Coneris:

Modular Reasoning about Error Bounds for Concurrent Probabilistic Programs

@ICFP 2025

Kwing Hei Li[1] Alejandro Aguirre[1] Simon Gregersen[2]
Philipp Haselwarter[1] Joseph Tassarotti[2] Lars Birkedal[1]

[1] *Aarhus University*, [2] *New York University*

A simple probability problem (based on real events)

Heili is applying to two different companies for a research internship.

A simple probability problem (based on real events)

Heili is applying to two different companies for a research internship.

Each company *independently* has a probability of $3/4$ of giving me an offer.

A simple probability problem (based on real events)

Heili is applying to two different companies for a research internship.

Each company *independently* has a probability of $3/4$ of giving me an offer.

What is the probability I will not have a single internship offer 😭 ?

A sequential probabilistic program

$$\begin{aligned}
&\text{let } l = \text{ref } 0 \text{ in} \\
&l \leftarrow (!\, l + \text{coin}_{3/4}); \\
&l \leftarrow (!\, l + \text{coin}_{3/4}); \\
&\text{let } x = !\, l \text{ in} \\
&\text{assert}(x > 0) \qquad\qquad //\text{fails if both coin}_{3/4} \text{ returns } 0
\end{aligned}$$

coin_p returns 1 with probability p and 0 with probability $1 - p$.

A sequential probabilistic program

$$
\begin{aligned}
&\mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in}\\
&l \leftarrow (!\,l + \mathsf{coin}_{3/4});\\
&l \leftarrow (!\,l + \mathsf{coin}_{3/4});\\
&\mathsf{let}\ x = !\,l\ \mathsf{in}\\
&\mathsf{assert}(x > 0) \qquad\qquad //\text{fails if both } \mathsf{coin}_{3/4} \text{ returns } 0
\end{aligned}
$$

coin_p returns 1 with probability p and 0 with probability $1 - p$.

Aim: show *twoAdd* **crashes with probability at most** $1/16$

Previous work: Eris (ICFP 24)

- *Eris*: a separation logic for proving error bounds of sequential probabilistic programs

Previous work: Eris (ICFP 24)

- *Eris*: a separation logic for proving error bounds of sequential probabilistic programs
- **KEY IDEA: internalize error as a separation logic resource, aka *error credit***

Previous work: Eris (ICFP 24)

- *Eris*: a separation logic for proving error bounds of sequential probabilistic programs
- **KEY IDEA: internalize error as a separation logic resource, aka *error credit***
- $⚡(\varepsilon)$ asserts ownership of ε error credits, with $\varepsilon \in [0, 1]$

Previous work: Eris (ICFP 24)

- *Eris*: a separation logic for proving error bounds of sequential probabilistic programs
- **KEY IDEA: internalize error as a separation logic resource, aka *error credit***
- $\lightning(\varepsilon)$ asserts ownership of ε error credits, with $\varepsilon \in [0, 1]$

Theorem (Adequacy of Eris)

If $\{\lightning(\varepsilon)\}\, e\, \{v.\phi(v)\}$ *then* $\Pr[e \Downarrow v \wedge v \notin \phi] \leqslant \varepsilon.$

Eris rules

$$\lightning(\varepsilon_1) * \lightning(\varepsilon_2) \dashv\vdash \lightning(\varepsilon_1 + \varepsilon_2)$$

Eris rules

$$↯(\varepsilon_1) * ↯(\varepsilon_2) \dashv\vdash ↯(\varepsilon_1 + \varepsilon_2)$$

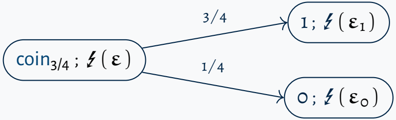
$$\{↯(1)\}\, e\, \{\phi\} \qquad\qquad (\Pr[e \Downarrow v \land v \notin \phi] \leqslant 1)$$

