}, \sigma \to^{1/2}_{\mathsf{h}} b, \sigma \qquad b \in \{\mathsf{true}, \mathsf{false}\}$$

$$\mathsf{rand}\ N, \sigma \to^{1/(N+1)}_{\mathsf{h}} z, \sigma \qquad z \in \{0, \dots, N\}, 0 \leq N$$

# Operational semantics

We start from a probabilistic head step reduction hdStep : $Cfg \to \mathcal{D}(Cfg)$:

$$
\begin{aligned}
(\lambda x.e)\ v, \sigma &\to^1_{\mathsf{h}} e[v/x], \sigma \\
\text{if true then } e_1 \text{ else } e_2, \sigma &\to^1_{\mathsf{h}} e_1, \sigma \\
\text{if false then } e_1 \text{ else } e_2, \sigma &\to^1_{\mathsf{h}} e_2, \sigma \\
!\ l, \sigma &\to^1_{\mathsf{h}} \sigma(l), \sigma \qquad l \in \mathrm{dom}(\sigma) \\
&\dots \\
\mathsf{flip}, \sigma &\to^{1/2}_{\mathsf{h}} b, \sigma \qquad b \in \{\mathsf{true}, \mathsf{false}\} \\
\mathsf{rand}\ N, \sigma &\to^{1/(N+1)}_{\mathsf{h}} z, \sigma \qquad z \in \{0, \dots, N\}, 0 \leq N
\end{aligned}
$$

and lift it to reduction in context step : $Cfg \to \mathcal{D}(Cfg)$:

$$
\frac{(e, \sigma) \to^p_{\mathsf{h}} (e', \sigma')}{(K[e], \sigma) \to^p (K[e'], \sigma')}
$$

# Probabilistic evaluation

We define a big-step evaluation, where $(e, \sigma) \Downarrow_n \mu$ states that after running $(e, \sigma)$ for $n$ steps, the output configurations distribute according to $\mu$

$$\frac{v \in \mathit{Val}}{(v, \sigma) \Downarrow_n \{(v, \sigma) \mapsto 1\}} \qquad \frac{e \notin \mathit{Val}}{(e, \sigma) \Downarrow_0 0} \text{where } 0 \triangleq (\lambda\_.0)$$

$$\frac{e \notin \mathit{Val} \qquad (e, \sigma) \to \{(e_i, \sigma_i) \mapsto p_i\}_{i \in I} \qquad \forall i \in I, (e_i, \sigma_i) \Downarrow_n \mu_i}{(e, \sigma) \Downarrow_{n+1} \bigoplus_i p_i \cdot \mu_i}$$

# Probabilistic evaluation

We can then take limits:

$$\frac{\forall i \in \mathbb{N}, (e, \sigma) \Downarrow_i \mu_i}{(e, \sigma) \Downarrow (\lambda \rho.\ \lim_{i \to \infty} \mu_i(\rho))}$$

This defines an evaluation function $\Downarrow : \mathit{Cfg} \to \mathcal{D}(\mathit{Cfg})$ mapping an initial configuration to a distribution over final configurations.

Exercise: convince yourself that this is well-defined

# Example: Randomized sum

Consider $f \triangleq \text{rec}\, f\, n = \text{if}\ n = 0\ \text{then}\ 0\ \text{else if}\ \text{flip}\ \text{then}\ n + f(n-1)\ \text{else}\ f(n-1)$ One possible execution trace of $f(2)$ is:

$$
\begin{aligned}
(f(2), []) &\to^1 (\text{if}\ 2 = 0\ \text{then}\ 0\ \text{else if}\ \text{flip}\ \text{then}\ 2 + f(2-1)\ \text{else}\ f(2-1), []) \\
&\to^1 (\text{if false then}\ 0\ \text{else if}\ \text{flip}\ \text{then}\ 2 + f(2-1)\ \text{else}\ f(2-1), []) \\
&\to^1 (\text{if}\ \text{flip}\ \text{then}\ 2 + f(2-1)\ \text{else}\ f(2-1), []) \\
&\to^{1/2} (\text{if true then}\ 2 + f(2-1)\ \text{else}\ f(2-1), []) \\
&\to^1 (2 + f(2-1), []) \\
&\dots \\
&\to^1 (2 + \text{if}\ \text{flip}\ \text{then}\ 1 + f(1-1)\ \text{else}\ f(1-1), []) \\
&\to^{1/2} (2 + \text{if false then}\ 1 + f(1-1)\ \text{else}\ f(1-1), []) \\
&\to^1 (2 + f(1-1), []) \\
&\dots \\
&\to^1 (2 + 0, []) \to^1 (2, [])
\end{aligned}
$$

# Example: Randomized sum

$f \triangleq \mathsf{rec}\, f\ n = \mathsf{if}\ n = 0\ \mathsf{then}\ 0\ \mathsf{else\ if}\ \mathsf{flip}\ \mathsf{then}\ n + f(n-1)\ \mathsf{else}\ f(n-1)$

The final distribution produced by $f(2)$ is:

$$(f(2), []) \Downarrow \{\, (0, []) \mapsto 1/4,\ (1, []) \mapsto 1/4,\ (2, []) \mapsto 1/4,\ (3, []) \mapsto 1/4 \,\}$$

# Example: Geometric distribution

Consider $g \triangleq \mathsf{rec}\, g\, n = \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ n\ \mathsf{else}\ g(n+1)$

One possible execution trace of $g(0)$ is:

$$
\begin{aligned}
g(0) &\to^{1} \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 0\ \mathsf{else}\ g(0+1) \\
&\to^{1/2} \mathsf{if}\ \mathsf{false}\ \mathsf{then}\ 0\ \mathsf{else}\ g(0+1) \\
&\to^{1} g(0+1) \\
&\to^{1} g(1) \\
&\to^{1} \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 1\ \mathsf{else}\ g(1+1) \\
&\to^{1/2} \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ 1\ \mathsf{else}\ g(1+1) \to^{1} 1
\end{aligned}
$$

This trace happens with probability $(1/2) \cdot (1/2) = 1/4$.

# Example: Geometric distribution

$$g \triangleq \mathsf{rec}\, g\, n = \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ n\ \mathsf{else}\ g(n+1)$$

# Example: Geometric distribution

$$g \triangleq \mathsf{rec}\, g\, n = \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ n\ \mathsf{else}\ g(n+1)$$

After n unrollings of $g(0)$, we get a strict subdistribution:

$$\{(0,[]) \mapsto \frac{1}{2},\ (1,[]) \mapsto \frac{1}{4},\ (2,[]) \mapsto \frac{1}{8},\ \ldots, (n-1,[]) \mapsto \frac{1}{2^n}\ \}$$

# Example: Geometric distribution

$$g \triangleq \mathsf{rec}\, g\, n = \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ n\ \mathsf{else}\ g(n+1)$$

After n unrollings of $g(0)$, we get a strict subdistribution:

$$\{(0,[]) \mapsto \frac{1}{2},\ (1,[]) \mapsto \frac{1}{4},\ (2,[]) \mapsto \frac{1}{8},\ \ldots, (n-1,[]) \mapsto \frac{1}{2^n}\ \}$$

By taking limits, we get a full distribution

$$(g(0),[]) \Downarrow \{(0,[]) \mapsto \frac{1}{2},\ (1,[]) \mapsto \frac{1}{4},\ (2,[]) \mapsto \frac{1}{8},\ \ldots, (n-1,[]) \mapsto \frac{1}{2^n},\ \ldots\}$$

# Extending Hoare Triples to Probabilistic Programs

How should a Hoare triple $\{P\}\, e\, \{Q\}$ account for probabilistic execution?

Standard reading:

✗ "If $e$ starts in a state satisfying $P$ and terminates, the final state satisfies $Q$."

# Extending Hoare Triples to Probabilistic Programs

How should a Hoare triple $\{P\}\, e\, \{Q\}$ account for probabilistic execution?

