Verifying Probabilistic Programs Using Separation Logic

TutorialFest @ POPL 2026

Alejandro Aguirre[*], Simon Oddershede Gregersen[†], Philipp G. Haselwarter[*]
[*] *Aarhus University,* [†] *CISPA Helmholtz Center for Information Security*

January 11, 2026

Motivation: Bloom filters

- Suppose you want to filter network traffic.
- You maintain a list of malicious IPs, which might be in the order of millions
- Whenever you receive a request, you lookup the IP up, and if it is in the list, you block it
- You receive many requests per second
- The vast majority of requests will not be malicious

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init` h_1, h_2, h_3

0	0	0	0	0	0	0	0

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init` h_1, h_2, h_3

0	0	0	0	0	0	0	0

`insert(x)`

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init`	h_1, h_2, h_3	$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$
`insert(x)`	$h_1(x), h_2(x), h_3(x) \xrightarrow{🎲}$	

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init`	h_1, h_2, h_3	$\begin{array}{\|c\|c\|c\|c\|c\|c\|c\|c\|}\hline 0&0&0&0&0&0&0&0\\\hline\end{array}$
`insert(x)`	$h_1(x), h_2(x), h_3(x) \longrightarrow$	$\begin{array}{\|c\|c\|c\|c\|c\|c\|c\|c\|}\hline 0&1&0&0&1&0&1&0\\\hline\end{array}$

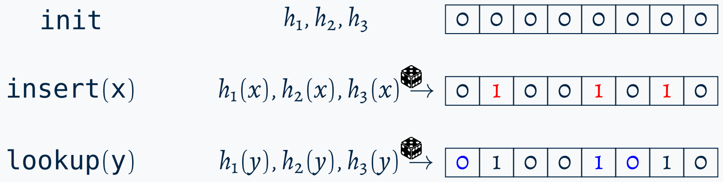
Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init`	h_1, h_2, h_3	$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$
`insert(x)`	$h_1(x), h_2(x), h_3(x) \xrightarrow{🎲}$	$\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}$
`lookup(y)`	$h_1(y), h_2(y), h_3(y) \xrightarrow{🎲}$	

Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.



Bloom filters

Bloom filters are probabilistic data structures to represent sets with $\mathcal{O}(1)$ insertion and membership queries, but some low probability of false positives.

`init`	h_1, h_2, h_3	$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$
`insert(x)`	$h_1(x), h_2(x), h_3(x) \longrightarrow$	$\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}$
`lookup(y)`	$h_1(y), h_2(y), h_3(y) \longrightarrow$	$\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}$

- If `lookup(y)` returns `false` $\Rightarrow$ `y` is definitely not in the set
- If `lookup(y)` returns `true` $\Rightarrow$ `y` is possibly in the set, do further processing

Bloom filters

What would be a good specification for a Bloom filter?

Bloom filters

What would be a good specification for a Bloom filter?

$$\{\mathsf{True}\}\ \mathsf{init}()\ \{l.\mathsf{isSet}(l, \emptyset)\}$$

$$\{\mathsf{isSet}(l, S)\}\ \mathsf{insert}(l, x)\ \{\_.\mathsf{isSet}(l, S \cup \{x\})\}$$

$$\{\mathsf{isSet}(l, S)\}\ \mathsf{lookup}(l, x)\ \{v.(v = \mathsf{if}\ x \in S\ \mathsf{then\ true\ else\ false}) * \mathsf{isSet}(l, S)\}$$

Bloom filters

What would be a good specification for a Bloom filter?

$$\{\mathsf{True}\}\ \mathsf{init}()\ \{l.\mathsf{isSet}(l, \emptyset)\}$$

$$\{\mathsf{isSet}(l, S)\}\ \mathsf{insert}(l, x)\ \{\_.\mathsf{isSet}(l, S \cup \{x\})\}$$

$$\{\mathsf{isSet}(l, S)\}\ \mathsf{lookup}(l, x)\ \{v.(v = \mathsf{if}\ x \in S\ \mathsf{then\ true\ else\ false}) * \mathsf{isSet}(l, S)\}$$

However, this does not work, the specifications do not account for false positives

Primality testing

- Key generation begins by generating a large prime (1024 bits)

Primality testing

- Key generation begins by generating a large prime (1024 bits)
- One possibility: generate a large number at random, test if it is prime, otherwise repeat

Primality testing

- Key generation begins by generating a large prime (1024 bits)
- One possibility: generate a large number at random, test if it is prime, otherwise repeat
- Naïve primality test: all divisors until square root ($\sim 2^{512}$ candidates)

Primality testing

- Key generation begins by generating a large prime (1024 bits)
- One possibility: generate a large number at random, test if it is prime, otherwise repeat
- Naïve primality test: all divisors until square root ($\sim 2^{512}$ candidates)
- Slightly less naïve primality test: all *odd* divisors until square root ($\sim 2^{511}$ candidates)

Miller-Rabin test

A probabilistic primality test (actually, a compositeness test). On each round:

```
def isPrime(n):
    if n == 1: return False
    elif n in [2,3,5,7] : return True
    elif n == 9 or n%2 == 0: return False
    else: # Factor (n-1) as m * 2^k
        m, k = (n-1), 0
        while (m % 2 == 0):
            m = m // 2
            k = k + 1
        for i in range(50):
            # Pick a random number in [2..n-2]
            x = random.randint(2, n - 2)
            # Compute x^m (mod n)
            b = pow(x,m,n)
            if b == 1 or b == n-1: continue
            else: # Compute x^(m * 2^j) (mod(n))
                for j in range(k-1):
                    b = pow(b,2,n)
                    if b == n-1: break
                else : return False
        return True
```

Miller-Rabin test

A probabilistic primality test (actually, a compositeness test). On each round:

- If n is prime, `isPrime(n)` always returns `True`

```python
def isPrime(n):
    if n == 1: return False
    elif n in [2,3,5,7] : return True
    elif n == 9 or n%2 == 0: return False
    else: # Factor (n-1) as m * 2^k
        m, k = (n-1), 0
        while (m % 2 == 0):
            m = m // 2
            k = k + 1
        for i in range(50):
            # Pick a random number in [2..n-2]
            x = random.randint(2, n - 2)
            # Compute x^m (mod n)
            b = pow(x,m,n)
            if b == 1 or b == n-1: continue
            else: # Compute x^(m * 2^j) (mod(n))
                for j in range(k-1):
                    b = pow(b,2,n)
                    if b == n-1: break
                else : return False
        return True
```

Miller-Rabin test

A probabilistic primality test (actually, a compositeness test). On each round:

- If n is prime, `isPrime(n)` always returns `True`
- If n is composite, `isPrime(n)` returns `False` with probability 1/2

```python
def isPrime(n):
    if n == 1: return False
    elif n in [2,3,5,7] : return True
    elif n == 9 or n%2 == 0: return False
    else: # Factor (n-1) as m * 2^k
        m, k = (n-1), 0
        while (m % 2 == 0):
            m = m // 2
            k = k + 1
        for i in range(50):
            # Pick a random number in [2..n-2]
            x = random.randint(2, n - 2)
            # Compute x^m (mod n)
            b = pow(x,m,n)
            if b == 1 or b == n-1: continue
            else: # Compute x^(m * 2^j) (mod(n))
                for j in range(k-1):
                    b = pow(b,2,n)
                    if b == n-1: break
                else : return False
        return True
```

Miller-Rabin test

A probabilistic primality test (actually, a compositeness test). On each round:

- If n is prime, `isPrime(n)` always returns **`True`**
- If n is composite, `isPrime(n)` returns **`False`** with probability $1/2$

Over 50 rounds, we get a wrong result with probability $1/(2^{50})$

What should be the specification of this program?

```
def isPrime(n):
    if n == 1: return False
    elif n in [2,3,5,7] : return True
    elif n == 9 or n%2 == 0: return False
    else: # Factor (n-1) as m * 2^k
        m, k = (n-1), 0
        while (m % 2 == 0):
            m = m // 2
            k = k + 1
        for i in range(50):
            # Pick a random number in [2..n-2]
            x = random.randint(2, n - 2)
            # Compute x^m (mod n)
            b = pow(x,m,n)
            if b == 1 or b == n-1: continue
            else: # Compute x^(m * 2^j) (mod(n))
                for j in range(k-1):
                    b = pow(b,2,n)
                    if b == n-1: break
                else : return False
        return True
```

Plan for today

1. A primer on Separation Logic and Iris
2. Probabilistic programs: syntax and semantics
3. A probabilistic separation logic: Eris
4. Supervised Rocq hacking
5. Eris case studies
6. Supervised Rocq hacking
7. Almost sure termination and error induction (time permitting)
8. Outlook on the Clutch Project

Plan for today

1. A primer on Separation Logic and Iris
2. Probabilistic programs: syntax and semantics
3. A probabilistic separation logic: Eris
4. Supervised Rocq hacking
5. Eris case studies
6. Supervised Rocq hacking
7. Almost sure termination and error induction (time permitting)
8. Outlook on the Clutch Project

BONUS: Come see Virgil's talk at CPP, on Tuesday at 16:45!