Eris rules

$$\lightning(\varepsilon_1) * \lightning(\varepsilon_2) \dashv\vdash \lightning(\varepsilon_1 + \varepsilon_2)$$

$$\{\lightning(1)\}\, e\, \{\phi\} \qquad (\Pr[e \Downarrow v \wedge v \notin \phi] \leqslant 1)$$

$$\frac{3/4 \cdot \varepsilon_1 + 1/4 \cdot \varepsilon_0 \leqslant \varepsilon}{\vdash \{\lightning(\varepsilon)\}\ \mathsf{coin}_{3/4}\ \{n\,.\ \lightning(\varepsilon_n)\}}\ \textsc{ht-coin-exp}$$



$$3/4 \cdot \varepsilon_1 + 1/4 \cdot \varepsilon_0 \leqslant \varepsilon$$

Eris in action

$$
\begin{array}{l}
\{\lightning(1/16)\} \\
\qquad \mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in} \\
\qquad l \leftarrow (!\,l + \mathsf{coin}_{3/4}); \\
\qquad l \leftarrow (!\,l + \mathsf{coin}_{3/4}); \\
\qquad \mathsf{let}\ x = !\,l\ \mathsf{in} \\
\qquad \mathsf{assert}(x > 0) \\
\{\mathsf{True}\}
\end{array}
$$

By adequacy, the probability of the program crashes is at most 1/16.

Eris in action

$$\{\lightning(1/16) \ast l \mapsto 0\}$$

$$\begin{aligned}&l \leftarrow (!\,l + \mathsf{coin}_{3/4});\\ &l \leftarrow (!\,l + \mathsf{coin}_{3/4});\\ &\mathsf{let}\; x = !\,l \;\mathsf{in}\\ &\mathsf{assert}(x > 0)\end{aligned}$$

$$\{\mathrm{True}\}$$

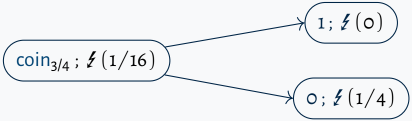
Allocate reference

Eris in action

$$\left\{ \begin{array}{l} \lightning(\text{if } x = 0 \text{ then } 1/4 \text{ else } 0) \;\; * \\ l \mapsto 0 \end{array} \right\}$$

$$\begin{array}{l} l \leftarrow (!\,l + x); \\ l \leftarrow (!\,l + \mathsf{coin}_{3/4}); \\ \text{let } x = !\,l \text{ in} \\ \mathsf{assert}(x > 0) \end{array}$$

$\{\text{True}\}$



$$3/4 \cdot 0 + 1/4 \cdot 1/4 \leqslant \frac{1}{16}$$

Eris in action

$$\{ \lightning(1/4) \ast l \mapsto 0 \}$$

$$\begin{aligned} & l \leftarrow (!\,l + 0); \\ & l \leftarrow (!\,l + \mathsf{coin}_{3/4}); \\ & \mathsf{let}\, x = !\,l\, \mathsf{in} \\ & \mathsf{assert}(x > 0) \end{aligned}$$

$$\{\mathsf{True}\}$$

We continue with the case $x = 0$, otherwise it is trivial

Eris in action

$$\{ \lightning(1/4) \ast l \mapsto 0 \}$$

$$\begin{array}{l} \quad l \leftarrow (!\,l + \mathsf{coin}_{3/4}); \\ \quad \mathsf{let}\ x = !\,l\ \mathsf{in} \\ \quad \mathsf{assert}(x > 0) \end{array}$$

$$\{\mathsf{True}\}$$

More steps...