Standard reading:

"If $e$ starts in a state satisfying $P$ and terminates, the final state satisfies $Q$."

Now $e$ evaluates to a distribution over final states, not a single state.

# Extending Hoare Triples to Probabilistic Programs

How should a Hoare triple $\{P\}\, e\, \{Q\}$ account for probabilistic execution?

Standard reading:

✗ "If $e$ starts in a state satisfying $P$ and terminates, the final state satisfies $Q$."

Now $e$ evaluates to a distribution over final states, not a single state.

Three main approaches in the literature.

# Approach 1: Distributional Assertion Program Logics

Recall: in basic Hoare logic, assertions are predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

# Approach 1: Distributional Assertion Program Logics

Recall: in basic Hoare logic, assertions are predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

In the distributional assertion approach, assertions range over distributions of states:

$$P : \mathcal{D}(\mathit{State}) \to \mathit{Prop}$$

# Approach 1: Distributional Assertion Program Logics

Recall: in basic Hoare logic, assertions are predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

In the distributional assertion approach, assertions range over distributions of states:

$$P : \mathcal{D}(\mathit{State}) \to \mathit{Prop}$$

Updated reading of $\{P\}\, e\, \{Q\}$:

"If $e$ starts from an initial distribution satisfying $P$,
then the distribution of final states satisfies $Q$."

# Approach 1: Distributional Assertion Program Logics

Recall: in basic Hoare logic, assertions are predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

In the distributional assertion approach, assertions range over distributions of states:

$$P : \mathcal{D}(\mathit{State}) \to \mathit{Prop}$$

Updated reading of $\{P\}\, e\, \{Q\}$:

> "If $e$ starts from an initial distribution satisfying $P$,
> then the distribution of final states satisfies $Q$."

Interpret $P * Q$ as probabilistic independence

*Examples: PSL, Lilac, Outcome Logic, Bluebell, ...*

# Approach 1: Pros and Cons

**Pros:**

- Directly express distributional invariants (e.g., "$\ell$ stores a uniformly distributed boolean")
- Extends to natural probabilistic notions (e.g. conditioning)

# Approach 1: Pros and Cons

**Pros:**

- Directly express distributional invariants (e.g., "$\ell$ stores a uniformly distributed boolean")
- Extends to natural probabilistic notions (e.g. conditioning)

**Cons:**

- Harder to extend with advanced features: concurrency, higher-order state, unknown adversarial code
- Some standard separation logic principles (frame, bind) require careful reworking

# Approach 2: Quantitative/Expectation Transformers

Predicates are **real-valued** functions over states: $P : \mathit{State} \to \mathbb{R}$.

# Approach 2: Quantitative/Expectation Transformers

Predicates are **real-valued** functions over states: $P : \mathit{State} \to \mathbb{R}$.

Define weakest precondition $\mathrm{wp}\, e\, \{P\} : \mathit{State} \to \mathbb{R}$ as **expected value** of $P$ under $e$

# Approach 2: Quantitative/Expectation Transformers

Predicates are **real-valued** functions over states: $P : \mathit{State} \to \mathbb{R}$.

Define weakest precondition $\mathrm{wp}\, e\, \{P\} : \mathit{State} \to \mathbb{R}$ as **expected value** of $P$ under $e$

**Pros:**

- Syntax directed "calculational" rules for computing expected value.

# Approach 2: Quantitative/Expectation Transformers

Predicates are **real-valued** functions over states: $P : \mathit{State} \to \mathbb{R}$.

Define weakest precondition wp $e\,\{P\} : \mathit{State} \to \mathbb{R}$ as **expected value** of $P$ under $e$

**Pros:**

- Syntax directed "calculational" rules for computing expected value.

**Cons:** Same as with Approach 1.

# Approach 3: Lifting-based Program Logics

In lifting-based approach, assertions stay predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

# Approach 3: Lifting-based Program Logics

In lifting-based approach, assertions stay predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

Instead, the meaning of the Hoare triple $\{P\}\, e\, \{Q\}$ is changed: the postcondition $Q$ is *lifted* to the distribution that $e$ evaluates to.