Plan for today

1. A primer on Separation Logic and Iris
2. Probabilistic programs: syntax and semantics
3. A probabilistic separation logic: Eris
4. Supervised Rocq hacking
5. Eris case studies
6. Supervised Rocq hacking
7. Almost sure termination and error induction (time permitting)
8. Outlook on the Clutch Project

BONUS: Come see Virgil's talk at CPP, on Tuesday at 16:45!

BONUS^2: Come see Puming's poster at the SRC, on Wednesday at 17:30!

Separation Logic: Affine Connectives & Rules

- In separation logic, assertions describe ownership of *resources* (historically: predicates on partial heaps)
- Intuition for $A \vdash B$ "A contains enough resources to obtain B"

Separation Logic: Affine Connectives & Rules

- In separation logic, assertions describe ownership of *resources* (historically: predicates on partial heaps)
- Intuition for $A \vdash B$ "*A* contains enough resources to obtain *B*"
- Resources cannot be copied $$\xcancel{\dfrac{P \vdash Q_1 \qquad P \vdash Q_2}{P \vdash Q_1 * Q_2}}$$

Separation Logic: Affine Connectives & Rules

- In separation logic, assertions describe ownership of *resources* (historically: predicates on partial heaps)
- Intuition for $A \vdash B$ "*A* contains enough resources to obtain *B*"
- Resources cannot be copied $\cancel{\dfrac{P \vdash Q_1 \qquad P \vdash Q_2}{P \vdash Q_1 * Q_2}}$
- Resources are *affine*, proofs track how resources are split & consumed

$$\frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad\qquad \frac{R \vdash P * (P \mathrel{-\!\!*} Q)}{R \vdash Q}$$

- Rules for quantifiers *etc* are unchanged

Separation Logic: Hoare Triples & Framing

- Hoare triples specify programs: $\{P\}\, e\, \{v.Q\}$

Separation Logic: Hoare Triples & Framing

- Hoare triples specify programs: $\{P\}\, e\, \{v.Q\}$
- "Given resources in precondition P,
  if e terminates with result value v,
  then postcondition $Q[v]$ is satisfied"

Separation Logic: Hoare Triples & Framing

- Hoare triples specify programs: $\{P\}\, e\, \{v.Q\}$
- "Given resources in precondition P,
  if e terminates with result value v,
  then postcondition $Q[v]$ is satisfied"
- Only need to track resources needed by e

$$\frac{\{P\}\, e\, \{v.Q\}}{\{P * R\}\, e\, \{v.Q * R\}}\ \textsc{Hoare-frame}$$

Separation Logic: Hoare Triples & Framing

- Hoare triples specify programs: $\{P\}\, e\, \{v.Q\}$
- "Given resources in precondition P,
  if e terminates with result value v,
  then postcondition $Q[v]$ is satisfied"
- Only need to track resources needed by e

$$\frac{\{P\}\, e\, \{v.Q\}}{\{P * R\}\, e\, \{v.Q * R\}}\ \textsc{Hoare-frame}$$

- Enables *local* reasoning, scales to large programs

Separation Logic: Resources and Knowledge

- Resources logically track program state, *e.g.*, the assertion

  $\ell_1 \mapsto 7 * \ell_2 \mapsto \text{"}\textit{hello world!}\text{"}$ describes a heap with an integer and a string

Separation Logic: Resources and Knowledge

- Resources logically track program state, *e.g.*, the assertion

  $\ell_1 \mapsto 7 * \ell_2 \mapsto \text{"}\textit{hello world!}\text{"}$ describes a heap with an integer and a string

- Resources can be *updated*

  $\{\ell_1 \mapsto 7\}\ \ell_1 \leftarrow 42\ \{\ell_1 \mapsto 42\}$

Separation Logic: Resources and Knowledge

- Resources logically track program state, *e.g.*, the assertion

  $\ell_1 \mapsto 7 * \ell_2 \mapsto \text{"\textit{hello world!}"}$ describes a heap with an integer and a string

- Resources can be *updated*

  $\{\ell_1 \mapsto 7\}\ \ell_1 \leftarrow 42\ \{\ell_1 \mapsto 42\}$

- *Pure*, mathematical facts can be embedded in SL as a resource, *e.g.*,

  $\ulcorner \mathit{isPrime}(53) \urcorner$ describes *knowledge* about 53

Separation Logic: Resources and Knowledge

- Resources logically track program state, *e.g.*, the assertion

  $\ell_1 \mapsto 7 * \ell_2 \mapsto \textit{"hello world!"}$ describes a heap with an integer and a string

- Resources can be *updated*

  $\{\ell_1 \mapsto 7\}\ \ell_1 \leftarrow 42\ \{\ell_1 \mapsto 42\}$

- *Pure*, mathematical facts can be embedded in SL as a resource, *e.g.*,
  $\ulcorner \mathit{isPrime}(53) \urcorner$ describes *knowledge* about 53

- Pure assertions can be copied; no need to track them in proofs

Separation Logic and Iris

Iris is a higher-order separation logic framework implemented in the Rocq prover

- Higher-order, impredicative assertions: $\{\,\{\Phi\}\,\mathsf{sort}(l)\,\{\Phi'\}\,\}\ e\ \{\Psi\}$
- Expressive resource model, including user-defined
- Interactive proof mode implemented in Rocq

ROCQ

Iris

A probabilistic sequential language

Sequential ML-like language with discrete sampling:

$$
\begin{aligned}
v, w \in \mathit{Val} &::= z \in \mathbb{Z} \mid b \in \mathbb{B} \mid () \mid \ell \in \mathit{Loc} \mid \mathsf{rec}\,\mathsf{f}\,\mathsf{x} = e \mid (v, w) \mid \mathsf{inl}\,v \mid \mathsf{inr}\,v \\
e \in \mathit{Expr} &::= v \mid \mathsf{x} \mid \mathsf{rec}\,\mathsf{f}\,\mathsf{x} = e \mid e_1\,e_2 \mid e_1 + e_2 \mid e_1 - e_2 \mid \ldots \mid \mathsf{if}\,e\,\mathsf{then}\,e_1\,\mathsf{else}\,e_2 \mid \\
&\quad (e_1, e_2) \mid \mathsf{fst}\,e \mid \mathsf{snd}\,e \mid \mathsf{inl}(e) \mid \mathsf{inr}(e) \mid \mathsf{match}\,e\,\mathsf{with}\ \mathsf{inl}\,v \Rightarrow e_1 \mid \mathsf{inr}\,w \Rightarrow e_2\ \mathsf{end} \mid \\
&\quad \mathsf{allocn}\,e_1\,e_2 \mid\ !\,e \mid e_1 \leftarrow e_2 \mid \mathsf{flip} \mid \mathsf{rand}\,e \\
K \in \mathit{Ectx} &::= - \mid e\,K \mid K\,v \mid \mathsf{allocn}\,K \mid\ !\,K \mid e \leftarrow K \mid K \leftarrow v \mid \mathsf{rand}\,K \mid \ldots
\end{aligned}
$$

$$
\sigma \in \mathit{State} \triangleq (\mathit{Loc} \xrightarrow{\mathrm{fin}} \mathit{Val}) \qquad\qquad \rho \in \mathit{Cfg} \triangleq \mathit{Expr} \times \mathit{State}
$$