Eris in action

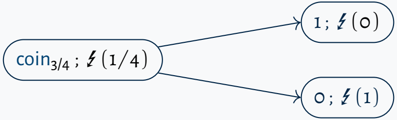
$$\{\lightning(\mathsf{if}\ x = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0) \ * l \mapsto 0\}$$

$$l \leftarrow (!\,l + x);$$

$$\mathsf{let}\ x = !\,l\ \mathsf{in}$$

$$\mathsf{assert}(x > 0)$$

$$\{\mathsf{True}\}$$



$$3/4 \cdot 0 + 1/4 \cdot 1 \leqslant \frac{1}{4}$$

Eris in action

$$\{\lightning\,(\mathsf{if}\ x = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0) \ * \ l \mapsto 0\}$$

$$\begin{array}{l} \quad l \leftarrow (!\,l + x); \\ \quad \mathsf{let}\ x = !\,l\ \mathsf{in} \\ \quad \mathsf{assert}(x > 0) \end{array}$$

$$\{\mathsf{True}\}$$

$$\begin{tikzcd} & \boxed{1 \,;\, \lightning\,(0)} \\ \boxed{\mathsf{coin}_{3/4} \,;\, \lightning\,(1/4)} \arrow[ur] \arrow[dr] & \\ & \boxed{0 \,;\, \lightning\,(1)} \arrow[r, Rightarrow] & \checkmark \end{tikzcd}$$

$$3/4 \cdot 0 + 1/4 \cdot 1 \leqslant \frac{1}{4}$$

Eris in action

$$\{\lightning(0) \ast l \mapsto 0\}$$

$$l \leftarrow (!\,l + 1);$$
$$\mathsf{let}\; x = !\,l\; \mathsf{in}$$
$$\mathsf{assert}(x > 0)$$

$$\{\mathsf{True}\}$$

We sampled a 1.

Eris in action

$$\{\lightning(0) \ast l \mapsto 1\}$$

$$\quad \mathsf{assert}(1 > 0)$$

$$\{\mathsf{True}\}$$

And the assert goes through!

We can reason about error bounds of sequential **probabilistic** programs.

Nobody applies for internships sequentially.

A concurrent probabilistic program

$$
\begin{aligned}
&\mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in} \\
&(\mathsf{faa}\ l\ (\mathsf{coin}_{3/4})\ |||\ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ; \\
&\mathsf{let}\ x = !\, l\ \mathsf{in} \\
&\mathsf{assert}(x > 0);
\end{aligned}
$$

A concurrent probabilistic program

$$
\begin{aligned}
&\mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in} \\
&(\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \ ||| \ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ; \\
&\mathsf{let}\ x = !\, l\ \mathsf{in} \\
&\mathsf{assert}(x > 0);
\end{aligned}
$$

$\mathsf{faa}\ l\ x$ reads from reference l and increments it by x *atomically*

Introducing CONERIS

Coneris: a new CSL inheriting error credits for error bound reasoning

Introducing CONERIS

Coneris: a new CSL inheriting error credits for error bound reasoning

Error credits and concurrency rules are the same, e.g. HT-COIN-EXP, parallel composition.

Introducing CONERIS

Coneris: a new CSL inheriting error credits for error bound reasoning

Error credits and concurrency rules are the same, e.g. HT-COIN-EXP, parallel composition.

Operational semantics of language extended to thread pools;
decision of which thread to step is decided by a scheduler

Introducing CONERIS

Coneris: a new CSL inheriting error credits for error bound reasoning

Error credits and concurrency rules are the same, e.g. HT-COIN-EXP, parallel composition.

Operational semantics of language extended to thread pools;
decision of which thread to step is decided by a scheduler

Theorem (Adequacy of Coneris)

If $\{\lightning(\varepsilon)\}\, e\, \{v.\phi(v)\}$ *then* ***for all possible schedulers*** $\mathfrak{s}$,
$\Pr[e \Downarrow_{\mathfrak{s}} v \wedge v \notin \phi] \leq \varepsilon$

First attempt in verifying *conTwoAdd*

$\{\lightning(1/16)\}$

$\quad \mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in}$
$\quad (\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \mid\mid\mid \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ;$
$\quad \mathsf{let}\ x = !\, l\ \mathsf{in}$
$\quad \mathsf{assert}(x > 0)$

$\{\mathsf{True}\}$

By adequacy, for all possible interleavings, probability of the program crashing is at most 1/16.