# Approach 3: Lifting-based Program Logics

In lifting-based approach, assertions stay predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

Instead, the meaning of the Hoare triple $\{P\}\, e\, \{Q\}$ is changed: the postcondition $Q$ is *lifted* to the distribution that $e$ evaluates to.

**Pros:** Easier to incorporate advanced separation logic features.

**Cons:** Typically only describe "one" probabilistic property at a time (expected running time, bound error events, etc.)

# Approach 3: Lifting-based Program Logics

In lifting-based approach, assertions stay predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

Instead, the meaning of the Hoare triple $\{P\}\, e\, \{Q\}$ is changed: the postcondition $Q$ is *lifted* to the distribution that $e$ evaluates to.

**Pros:** Easier to incorporate advanced separation logic features.

**Cons:** Typically only describe "one" probabilistic property at a time (expected running time, bound error events, etc.)

# Approach 3: Lifting-based Program Logics

In lifting-based approach, assertions stay predicates over states, $P : \mathit{State} \to \mathit{Prop}$.

Instead, the meaning of the Hoare triple $\{P\}\, e\, \{Q\}$ is changed: the postcondition $Q$ is *lifted* to the distribution that $e$ evaluates to.

**Pros:** Easier to incorporate advanced separation logic features.

**Cons:** Typically only describe "one" probabilistic property at a time (expected running time, bound error events, etc.)

Clutch family of logics follow the lifting-based approach.

# Eris: Error Credits for "Up-to-Bad" Reasoning

# Motivating Error Bound Reasoning

$$
\begin{aligned}
&\mathit{collide} \triangleq \\
&\quad \lambda\_. \\
&\qquad \mathsf{let}\ x = \mathsf{rand}(2^{64} - 1)\ \mathsf{in} \\
&\qquad \mathsf{let}\ y = \mathsf{rand}(2^{64} - 1)\ \mathsf{in} \\
&\qquad x = y
\end{aligned}
$$

Probability *collide* () returns true is $2^{-64}$

# (Almost) Specifications

Can we have a logic capturing that *collide* nearly always returns false?

$$\{\mathsf{True}\}\ \mathit{collide}\ ()\ \{x.\, x = \mathsf{false}\}_{\approx}$$

meaning the postcondition is almost always true?

# Barthe et al.'s Approximate Hoare Logic (aHL)

Annotate triples with a non-negative real number $\epsilon$:

$$\{P\}\, e\, \{Q\}_\epsilon$$

meaning postcondition will hold with probability *at least* $1 - \epsilon$.

Set $\epsilon = 0$ for usual Hoare triple.

# Barthe et al.'s Approximate Hoare Logic (aHL)

Annotate triples with a non-negative real number $\epsilon$:

$$\{P\}\, e\, \{Q\}_{\epsilon}$$

meaning postcondition will hold with probability *at least* $1 - \epsilon$.

Set $\epsilon = 0$ for usual Hoare triple.

Our example would be:

$$\{\mathsf{True}\}\ \mathit{collide}\ ()\ \{x.\, x = \mathsf{false}\}_{2^{-64}}$$

# Basic aHL Rules (adapted)

$$\frac{\{P\}\ e_1\ \{Q\}_{\epsilon_1} \qquad \{Q\}\ e_2\ \{R\}_{\epsilon_2}}{\{P\}\ e_1; e_2\ \{R\}_{\epsilon_1+\epsilon_2}} \qquad \frac{\{P\}\ e\ \{Q\}_{\epsilon_1} \qquad \epsilon_1 \leq \epsilon_2}{\{P\}\ e\ \{Q\}_{\epsilon_2}}$$

$$\frac{\Pr_{x \sim \mathsf{Unif}(N)}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{rand}(N)\ \{x.\, x \in S\}_{\epsilon}}$$