Probability distributions

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

Probability distributions

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

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

Probability distributions

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

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

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

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

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. $\sum_{i \in I} p_i \leqslant 1$
- For each $i \in I$, a distribution ν_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. $\sum_{i \in I} p_i \leqslant 1$
- For each $i \in I$, a distribution ν_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. $\sum_{i \in I} p_i \leqslant 1$
- For each $i \in I$, a distribution ν_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

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

Operational semantics

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

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

Operational semantics

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

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

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

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

Probabilistic evaluation

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

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

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

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

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

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

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:

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

Example: Geometric distribution

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

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

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

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

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

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

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

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:

$$\{(0, \square) \mapsto \frac{1}{2},\ (1, \square) \mapsto \frac{1}{4},\ (2, \square) \mapsto \frac{1}{8},\ \dots, (n-1, \square) \mapsto \frac{1}{2^n}\}$$

By taking limits, we get a full distribution

$$(g(0), \square) \Downarrow \{(0, \square) \mapsto \frac{1}{2},\ (1, \square) \mapsto \frac{1}{4},\ (2, \square) \mapsto \frac{1}{8},\ \dots, (n-1, \square) \mapsto \frac{1}{2^n},\ \dots\}$$

A Probabilistic Separation Logic: Eris [ICFP 24]

A higher-order separation logic to reason about probability of errors in higher-order probabilistic programs

- Error probability represented as a separation logic resource $\lightning(\varepsilon)$
- $\lightning(\varepsilon)$ allows to "pay" for a step that fails with probability $\leqslant \varepsilon$
- Compositionality and modularity inherited "for free" from the ambient HO separation logic
- The resource representation enables new reasoning principles
- Fully mechanized in Rocq and Iris

Eris [ICFP ’24]

The core concept of Eris is a new type of resource, error credits

Eris [ICFP '24]

The core concept of Eris is a new type of resource, error credits

- $\lightning(\varepsilon)$ asserts ownership of ε error credits, where $\varepsilon \in [0, 1]$

Eris [ICFP '24]

The core concept of Eris is a new type of resource, error credits

- $\lightning(\varepsilon)$ asserts ownership of ε error credits, where $\varepsilon \in [0, 1]$
- Intuition: we can spend $\lightning(\varepsilon)$ to prevent an error that happens with probability $\leqslant \varepsilon$

Eris [ICFP '24]

The core concept of Eris is a new type of resource, error credits

- $↯(\varepsilon)$ asserts ownership of ε error credits, where $\varepsilon \in [0, 1]$
- Intuition: we can spend $↯(\varepsilon)$ to prevent an error that happens with probability $\leqslant \varepsilon$
- Error credits obey the following laws:

$$\vdash ↯(0) \qquad ↯(\varepsilon_1) * ↯(\varepsilon_2) \dashv\vdash ↯(\varepsilon_1 + \varepsilon_2) \qquad ↯(1) \vdash \bot$$

Eris Hoare Triples

Hoare triples in Eris look similar on the surface to regular Hoare triples:

$$\frac{\Gamma \vdash P : \mathit{Prop} \qquad \Gamma \vdash e : \mathit{Expr} \qquad \Gamma, v : \mathit{Val} \vdash Q : \mathit{Prop}}{\Gamma \vdash \{P\}\, e\, \{v.Q\} : \mathit{Prop}}$$

Eris Hoare Triples

Hoare triples in Eris look similar on the surface to regular Hoare triples:

$$\frac{\Gamma \vdash P : \mathit{Prop} \qquad \Gamma \vdash e : \mathit{Expr} \qquad \Gamma, v : \mathit{Val} \vdash Q : \mathit{Prop}}{\Gamma \vdash \{P\}\, e\, \{v.Q\} : \mathit{Prop}}$$

However, they have a *probabilistic interpretation*. In particular, for $v : \mathit{Val} \vdash \varphi : \mathit{Prop}$, $\{⚡(\varepsilon)\}\, e\, \{v.\varphi(v)\}$ holds if:

- The probability that e gets stuck is at most ε, and
- The probability that e returns v such that $\neg\varphi(v)$ is at most ε

Rules for deterministic commands

All the usual rules for the determinisic fragment of the language still hold, e.g.:

$$\frac{}{S \vdash \{l \mapsto u\}\ !\,l\,\{v\,.\,v = u * l \mapsto v\}}\ \textsc{ht-load}$$

$$\frac{}{S \vdash \{l \mapsto u\}\, l \leftarrow w\,\{v\,.\,v = () * l \mapsto w\}}\ \textsc{ht-store}$$

$$\frac{S \vdash \{P * b = \mathsf{true}\}\, e_1\, \{v\,.\,Q\} \qquad S \vdash \{P * b = \mathsf{false}\}\, e_2\, \{v\,.\,Q\}}{S \vdash \{P\}\, \mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\, \{v\,.\,Q\}}\ \textsc{ht-if}$$

Structural rules

Perhaps more surprising, structural rules are still the same!

$$\frac{K \text{ eval. ctx.} \qquad S \vdash \{P\}\, e\, \{v\,.\,Q\} \qquad S \vdash \forall v.\{Q\}\, K[v]\, \{w\,.\,R\}}{S \vdash \{P\}\, K[e]\, \{w\,.\,R\}}\ \textsc{ht-bind}$$

$$\frac{S \vdash \{P\}\, e\, \{v\,.\,Q\}}{S \vdash \{P * R\}\, e\, \{v\,.\,Q * R\}}\ \textsc{ht-frame}$$

Structural rules

Perhaps more surprising, structural rules are still the same!

$$\frac{K \text{ eval. ctx.} \qquad S \vdash \{P\}\, e\, \{v\,.\, Q\} \qquad S \vdash \forall v. \{Q\}\, K[v]\, \{w\,.\, R\}}{S \vdash \{P\}\, K[e]\, \{w\,.\, R\}}\ \textsc{ht-bind}$$

$$\frac{S \vdash \{P\}\, e\, \{v\,.\, Q\}}{S \vdash \{P * R\}\, e\, \{v\,.\, Q * R\}}\ \textsc{ht-frame}$$

Two observations:

- Ensuring this is not trivial
- Having these rules is the key to making Eris work

Rules for probabilistic commands

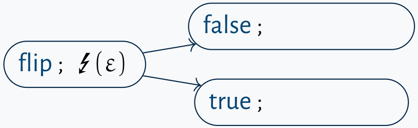
Error credits can be distributed along probabilistic choices:

$$\dfrac{\dfrac{F(\mathsf{true}) + F(\mathsf{false})}{2} = \varepsilon}{S \vdash \{ ↯(\varepsilon) \}\ \mathsf{flip}\ \{ b : \mathbb{B}\ .\ ↯(F(b)) \}}\ \textsc{ht-flip}$$

Rules for probabilistic commands

Error credits can be distributed along probabilistic choices:

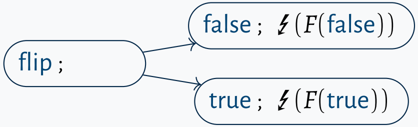
$$\dfrac{\dfrac{F(\mathsf{true}) + F(\mathsf{false})}{2} = \varepsilon}{S \vdash \{⚡(\varepsilon)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ ⚡(F(b))\}}\ \textsc{ht-flip}$$



Rules for probabilistic commands

Error credits can be distributed along probabilistic choices:

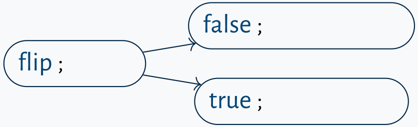
$$\frac{\dfrac{F(\mathsf{true}) + F(\mathsf{false})}{2} = \varepsilon}{S \vdash \{ \lightning(\varepsilon) \}\ \mathsf{flip}\ \{ b : \mathbb{B}\ .\ \lightning(F(b)) \}}\ \textsc{ht-flip}$$