First attempt in verifying *conTwoAdd*

$$\{↯(1/16) \ast l \mapsto 0\}$$
$$(\mathsf{faa}\, l\, (\mathsf{coin}_{3/4}) \mathbin{|||} \mathsf{faa}\, l\, (\mathsf{coin}_{3/4}))\,;$$
$$\mathsf{let}\, x = {!}\, l\, \mathsf{in}$$
$$\mathsf{assert}(x > 0)$$
$$\{\mathsf{True}\}$$

HT-PAR-COMP

$$\frac{\{P_1\}\, e_1\, \{v_1.\, Q_1\, v_1\} \qquad \{P_2\}\, e_2\, \{v_2.\, Q_2\, v_2\}}{\{P_1 \ast P_2\}\, e_1 \mathbin{|||} e_2\, \{(v_1, v_2).\, Q_1\, v_1 \ast Q_2\, v_2\}}$$

First attempt in verifying *conTwoAdd*

$\{ \lightning(1/16) \ * l \mapsto 0 \}$

$\quad (\mathsf{faa}\ l\ (\mathsf{coin}_{3/4})\ |||\ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ;$
$\quad \mathsf{let}\ x = !\, l\ \mathsf{in}$
$\quad \mathsf{assert}(x > 0)$

$\{\mathsf{True}\}$

$$\textsc{ht-par-comp}\ \dfrac{\{P_1\}\ e_1\ \{v_1.\ Q_1\ v_1\} \qquad \{P_2\}\ e_2\ \{v_2.\ Q_2\ v_2\}}{\{P_1 * P_2\}\ e_1\ |||\ e_2\ \{(v_1, v_2).\ Q_1\ v_1 * Q_2\ v_2\}}$$

1. We need to share the $l \mapsto 0$ resource between the two threads

First attempt in verifying *conTwoAdd*

$\{\lightning(1/16) \ast l \mapsto 0\}$

$(\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \mid\mid\mid \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ;$
$\mathsf{let}\ x = !\,l\ \mathsf{in}$
$\mathsf{assert}(x > 0)$

$\{\mathsf{True}\}$

HT-PAR-COMP

$$\frac{\{P_1\}\, e_1\, \{v_1.\, Q_1\, v_1\} \qquad \{P_2\}\, e_2\, \{v_2.\, Q_2\, v_2\}}{\{P_1 \ast P_2\}\, e_1 \mid\mid\mid e_2\, \{(v_1, v_2).\, Q_1\, v_1 \ast Q_2\, v_2\}}$$

1. We need to share the $l \mapsto 0$ resource between the two threads
2. Splitting $\lightning(1/16)$ into $\lightning(1/32) \ast \lightning(1/32)$ is not enough for each thread to avoid sampling 0

Invariant

Invariants allow us to share resources between threads

Invariant

Invariants allow us to share resources between threads

$$\frac{\textsc{ht-inv-open} \quad e \text{ atomic} \qquad \{I * P\}\, e\, \{I * Q\}}{\{\boxed{I} * P\}\, e\, \{Q\}}$$

We can access resources in invariants across atomic steps ...

Invariant

Invariants allow us to share resources between threads

$$\frac{\textsc{ht-inv-open} \quad e \text{ atomic} \qquad \{I * P\}\, e\, \{I * Q\}}{\left\{\boxed{I} * P\right\}\, e\, \{Q\}}$$

We can access resources in invariants across atomic steps ...

and we have to reestablish the invariant at the end of the atomic step

Invariant

Invariants allow us to share resources between threads

$$\frac{\overset{\text{HT-INV-OPEN}}{e \text{ atomic}} \qquad \{I * P\}\, e\, \{I * Q\}}{\left\{\boxed{I} * P\right\} e\, \{Q\}}$$

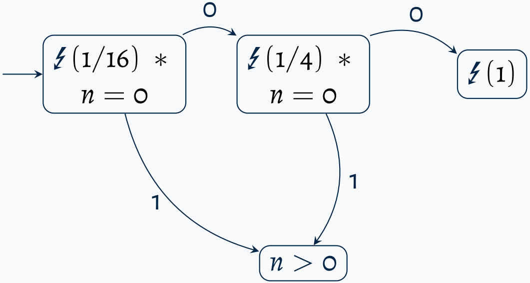
We can access resources in invariants across atomic steps ...