# Limitation 1: No error dependency

$$\frac{\{P\}\ e_1\ \{Q\}_{\epsilon_1} \qquad \{Q\}\ e_2\ \{R\}_{\epsilon_2}}{\{P\}\ e_1; e_2\ \{R\}_{\epsilon_1 + \epsilon_2}}$$

What if the error for $e_2$ depends on what $e_1$ did:

# Limitation 1: No error dependency

$$\frac{\{P\}\ e_1\ \{Q\}_{\epsilon_1} \qquad \{Q\}\ e_2\ \{R\}_{\epsilon_2}}{\{P\}\ e_1; e_2\ \{R\}_{\epsilon_1+\epsilon_2}}$$

What if the error for $e_2$ depends on what $e_1$ did:

$$\begin{array}{l} \mathit{collideAlt} \triangleq \\ \quad \lambda\_. \\ \quad\quad \mathsf{let}\ \mathit{len} = \mathsf{rand}(128)\ \mathsf{in} \\ \quad\quad \mathsf{let}\ x = \mathsf{rand}(2^{\mathit{len}} - 1)\ \mathsf{in} \\ \quad\quad \mathsf{let}\ y = \mathsf{rand}(2^{\mathit{len}} - 1)\ \mathsf{in} \\ \quad\quad x = y \end{array}$$

# Limitation 2: Errors propagate everywhere

Consider a standard specification for list *iter*:

$$\frac{\forall x.\, \{P(x)\}\ f\ x\ \{Q(x)\}}{\left\{ \mathop{\Large *}_{x \in l} P(x) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}}$$

# Limitation 2: Errors propagate everywhere

Consider a standard specification for list *iter*:

$$\frac{\forall x.\, \{P(x)\}\ f\ x\ \{Q(x)\}}{\left\{ \mathop{\Large *}_{x \in l} P(x) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}}$$

Now consider an "error"-aware version:

$$\frac{\forall x.\, \{P(x)\}\ f\ x\ \{Q(x)\}_{\epsilon(x)}}{\left\{ \mathop{\Large *}_{x \in l} P(x) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}_{\sum_{x \in l} \epsilon(x)}}$$

Can't be derived from the first spec, need to go back to the code.

# Limitation 3: Errors expose internals

Consider Random-Oracle model of a hash function:

$$
\begin{array}{l}
\mathit{genhash} \triangleq \\
\quad \lambda\_.\ \mathsf{let}\ m = \mathit{init\_map}\ ()\ \mathsf{in} \\
\qquad (\lambda k.\ \mathsf{match}\ \mathit{get}\ \ m\ \ k\ \mathsf{with} \\
\qquad\quad \mathsf{Some}(x) \Rightarrow x \\
\qquad\quad \mid \mathsf{None} \quad\ \Rightarrow \mathsf{let}\ x = \mathsf{rand256}()\ \mathsf{in} \\
\qquad\qquad\qquad\quad \mathit{set}\ \ m\ \ x; x \\
\qquad\quad \mathsf{end})
\end{array}
$$

Collisions are unlikely if few hashes are made.

# Limitation 3: Errors expose internals/complexity

Want a specification like:

$$\{hash(f, m) * k \notin \mathrm{dom}(m)\}$$
$$f\ \ k$$
$$\{v.\, hash(f, m[k := v]) * v \notin \mathrm{cod}(m)\}_{\epsilon}$$

But what is $\epsilon$?

# Limitation 3: Errors expose internals/complexity

Want a specification like:

$$\{hash(f, m) * k \notin \mathrm{dom}(m)\}$$
$$\quad f\ \ k$$
$$\{v.\, hash(f, m[k := v]) * v \notin \mathrm{cod}(m)\}_{\epsilon}$$

But what is $\epsilon$?

| | |
|---|---|
| First insertion: | $0$ |
| Second insertion: | $1/2^{256}$ |
| Third insertion: | $2/2^{256}$ |
| $\vdots$ | $\vdots$ |

# Drawing Inspiration from Atkey's Time Credits

We want a better way to track the error.