Rules for probabilistic commands

Error credits can be distributed along probabilistic choices:

$$\dfrac{\dfrac{F(\mathsf{true}) + F(\mathsf{false})}{2} = \varepsilon}{S \vdash \{\lightning(\varepsilon)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ \lightning(F(b))\}}\ \textsc{ht-flip}$$



Rules for probabilistic commands

Error credits can be distributed along probabilistic choices:

$$\frac{\dfrac{F(\mathsf{true}) + F(\mathsf{false})}{2} = \varepsilon}{S \vdash \{↯(\varepsilon)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ ↯(F(b))\}}\ \textsc{ht-flip} \qquad \frac{\dfrac{F(0) + \cdots + F(N)}{N+1} = \varepsilon}{S \vdash \{↯(\varepsilon)\}\ \mathsf{rand}\ N\{n : \mathbb{N}\ .\ (n \leqslant N) * ↯(F(n))\}}\ \textsc{ht-rand}$$

Derived rules

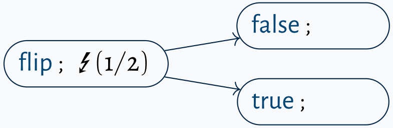
One simple derived rule is that we can spend $\lightning(1/2)$ to choose the outcome of a flip:

$$\frac{}{S \vdash \{\lightning(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{true}\}}\ \text{HT-FLIP-T}$$

Derived rules

One simple derived rule is that we can spend $\lightning(1/2)$ to choose the outcome of a flip:

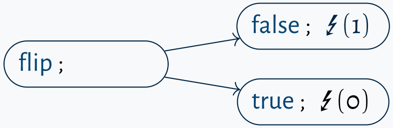
$$\frac{}{S \vdash \{\lightning(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{true}\}}\ \textsc{ht-flip-t}$$



Derived rules

One simple derived rule is that we can spend ↯(1/2) to choose the outcome of a flip:

$$\frac{}{S \vdash \{↯(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{true}\}}\ \text{HT-FLIP-T}$$



Derived rules

One simple derived rule is that we can spend $\lightning(1/2)$ to choose the outcome of a flip:

$$\frac{}{S \vdash \{\lightning(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{true}\}}\ \text{HT-FLIP-T}$$

$$\begin{tikzcd}
 & \mathsf{false}\,;\ \lightning(1) \arrow[r, Rightarrow] & \bot \\
\mathsf{flip}\,; \arrow[ru] \arrow[rd] & & \\
 & \mathsf{true}\,;\ \lightning(0) &
\end{tikzcd}$$

Derived rules

One simple derived rule is that we can spend $\lightning(1/2)$ to choose the outcome of a flip:

$$\frac{}{S \vdash \{\lightning(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{true}\}}\ \textsc{ht-flip-t}$$

$$\frac{}{S \vdash \{\lightning(1/2)\}\ \mathsf{flip}\ \{b : \mathbb{B}\ .\ b = \mathsf{false}\}}\ \textsc{ht-flip-f}$$

Derived rules

In general, $\lightning(\varepsilon)$ credits can be used to avoid a set of outcomes whose probability is $\leqslant \varepsilon$

Derived rules

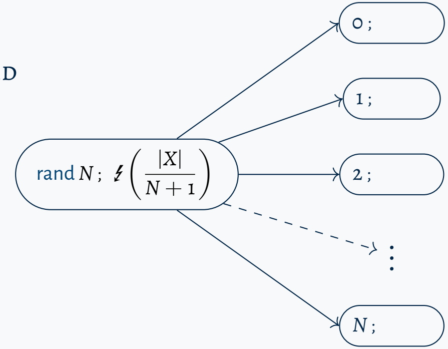
In general, $\lightning(\varepsilon)$ credits can be used to avoid a set of outcomes whose probability is $\leqslant \varepsilon$

$$\frac{X \subseteq \{0, \ldots, N\} \qquad \varepsilon = \dfrac{|X|}{N+1}}{\vdash \{\lightning(\varepsilon)\}\ \mathsf{rand}\, N\, \{n \,.\, n \notin X\}}\ \textsc{rand-avoid}$$

Derived rules

In general, $\lightning(\varepsilon)$ credits can be used to avoid a set of outcomes whose probability is $\leqslant \varepsilon$

$$\frac{X \subseteq \{0, \ldots, N\} \qquad \varepsilon = \dfrac{|X|}{N+1}}{\vdash \{\lightning(\varepsilon)\}\ \mathsf{rand}\, N\, \{n \,.\, n \notin X\}}\ \textsc{rand-avoid}$$

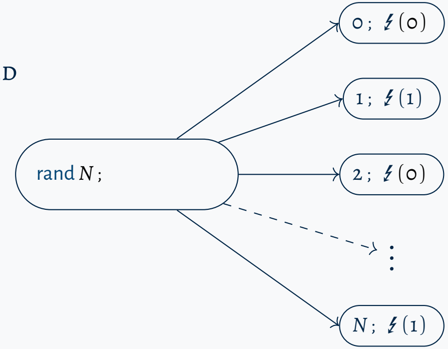


$\mathsf{rand}\, N\,;\ \lightning\left(\dfrac{|X|}{N+1}\right)$
$0\,;$
$1\,;$
$2\,;$
$\vdots$
$N\,;$

Derived rules

In general, $\lightning(\varepsilon)$ credits can be used to avoid a set of outcomes whose probability is $\leqslant \varepsilon$

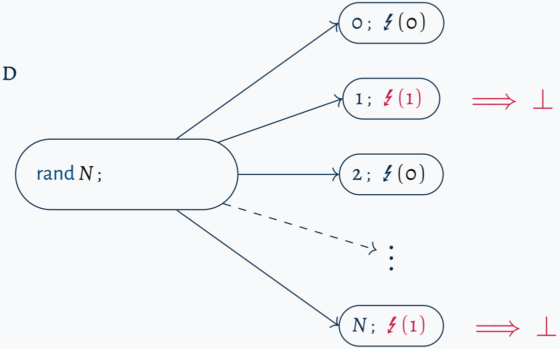
$$\frac{X \subseteq \{0, \ldots, N\} \qquad \varepsilon = \dfrac{|X|}{N+1}}{\vdash \{\lightning(\varepsilon)\}\ \mathsf{rand}\, N\, \{n\,.\, n \notin X\}}\ \textsc{rand-avoid}$$



Derived rules

In general, $\lightning(\varepsilon)$ credits can be used to avoid a set of outcomes whose probability is $\leqslant \varepsilon$

$$\frac{X \subseteq \{0, \ldots, N\} \qquad \varepsilon = \dfrac{|X|}{N+1}}{\vdash \{\lightning(\varepsilon)\}\ \mathsf{rand}\, N\ \{n\ .\ n \notin X\}}\ \textsc{rand-avoid}$$


rand N ;
0 ; $\lightning(0)$
1 ; $\lightning(1)$ $\Longrightarrow \perp$
2 ; $\lightning(0)$
$\vdots$
N ; $\lightning(1)$ $\Longrightarrow \perp$

Example: Double flip

$$\mathsf{twoFlip}\;() \triangleq \mathsf{if}\;\;\mathsf{flip}\;\;\mathsf{then}\;1\;\mathsf{else}\;\mathsf{if}\;\;\mathsf{flip}\;\;\mathsf{then}\;1\;\mathsf{else}\;0$$

We want to reason about the probability of observing 1 in the output

Example: Double flip

$$\mathsf{twoFlip}\,() \triangleq \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

We want to reason about the probability of observing 1 in the output

$$\{\lightning\,(??)\}\ \mathsf{twoFlip}\ \{v.\,v = 1\}$$

Example: Double flip

$$\mathsf{twoFlip}\ () \triangleq \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else\ if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

We want to reason about the probability of observing 1 in the output

$$\{ ↯ (1/4)\}\ \mathsf{twoFlip}\ \{v. v = 1\}$$

Example: Double flip (Proof sketch)

$$\mathsf{twoFlip}\,()\triangleq \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