and we have to reestablish the invariant at the end of the atomic step

Key idea: we store both error credit and reference resource in an *invariant*

Coming up with an invariant

$I \triangleq \exists n. l \mapsto n \ast$



$\{\boxed{I} \ast \ldots\}$

$\quad(\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \mid\mid\mid \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ;$

$\quad\mathsf{let}\ x = !\ l\ \mathsf{in}$

$\quad\mathsf{assert}(x > 0)$

$\{\mathsf{True}\}$

We can reason about error bounds of ~~sequential~~ **concurrent** probabilistic programs.

Now let's refactor stuff into a randomized concurrent counter

$$
\begin{array}{l}
\mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in} \\
(\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \ ||| \ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ; \\
\mathsf{let}\ x = !\,l\ \mathsf{in} \\
\mathsf{assert}(x > 0)
\end{array}
$$

Now let's refactor stuff into a randomized concurrent counter

$$\begin{array}{l} \mathsf{let}\ l = \mathsf{ref}\ 0\ \mathsf{in} \\ (\mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \mid\mid\mid \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}))\ ; \\ \mathsf{let}\ x = !\, l\ \mathsf{in} \\ \mathsf{assert}(x > 0) \end{array} \quad \Rightarrow \quad \begin{array}{l} \begin{aligned} \mathit{create} &\triangleq \lambda\_.\ \mathsf{ref}\ 0 \\ \mathit{read} &\triangleq \lambda l.\ !\, l \\ \mathit{incr} &\triangleq \lambda l.\ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4}) \end{aligned} \\ \\ \mathsf{let}\ l = \mathit{create}\ ()\ \mathsf{in} \\ (\mathit{incr}\ l \mid\mid\mid \mathit{incr}\ l)\ ; \\ \mathsf{let}\ x = \mathit{read}\ l\ \mathsf{in} \\ \mathsf{assert}(x > 0) \end{array}$$

Problem 1: interaction with invariants

$$\left\{ \boxed{I} \ast \ldots \right\}$$

$$\quad \mathit{incr}\, l \triangleq (\ \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})\ )\ l$$

$$\{\ldots\}$$

Problem 1: interaction with invariants

$$\left\{ \boxed{I} \;*\; \ldots \right\}$$

$$\quad \mathit{incr}\, l \triangleq (\lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4}))\ l$$

$$\{\ldots\}$$

Problem 1: interaction with invariants

$$\left\{ \boxed{I} \ast \ \ldots \right\}$$

$$\mathit{incr}\, l \triangleq (\ \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})\ )\ l$$

$$\{\ldots\}$$

Problem 1: interaction with invariants

$$\{\boxed{I} * \ldots\}$$
$$\quad \mathit{incr}\, l \triangleq (\lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4}))\ l$$
$$\{\ldots\}$$

HT-INV-OPEN

$$\frac{e \text{ atomic} \qquad \{I * P\}\, e\, \{I * Q\}}{\{\boxed{I} * P\}\, e\, \{Q\}}$$

Problem 1: interaction with invariants

$$\{\boxed{I} * \ldots\}$$
$$\quad \mathit{incr}\, l \triangleq (\ \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})\ )\ l$$
$$\{\ldots\}$$

HT-INV-OPEN

$$\frac{e \text{ atomic} \qquad \{I * P\}\, e\, \{I * Q\}}{\{\boxed{I} * P\}\, e\, \{Q\}}$$

Challenge: specification should be expressive enough to open invariants (twice!)

Linearizability

A standard notion of correctness for concurrent data structure is linearizability:

Linearizability

A standard notion of correctness for concurrent data structure is linearizability:

Intuitively, all operations appear to take place atomically.