Inspiration: Atkey proposed separation logic with time credit assertions $\$n$ to reason about running time:

$$\{\$n\}\ \mathsf{tick}()\ \{\$(n-1)\} \qquad\qquad \$(n_1 + n_2) \dashv\vdash \$n_1 * \$n_2$$

# Drawing Inspiration from Atkey's Time Credits

We want a better way to track the error.

Inspiration: Atkey proposed separation logic with time credit assertions $\$n$ to reason about running time:

$$\{\$n\}\ \mathsf{tick}()\ \{\$(n-1)\} \qquad\qquad \$(n_1 + n_2) \dashv\vdash \$n_1 * \$n_2$$

The specification

$$\{\$n\}\ e\ \{Q\}$$

implies $e$ does at most $n$ calls to tick().

# Atkey's Motivation: Amortized Reasoning

Operations in some data structures do not have constant costs:

$$1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 6, \ldots$$

Mixture of $O(1)$ and occasionally $O(n)$. But "average" is $O(1)$.

# Atkey's Motivation: Amortized Reasoning

Operations in some data structures do not have constant costs:

$$1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 6, \ldots$$

Mixture of $O(1)$ and occasionally $O(n)$. But "average" is $O(1)$.

With amortized point of view:

$$2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, \ldots$$

Intuition: pay "extra" per operation, to pay for costly $O(n)$ operations.

# Time Credits support amortized reasoning

Consider a dynamic, resizing array:

$$\{\$3 * \mathit{dynarray}(a, \mathit{ls})\}$$
$$\mathit{arrayInsert}\ a\ v$$
$$\{\mathit{dynarray}(a, \mathit{ls} +\!\!+ [v])\}$$

**Idea:** "bank" excess credits in $\mathit{dynarray}(a, \mathit{ls})$ predicate in postcondition.

- **Non-resizing**: spend \$1, move \$2 to $\mathit{dynarray}(a, \mathit{ls})$
- **Resizing**: use banked $\$|\mathit{ls}|$ to copy

# Eris: introduce error credits for approximate bounds

Error credit assertion $↯(\epsilon)$ represents "permission" to incur $\epsilon$ approximation error.

Then the triple

$$\{↯(\epsilon) * P\}\ e\ \{Q\}$$

is analogous to

$$\{P\}\ e\ \{Q\}_\epsilon$$

# Error Credit Rules: Spending

$$\frac{\Pr_{x \sim \mathsf{Unif}(N)}[x \notin S] < \epsilon}{\{\lightning(\epsilon)\}\ \mathsf{rand}(N)\ \{x.\, x \in S\}}$$

# Error Credit Rules: Splitting

$$↯(\epsilon_1 + \epsilon_2) \dashv\vdash ↯(\epsilon_1) * ↯(\epsilon_2)$$

Implies sequencing:

$$\frac{\{↯(\epsilon_1) * P\}\ e_1\ \{Q\} \qquad \{↯(\epsilon_2) * Q\}\ e_2\ \{R\}}{\{↯(\epsilon_1 + \epsilon_2) * P\}\ e_1; e_2\ \{R\}}$$

# Error Credit Rules: Max Error

Error credit bounded by 1: owning more than 1 error credit is inconsistent.

$$↯(1) \vdash \text{False}$$

Intuition: $↯(\epsilon)$ is permission to fail with probability at most $\epsilon$, so $↯(1)$ permits always failing. Nothing to prove in such a case.

# Error Credit Rules: Averaging

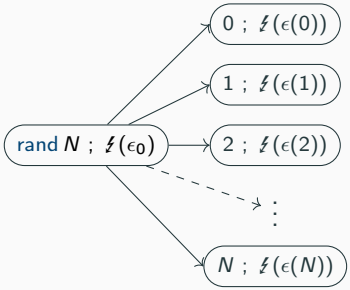
Increase error credits along branches of a sample, as long as expected value is the same.

$$\frac{\mathbb{E}_{x \sim \mathsf{Unif}(N)}[\epsilon(x)] = \epsilon_0}{\{\lightning(\epsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ \lightning(\epsilon(x))\}}$$