To show: $\{\lightning(1/4)\}\ \mathsf{twoFlip}\ \{v.\, v = 1\}$

Example: Double flip (Proof sketch)

$$\mathsf{twoFlip}\,()\triangleq \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

To show: $\{ \lightning\,(1/4)\}\ \mathsf{twoFlip}\ \{v.v = 1\}$

1. We begin by applying HT-FLIP around the first flip

Example: Double flip (Proof sketch)

$$\text{twoFlip}\,() \triangleq \text{if flip then } 1 \text{ else if flip then } 1 \text{ else } 0$$

To show: $\{\lightning\,(1/4)\}\,\text{twoFlip}\,\{v.\, v = 1\}$

1. We begin by applying HT-FLIP around the first flip
2. We apply HT-FLIP with the function $F(b) \triangleq \text{if } b \text{ then } 0 \text{ else } (1/2)$

Example: Double flip (Proof sketch)

$$\mathsf{twoFlip}\,() \triangleq \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \ \mathsf{flip}\ \ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

To show: $\{\lightning\,(1/4)\}\,\mathsf{twoFlip}\,\{\mathsf{v}.\mathsf{v} = 1\}$

1. We begin by applying HT-FLIP around the first flip
2. We apply HT-FLIP with the function $F(b) \triangleq \mathsf{if}\ b\ \mathsf{then}\ 0\ \mathsf{else}\ (1/2)$
3. Case distinction on b:
   - If $b = \mathsf{true}$ we go to the then branch and conclude with HT-RET
   - Otherwise we go to the else branch, now with $\lightning\,(1/2)$

Example: Double flip (Proof sketch)

$$\mathsf{twoFlip}\,() \triangleq \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

To show: $\{↯(1/4)\}\ \mathsf{twoFlip}\ \{v.v = 1\}$

1. We begin by applying HT-FLIP around the first flip
2. We apply HT-FLIP with the function $F(b) \triangleq \mathsf{if}\ b\ \mathsf{then}\ 0\ \mathsf{else}\ (1/2)$
3. Case distinction on b:
   - If $b = \mathsf{true}$ we go to the then branch and conclude with HT-RET
   - Otherwise we go to the else branch, now with $↯(1/2)$
4. We have $↯(1/2)$, so we can apply HT-FLIP-T

Example: Double flip (Proof sketch)

$$\mathsf{twoFlip}\,() \triangleq \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

To show: $\{ ↯(1/4) \}\ \mathsf{twoFlip}\ \{ v.v = 1 \}$

1. We begin by applying HT-FLIP around the first flip
2. We apply HT-FLIP with the function $F(b) \triangleq \mathsf{if}\ b\ \mathsf{then}\ 0\ \mathsf{else}\ (1/2)$
3. Case distinction on b:
   - If $b = \mathsf{true}$ we go to the then branch and conclude with HT-RET
   - Otherwise we go to the else branch, now with $↯(1/2)$
4. We have $↯(1/2)$, so we can apply HT-FLIP-T
5. We conclude by applying HT-RET

ROCQ

Iris

Tutorial materials

You can find the sources for the tutorial on our Github: https://github.com/logsem/clutch

Follow instructions for installation. Two options:

- Running a Docker container within VSCode
- Installation of dependencies through OPAM and building

Reasoning about a geometric sampler

Consider the following sampler, simulating a geometric trial with parameter $\rho = 1/3$

$$\begin{aligned} \mathsf{geo}\;() \triangleq\; & \mathsf{let}\; x = \mathsf{Rand}\; 2\; \mathsf{in} \\ & \mathsf{if}\; (x = 0)\; \mathsf{then}\; 0\; \mathsf{else}\; 1 + \mathsf{geo}\;() \end{aligned}$$

We can now write and prove specifications to describe how the output distributes

Reasoning about a geometric sampler

Consider the following sampler, simulating a geometric trial with parameter $\rho = 1/3$

$$\begin{aligned} \mathsf{geo}\ () \triangleq\ & \mathsf{let}\ \mathsf{x} = \mathsf{Rand}\ 2\ \mathsf{in} \\ & \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + \mathsf{geo}\ () \end{aligned}$$

We can now write and prove specifications to describe how the output distributes

- Prob. of sampling $v < 0$ is 0: $\{\mathsf{True}\}\ \mathsf{geo}\ ()\ \{v. 0 \leqslant v\}$

Reasoning about a geometric sampler

Consider the following sampler, simulating a geometric trial with parameter $\rho = 1/3$

$$
\begin{aligned}
\mathsf{geo}\,() \triangleq\ & \mathsf{let}\ \mathsf{x} = \mathsf{Rand}\ 2\ \mathsf{in} \\
& \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + \mathsf{geo}\,()
\end{aligned}
$$

We can now write and prove specifications to describe how the output distributes

- Prob. of sampling $v < 0$ is 0: $\{\mathsf{True}\}\ \mathsf{geo}\,()\ \{v.0 \leqslant v\}$
- Prob. of sampling $v = 0$ is at most $1/3$: $\{\lightning\,(1/3)\}\ \mathsf{geo}\,()\ \{v.0 < v\}$

Reasoning about a geometric sampler

Consider the following sampler, simulating a geometric trial with parameter $\rho = 1/3$

$$\begin{aligned} \mathsf{geo}\,() \triangleq\ & \mathsf{let}\ \mathsf{x} = \mathsf{Rand}\ 2\ \mathsf{in} \\ & \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + \mathsf{geo}\,() \end{aligned}$$

We can now write and prove specifications to describe how the output distributes

- Prob. of sampling $v < 0$ is 0: $\{\mathsf{True}\}\ \mathsf{geo}\,()\ \{v.0 \leqslant v\}$
- Prob. of sampling $v = 0$ is at most $1/3$: $\{\lightning\,(1/3)\}\ \mathsf{geo}\,()\ \{v.0 < v\}$
- Prob. of sampling $v \neq 0$ is at most $\leqslant 2/3$: $\{\lightning\,(2/3)\}\ \mathsf{geo}\,()\ \{v.0 = v\}$

Reasoning about a geometric sampler

Consider the following sampler, simulating a geometric trial with parameter $\rho = 1/3$

$$\begin{aligned} \mathsf{geo}\,() \triangleq{}& \mathsf{let}\ \mathsf{x} = \mathsf{Rand}\ 2\ \mathsf{in} \\ & \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + \mathsf{geo}\,() \end{aligned}$$

We can now write and prove specifications to describe how the output distributes

- Prob. of sampling $v < 0$ is 0: $\{\mathsf{True}\}\ \mathsf{geo}\,()\ \{v.0 \leqslant v\}$
- Prob. of sampling $v = 0$ is at most $1/3$: $\{\lightning\,(1/3)\}\ \mathsf{geo}\,()\ \{v.0 < v\}$
- Prob. of sampling $v \neq 0$ is at most $\leqslant 2/3$: $\{\lightning\,(2/3)\}\ \mathsf{geo}\,()\ \{v.0 = v\}$
- Prob. of sampling $v > n$ is at most $(2/3)^{n+1}$: $\forall n.\{\lightning\,((2/3)^{n+1})\}\ \mathsf{geo}\,()\ \{v.n \geqslant v\}$

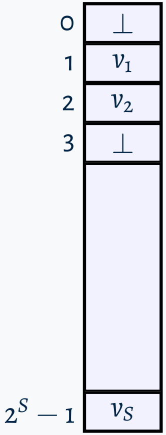
ROCQ

Iris

Modelling hash functions

We will use the random oracle model:

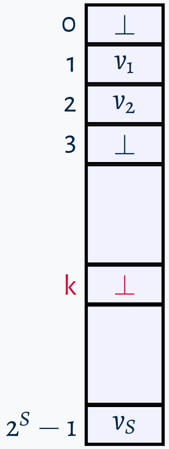
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

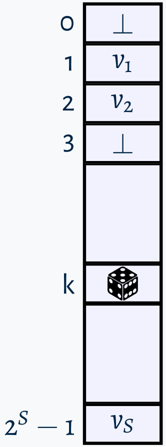
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

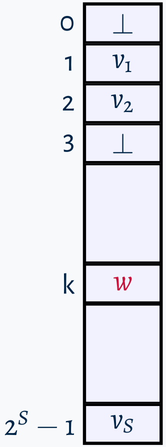
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

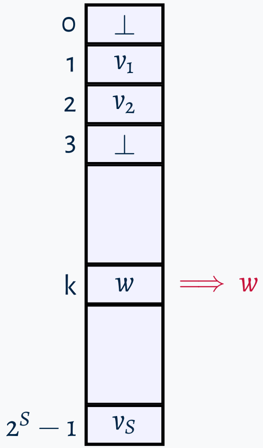
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

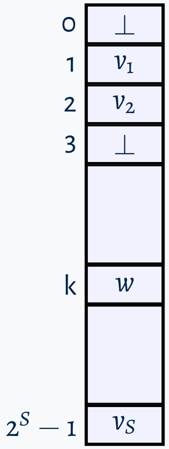
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

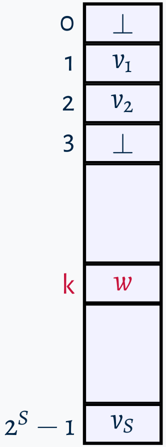
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

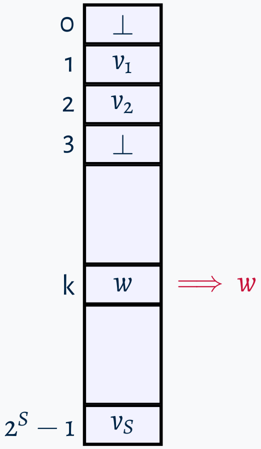
- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Modelling hash functions

We will use the random oracle model:

- Hash functions behave as mutable partial maps from keys to values
- The first time a key is hashed, we uniformly sample a value for it
- The subsequent times the key is hashed, we return the same value



Specifications for hash function

We axiomatize the behavior of hash functions through abstract specifications. The fact that a RandML function *hf* behaves as a hash is modelled by a predicate:

$$\mathsf{hashfun}\ (\mathsf{vsize} \colon \mathbb{N})\ (\mathsf{hf} \colon \mathit{Val})\ (\mathsf{m} \colon \mathbb{N} \rightarrow_{\mathsf{fin}} \mathbb{N}) \colon \mathit{iProp}$$

where:

- vsize is the size of the value space
- hf is the handle of the hash function
- m is the partial map between hash keys to hash values

Specifications for hash function

The behavior of initialization and querying is also axiomatized:

- Initialization produces a hash with an empty map

$$\{\mathsf{True}\}\ \mathsf{init}\ vs\ \{f.\ \mathsf{hashfun}\ vs\ f\ \emptyset\}$$

Specifications for hash function

The behavior of initialization and querying is also axiomatized:

- Initialization produces a hash with an empty map

$$\{\mathsf{True}\}\ \mathsf{init}\ vs\ \{f.\ \mathsf{hashfun}\ vs\ f\ \emptyset\}$$

- Querying a key in the map should return its value

$$\{\mathsf{hashfun}\ vs\ f\ m * m[k] = v\}\ f\ k\ \{w.\ w = v * \mathsf{hashfun}\ vs\ f\ m\}$$

Specifications for hash function

The behavior of initialization and querying is also axiomatized:

- Initialization produces a hash with an empty map

$$\{\mathsf{True}\}\ \mathsf{init}\ vs\ \{f.\ \mathsf{hashfun}\ vs\ f\ \emptyset\}$$

- Querying a key in the map should return its value

$$\{\mathsf{hashfun}\ vs\ f\ m * m[k] = v\}\ f\ k\ \{w.\ w = v * \mathsf{hashfun}\ vs\ f\ m\}$$

- Querying a fresh key should return a uniformly sampled value. We capture this by requiring that the probability of the key falling in a finite set of values S is $|S|/|vs|$:

$$\begin{aligned}&\{\mathsf{hashfun}\ vs\ f\ m * ⚡((|S|/vs)\cdot\varepsilon_I) * ⚡((1-|S|/vs)\cdot\varepsilon_O) * m[k] = \bot\}\\ &\quad f\ k\\ &\{w.\ \mathsf{hashfun}\ vs\ f\ m[k \mapsto w] * ((w \in S * ⚡(\varepsilon_I)) \vee (w \notin S * ⚡(\varepsilon_O)))\}\end{aligned}$$

ROCQ

Iris

Almost sure termination

Consider the fair random walk:

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

Almost sure termination

Consider the fair random walk:

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

$$(\mathsf{rw}\ 1, [\,]) \Downarrow \{() \mapsto 1\}$$

Almost sure termination

Consider the fair random walk:

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

$$(\mathsf{rw}\ 1, [])\Downarrow \{() \mapsto 1\}$$

The program terminates with probability 1. This is called *almost sure termination*.

Almost sure termination

Consider the fair random walk:

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

$$(\mathsf{rw}\ 1, [])\Downarrow \{() \mapsto 1\}$$

The program terminates with probability 1. This is called *almost sure termination*.

It also does in 2D, but not in 3D.

Reasoning about almost sure termination

Consider again the geometric trial with parameter 1/3:

$$
\begin{aligned}
\mathsf{geo}\,() \triangleq\ & \mathsf{let}\ x = \mathsf{Rand}\ 2\ \mathsf{in} \\
& \mathsf{if}\ (x \leqslant 0)\ \mathsf{then}\ 0 \\
& \qquad\qquad\quad \mathsf{else}\ 1 + \mathsf{geo}\,()
\end{aligned}
$$

We want to prove that it terminates with prob. 1. What specification should we write?

Reasoning about almost sure termination

Consider again the geometric trial with parameter 1/3:

$$\begin{aligned}\mathsf{geo}\,() \triangleq\ & \mathsf{let}\ x = \mathsf{Rand}\ 2\ \mathsf{in} \\ & \mathsf{if}\ (x \leqslant 0)\ \mathsf{then}\ 0 \\ & \qquad\qquad\quad \mathsf{else}\ 1 + \mathsf{geo}\,()\end{aligned}$$

We want to prove that it terminates with prob. 1. What specification should we write?

$$\{\mathsf{True}\}\ \mathsf{geo}\,()\ \{\mathsf{True}\}$$

But this specification is also satisfied by any diverging program…

Total correctness logic

Eris also defines a total correctness variant with the following interpretation:

$$[\lightning(\varepsilon)]\ e\ [v.\varphi(v)] \Rightarrow \begin{array}{c}\text{with probability at least } 1-\varepsilon,\ e \text{ will terminate} \\ \text{and return a result } v \text{ s.t. } \varphi(v)\end{array}$$

Total correctness logic

Eris also defines a total correctness variant with the following interpretation:

$$[\lightning(\varepsilon)]\ e\ [v.\varphi(v)] \Rightarrow \begin{array}{c}\text{with probability at least } 1-\varepsilon,\ e \text{ will terminate} \\ \text{and return a result } v \text{ s.t. } \varphi(v)\end{array}$$

Now this specification implies AST:

$$[\mathsf{True}]\ \mathsf{geo}\ ()\ [\mathsf{True}]$$

Total correctness logic

Eris also defines a total correctness variant with the following interpretation:

$$[\lightning(\varepsilon)]\ e\ [v.\varphi(v)] \Rightarrow \begin{array}{c}\text{with probability at least } 1-\varepsilon,\ e \text{ will terminate}\\ \text{and return a result } v \text{ s.t. } \varphi(v)\end{array}$$

Now this specification implies AST:

$$[\mathsf{True}]\ \mathsf{geo}\ ()\ [\mathsf{True}]$$

NOTE: the usual recursion rule becomes unsound!