# Error Credit Rules: Averaging

Increase error credits along branches of a sample, as long as expected value is the same.

$$\frac{\mathbb{E}_{x \sim \mathsf{Unif}(N)}[\epsilon(x)] = \epsilon_0}{\{↯(\epsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ ↯(\epsilon(x))\}}$$

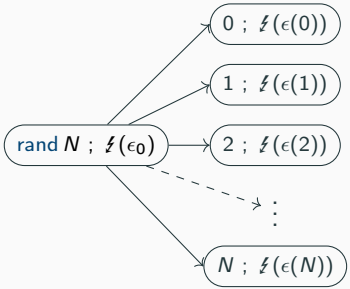


# Error Credit Rules: Averaging

Increase error credits along branches of a sample, as long as expected value is the same.

$$\frac{\mathbb{E}_{x \sim \mathsf{Unif}(N)}[\epsilon(x)] = \epsilon_0}{\{\lightning(\epsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ \lightning(\epsilon(x))\}}$$

Can derive spending rule by combining with $\lightning(1) \vdash \mathsf{False}$



# Resolves Limitation 1: Error dependency

Recall in aHL error for later operations could not depend on previous steps:

$$\frac{\{P\}\ e_1\ \{Q\}_{\epsilon_1} \qquad \{Q\}\ e_2\ \{R\}_{\epsilon_2}}{\{P\}\ e_1; e_2\ \{R\}_{\epsilon_1+\epsilon_2}}$$

In Eris, support for dependency is automatic from standard sequencing rule:

$$\frac{\{P\}\ e_1\ \{Q\} \qquad \{Q\}\ e_2\ \{R\}}{\{P\}\ e_1; e_2\ \{R\}}$$

$Q$ can refer to state after $e_1$

# Resolves Limitation 2: Errors no longer propagate everywhere

Recall specification for list *iter*:

$$\frac{\forall x.\, \{P(x)\}\ f\ x\ \{Q(x)\}}{\left\{ \mathop{\Large *}_{x \in l} P(x) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}}$$

# Resolves Limitation 2: Errors no longer propagate everywhere

Recall specification for list *iter*:

$$\frac{\forall x.\, \{P(x)\}\ f\ x\ \{Q(x)\}}{\left\{ \mathop{\Large *}_{x \in l} P(x) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}}$$

Can derive error credit version:

$$\frac{\left\{P'(x) * \lightning(\epsilon(x))\right\}\ f\ x\ \{Q(x)\}}{\left\{ \mathop{\Large *}_{x \in l} P(x) * \lightning(\epsilon(x)) \right\}\ \mathit{iter}\ f\ l\ \left\{ \mathop{\Large *}_{x \in l} Q(x) \right\}}$$

# Resolves Limitation 3: Can amortize costs

Amortize by putting error credits in representation predicate.

Example: can give constant error cost* $\epsilon$ for non-colliding hash

$$\{↯(\epsilon) * \mathit{hash}(f, m) * k \notin \mathrm{dom}(m)\}\ f\ \ k\ \{v.\ \mathit{hash}(f, m[k := v]) * v \notin \mathrm{cod}(m)\}$$

* up to a maximum number of insertions

# Retains Expressivity of Modern Separation Logic

Eris preserves other standard rules of modern separation logic.

# Retains Expressivity of Modern Separation Logic

Eris preserves other standard rules of modern separation logic.

Generalizations:

- **Coneris** [ICFP25] gives worst-case error bounds across all schedules of concurrent programs.
- **Continuous-Eris** [LICS26] handles exact real arithmetic samplers for continuous distributions

# Retains Expressivity of Modern Separation Logic

Eris preserves other standard rules of modern separation logic.

Generalizations:

- **Coneris** [ICFP25] gives worst-case error bounds across all schedules of concurrent programs.
- **Continuous-Eris** [LICS26] handles exact real arithmetic samplers for continuous distributions

Example case studies:

- Merkle tree
- Resizing hash tables
- Concurrent Bloom Filter


# Rocq Exercises