$$\frac{(\forall w.\ \ [P]\ (\mathsf{rec}\, f\, x = e)\ w\ [Q]\,) \vdash\ [P]\ e[v/x][(\mathsf{rec}\, f\, x = e)/f]\ [Q]}{\vdash [P]\ (\mathsf{rec}\, f\, x = e)\ v\ [Q]}\ \text{UNSOUND}$$

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (\lightning(\varepsilon') \mathrel{-\!\!*} P) * \lightning(\varepsilon) \vdash P}{\lightning(\varepsilon) \vdash P}\ \textsc{err-ind}$$

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (↯(\varepsilon') \mathrel{-\!\!*} P) * ↯(\varepsilon) \vdash P}{↯(\varepsilon) \vdash P}\ \textsc{err-ind}$$

- Assuming a strictly positive amount of credits ε

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (\lightning(\varepsilon') \mathrel{-\!\!*} P) * \lightning(\varepsilon) \vdash P}{\lightning(\varepsilon) \vdash P}\ \textsc{err-ind}$$

- Assuming a strictly positive amount of credits ε
- We choose a strictly larger amount of credits $\varepsilon' > \varepsilon'$

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (\lightning(\varepsilon') \mathrel{-\!\!*} P) * \lightning(\varepsilon) \vdash P}{\lightning(\varepsilon) \vdash P}\ \textsc{err-ind}$$

- Assuming a strictly positive amount of credits ε
- We choose a strictly larger amount of credits $\varepsilon' > \varepsilon'$
- We get an induction hypothesis guarded by $\lightning(\varepsilon')$

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (↯(\varepsilon') \mathrel{-\!\!*} P) * ↯(\varepsilon) \vdash P}{↯(\varepsilon) \vdash P}\ \textsc{err-ind}$$

- Assuming a strictly positive amount of credits ε
- We choose a strictly larger amount of credits $\varepsilon' > \varepsilon'$
- We get an induction hypothesis guarded by $↯(\varepsilon')$
- Error credits can be obtained by taking probabilistic choices

Error induction

$$\frac{\varepsilon > 0 \qquad \varepsilon < \varepsilon' \qquad (\lightning(\varepsilon') \mathrel{-\!\!*} P) * \lightning(\varepsilon) \vdash P}{\lightning(\varepsilon) \vdash P}\ \textsc{err-ind}$$

- Assuming a strictly positive amount of credits ε
- We choose a strictly larger amount of credits $\varepsilon' > \varepsilon'$
- We get an induction hypothesis guarded by $\lightning(\varepsilon')$
- Error credits can be obtained by taking probabilistic choices

Intuitively: $d = \varepsilon' - \varepsilon > 0$, so if from $\lightning(\varepsilon)$ we can get to $\lightning(\varepsilon) * \lightning(d)$, we can repeat this until we get to $\lightning(1)$.

Kickstarting the induction

One can get a positive amount of error credits out of thin air, both for the total and for the partial logics:

$$\frac{\forall \varepsilon. \{ ↯(\varepsilon) \ast (0 < \varepsilon) \ast P \}\, e\, \{ v. Q \}}{\{ P \}\, e\, \{ v. Q \}}\ \textsc{ht-err-pos}$$

The reason is that one is proving, for all $0 < \varepsilon$, that some error happens with probability $\leqslant \varepsilon$. By continuity, the event must happen with probability 0.

ROCQ

Iris

Summary: Eris

We introduce a new class of resources, and specify their interaction with sampling

$$\frac{\frac{\varepsilon(0)+\cdots+\varepsilon(N)}{N+1} = \varepsilon'}{\vdash \{ ↯(\varepsilon') \}\ \mathsf{rand}\ N \{ n : \mathbb{N}\ .\ (n \leqslant N) * ↯(\varepsilon(n)) \}}\ \text{RAND}$$

Summary: Eris

We introduce a new class of resources, and specify their interaction with sampling

$$\frac{\frac{\varepsilon(0)+\cdots+\varepsilon(N)}{N+1} = \varepsilon'}{\vdash \{\lightning(\varepsilon')\}\ \mathsf{rand}\ N\, \{n : \mathbb{N}\,.\,(n \leqslant N) * \lightning(\varepsilon(n))\}}\ \textsc{rand}$$

These resources have minimal interaction with standard Iris

$$\frac{\dfrac{\text{Standard Iris}}{\vdash \{\Phi_{\mathsf{det}}\}\, e_1\, \{\Phi'_{\mathsf{det}}\}} \qquad \dfrac{\text{Eris}}{\vdash \{\lightning(\varepsilon) * \Phi'_{\mathsf{det}}\}\, e_2\, \{\lightning(\varepsilon') * \Psi_{\mathsf{det}}\}} \qquad \mathsf{det}(e_1)}{\vdash \{\lightning(\varepsilon) * \Phi_{\mathsf{det}}\}\, e_1; e_2\, \{\lightning(\varepsilon') * \Psi_{\mathsf{det}}\}}\ \text{FRAME+SEQ}$$

Summary: Eris

We introduce a new class of resources, and specify their interaction with sampling

$$\frac{\frac{\varepsilon(0)+\cdots+\varepsilon(N)}{N+1} = \varepsilon'}{\vdash \{\lightning(\varepsilon')\}\ \mathsf{rand}\ N\,\{n : \mathbb{N}\ .\ (n \leqslant N) * \lightning(\varepsilon(n))\}}\ \text{RAND}$$

These resources have minimal interaction with standard Iris

$$\frac{\dfrac{\text{Standard Iris}}{\vdash \{\Phi_{\mathsf{det}}\}\, e_1\, \{\Phi'_{\mathsf{det}}\}} \qquad \dfrac{\text{Eris}}{\vdash \{\lightning(\varepsilon) * \Phi'_{\mathsf{det}}\}\, e_2\, \{\lightning(\varepsilon') * \Psi_{\mathsf{det}}\}} \qquad \mathsf{det}(e_1)}{\vdash \{\lightning(\varepsilon) * \Phi_{\mathsf{det}}\}\, e_1; e_2\, \{\lightning(\varepsilon') * \Psi_{\mathsf{det}}\}}\ \text{FRAME+SEQ}$$

And a "probabilistic" adequacy theorem:

Theorem (Adequacy)

If $\phi : \mathit{Val} \to \mathit{Prop}$ *and* $\{\lightning(\varepsilon)\}\, e\, \{\phi\}$ *then* $\mathsf{Pr}_{\mathsf{exec}(e)}[\neg\phi] \leqslant \varepsilon$.

The bigger picture

We introduce a new class of resources, and specify their interaction with sampling

$$\frac{\text{Premises}}{\vdash \{\mathsf{Prob.Res}_1\}\ \mathsf{rand}\ N\,\{\mathsf{Prob.Res}_2\}}$$

These resources have minimal interaction with standard Iris

$$\frac{\dfrac{\text{Standard Iris}}{\vdash \{\Phi_{\mathsf{det}}\}\, e_1\, \{\Phi'_{\mathsf{det}}\}} \quad \dfrac{\cdots}{\vdash \{\mathsf{Prob.Res}_1 * \Phi'_{\mathsf{det}}\}\, e'\, \{\mathsf{Prob.Res}_2 * \Psi_{\mathsf{det}}\}} \qquad \mathsf{det}(e_1)}{\vdash \{\mathsf{Prob.Res}_1 * \Phi_{\mathsf{det}}\}\, e_1; e_2\, \{\mathsf{Prob.Res}_2 * \Psi_{\mathsf{det}}\}}\ \text{FRAME+SEQ}$$

And a generalized adequacy theorem:

Theorem (Adequacy)

If $\{\mathsf{Prob.Res.}_1 * \Phi_{\mathsf{det}}\}\, e\, \{\mathsf{Prob.Res.}_2 * \Psi_{\mathsf{det}}\}$ *is derivable, then we have [probabilistic property of the program execution]*

Expected runtime

We reintroduce cost credits $\$(n)$, used in Iris to reason about running time.

The logic is parametrized by a *cost model* (accounts for running time, entropy, etc.) Each operation has own associated cost, e.g.

$$\frac{}{\vdash \{\ell \mapsto v * \$(\mathsf{c}_{\mathsf{load}})\} \; !\,\ell \; \{w \,.\, w = v * \ell \mapsto v\}}\;\text{LOAD}$$

Cost credits can be distributed in sampling instructions, same as error credits:

$$\frac{\frac{T(0)+\cdots+T(N)}{N+1} = t}{\vdash \{\$(t) * \$(\mathsf{c}_{\mathsf{rand}})\} \; \mathsf{rand}\; N \; \{n : \mathbb{N} \,.\, (n \leqslant N) * \$(T(n))\}}\;\text{RAND}$$

Theorem (Adequacy for cost)

If $\{\$(n)\}\; e\; \{\mathsf{True}\}$ *then the expected cost of* e *is at most* n

Proving probabilistic program equivalence

$$\text{eager} \triangleq \begin{array}{l}\text{let } b = \text{flip in}\\ \lambda\_.\ b\end{array}$$

$$\text{lazy} \triangleq \begin{array}{l}\text{let } r = \text{ref(None) in}\\ \lambda\_.\ \text{match } !r \text{ with}\\ \qquad \text{Some}(b) \Rightarrow b\\ \qquad \mid \text{None} \Rightarrow \text{let } b = \text{flip in}\\ \qquad\qquad\qquad\quad r \leftarrow \text{Some}(b);\\ \qquad\qquad\qquad\quad b\\ \qquad \text{end}\end{array}$$

Proving probabilistic program equivalence

$$\mathsf{eager} \triangleq \begin{array}{l}\mathsf{let}\ b = \mathsf{flip}\ \mathsf{in} \\ \lambda\,\_.\ b\end{array}$$

$$\mathsf{lazy} \triangleq \begin{array}{l}\mathsf{let}\ r = \mathsf{ref}(\mathsf{None})\ \mathsf{in} \\ \lambda\,\_.\ \mathsf{match}\ !r\ \mathsf{with} \\ \qquad \mathsf{Some}(b) \Rightarrow b \\ \qquad |\ \mathsf{None} \Rightarrow \mathsf{let}\ b = \mathsf{flip}\ \mathsf{in} \\ \qquad\qquad\qquad r \leftarrow \mathsf{Some}(b); \\ \qquad\qquad\qquad b \\ \quad\ \mathsf{end}\end{array}$$

How do we prove that they implement the same distribution?

Separation logics for probabilistic program equivalence

Goal: Prove equivalence between two probabilistic programs e_1, e_2. Needs two kinds of resource:

First, a resource $\mathsf{spec}(e_2)$ to track the 2nd program. It can be executed independently:

$$\frac{\{\mathsf{spec}(w) * \ell \mapsto_s w\}\, e\, \{v.\Phi\}}{\{\mathsf{spec}(!\,\ell) * \ell \mapsto_s w\}\, e\, \{v.\Phi\}}\ \text{LD-R}$$

Specs interact with the 1st program through coupling

$$\frac{\forall n \leqslant N.\, \{\mathsf{spec}(n)\}\, n\, \{\Phi\}}{\{\mathsf{spec}(\mathsf{rand}\ N)\}\ \mathsf{rand}\ N\, \{\Phi\}}\ \text{CPL-RND}$$

Separation logics for probabilistic program equivalence

Second, a *tape* resource, that allows us to generate randomness asynchronously

$$\frac{\forall n. \left\{ n < N * \iota \hookrightarrow^{N} \vec{n} \cdot n \right\} e \, \{v. \Phi\}}{\left\{ \iota \hookrightarrow^{N} \vec{n} \right\} e \, \{v. \Phi\}} \qquad \frac{}{\left\{ \iota \hookrightarrow^{N} n \cdot \vec{n} \right\} \ \mathsf{rand} \, N \, \iota \, \{v. v = n * \iota \hookrightarrow \vec{n}\}}$$

Tapes can also be populated via coupling

$$\frac{\forall n \leqslant N. \left\{ \iota \hookrightarrow_{s}^{N} \vec{n} \cdot n \right\} n \, \{\Phi\}}{\left\{ \iota \hookrightarrow_{s}^{N} \vec{n} \right\} \ \mathsf{rand} \ N \, \{\Phi\}} \ \text{CPL-RND-TP}$$

This turns reasoning about probabilistic choice into reasoning about state.

Separation logics for probabilistic program equivalence

The adequacy theorem gives us a way to reason about equivalence

Theorem (Adequacy for equality)

If $\{\mathsf{spec}(e')\}\, e\, \{v.\exists v'.v = v' * \mathsf{spec}(v')\}$ *then, for all* w, $\mathsf{Pr}[e \Downarrow w] \leqslant \mathsf{Pr}[e' \Downarrow w]$.

Separation logics for probabilistic program equivalence

The adequacy theorem gives us a way to reason about equivalence

Theorem (Adequacy for equality)

If $\{\mathsf{spec}(e')\}\, e\, \{v.\exists v'.v = v' * \mathsf{spec}(v')\}$ *then, for all* w, $\mathsf{Pr}[e \Downarrow w] \leqslant \mathsf{Pr}[e' \Downarrow w]$.

In other words, to show that e and e' implement the same distribution we prove both:

$$\left\{\mathsf{spec}(e')\right\}\, e\, \left\{v.\exists v'.v = v' * \mathsf{spec}(v')\right\}$$

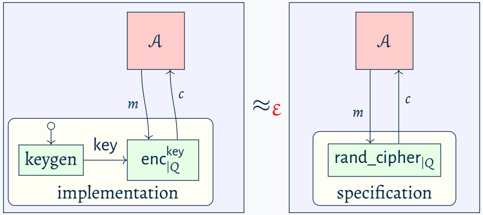
$$\{\mathsf{spec}(e)\}\, e'\, \left\{v.\exists v'.v = v' * \mathsf{spec}(v')\right\}$$

Approximate couplings

- By adding error credits to the previous setup we can implement *approximate couplings*

$$\frac{N \leqslant M \qquad \forall n \leqslant N.\,\{\mathsf{spec}(n)\}\; n \;\{v.\Phi\}}{\{\lightning\left(\frac{M-N}{M+1}\right) * \mathsf{spec}(\mathsf{rand}\ M)\}\ \mathsf{rand}\ N\,\{v.\Phi\}}\ \text{ACPL-RND}$$

- We can reason about convergence by using error credits and taking limits
- Applications to security and verification of probabilistic data structures



Concurrent Probabilistic Programs

What if we want multiple clients to access the Bloom filter concurrently?

In ongoing work, we are extending these concepts to concurrent programs.

$$\frac{\{\Phi_1\}\, e_1\, \{v_1.\Psi_1\} \qquad \{\Phi_2\}\, e_2\, \{v_2.\Psi_2\}}{\{\Phi_1 * \Phi_2\}\, (e_1 ||| e_2)\, \{(v_1, v_2).\Psi_1 * \Psi_2\}}\ \text{PAR}$$

Challenges:

- Interaction between sampling and schedulers (non-determinism)
- Adapting Iris idioms to probabilistic setting (invariants, ghost state)
- New concept of *randomized logical atomicity*[ICFP '25]

Conclusions

- Separation logic: a lightweight, expressive approach to probabilistic reasoning
- By isolating probabilistic reasoning to sampling statements, we can retain all specifications of deterministic programs and compatibility with standard Iris
- The approach is applicable to multiple scenarios, and scales to large programs:
  - Error credits: Bounds on error probabilities
  - Tapes + Specs: Program equivalence, termination preserving refinement
  - Cost credits: Bounds on expected cost/runtime

Conclusions

- Separation logic: a lightweight, expressive approach to probabilistic reasoning
- By isolating probabilistic reasoning to sampling statements, we can retain all specifications of deterministic programs and compatibility with standard Iris
- The approach is applicable to multiple scenarios, and scales to large programs:
  - Error credits: Bounds on error probabilities
  - Tapes + Specs: Program equivalence, termination preserving refinement
  - Cost credits: Bounds on expected cost/runtime

Code: https://github.com/logsem/clutch

Contact: alejandro@cs.au.dk gregersen@cispa.de philipp@haselwarter.org