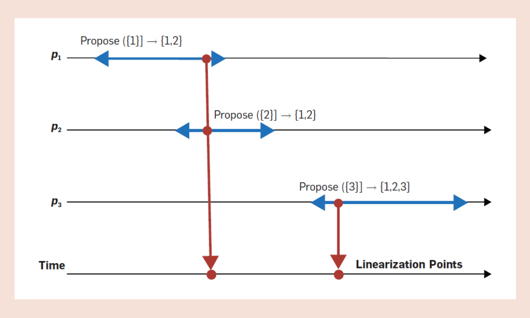


Figure from "A Linearizability-based Hierarchy for Concurrent Specifications"

Logical atomicity

Concurrent separation logics internalize the notion of *linearizability* through *logical atomicity*

Logical atomicity

Concurrent separation logics internalize the notion of *linearizability* through *logical atomicity*

In Iris, logical atomicity can be expressed via HOCAP specifications

Logical atomicity

Concurrent separation logics internalize the notion of *linearizability* through *logical atomicity*

In Iris, logical atomicity can be expressed via HOCAP specifications

We parametrize preconditions with view shifts written with $\Rrightarrow P$;
they describes how logical state of a module changes at linearization point

Logical atomicity

Concurrent separation logics internalize the notion of *linearizability* through *logical atomicity*

In Iris, logical atomicity can be expressed via HOCAP specifications

We parametrize preconditions with view shifts written with $\Rrightarrow P$;
they describes how logical state of a module changes at linearization point

But this is not enough when we also have probability...

Problem 2: different implementations

$$incr_1 \triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})$$

Problem 2: different implementations

$$incr_1 \triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})$$

$$incr_2 \triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{if}\ \mathsf{coin}_{1/2} = 1\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{coin}_{1/2})$$

Problem 2: different implementations

$$incr_1 \triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{coin}_{3/4})$$

$$incr_2 \triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{if}\ \mathsf{coin}_{1/2} = 1\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{coin}_{1/2})$$

$$\begin{aligned} incr_3 \triangleq{} & \\ & \mathsf{rec}\, f\, l = \mathsf{let}\, x = \mathsf{unif}\{0, \ldots, 7\}\ \mathsf{in} \\ & \quad \mathsf{if}\, x < 4\ \mathsf{then}\ (\mathsf{if}\, x > 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0)\ \mathsf{else}\, f l \end{aligned}$$

Problem 2: different implementations

$$incr_1 \triangleq \lambda l.\ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4})$$

$$incr_2 \triangleq \lambda l.\ \mathsf{faa}\ l\ (\mathsf{if}\ \mathsf{coin}_{1/2} = 1\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{coin}_{1/2})$$

$$\begin{aligned}
&incr_3 \triangleq \\
&\quad \mathsf{rec}\ f\ l = \mathsf{let}\ x = \mathsf{unif}\{0, \ldots, 7\}\ \mathsf{in} \\
&\qquad \mathsf{if}\ x < 4\ \mathsf{then}\ (\mathsf{if}\ x > 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0)\ \mathsf{else}\ f l
\end{aligned}$$

Even though the randomness operations may not be atomic, they appear to be *logically atomic*

Problem 2: different implementations

$$incr_1 \triangleq \lambda l.\ \mathsf{faa}\ l\ (\mathsf{coin}_{3/4})$$

$$incr_2 \triangleq \lambda l.\ \mathsf{faa}\ l\ (\mathsf{if}\ \mathsf{coin}_{1/2} = 1\ \mathsf{then}\ 1\ \mathsf{else}\ \mathsf{coin}_{1/2})$$

$$\begin{aligned}
&incr_3 \triangleq \\
&\quad \mathsf{rec}\ f\ l = \mathsf{let}\ x = \mathsf{unif}\{0, \ldots, 7\}\ \mathsf{in} \\
&\qquad \mathsf{if}\ x < 4\ \mathsf{then}\ (\mathsf{if}\ x > 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0)\ \mathsf{else}\ f\, l
\end{aligned}$$

Even though the randomness operations may not be atomic, they appear to be *logically atomic*

Challenge: our specification should be satisfied by all three implementations above

We need to capture “randomized logical atomicity”

Key idea: We capture a notion of *randomized logical atomicity*

We need to capture "randomized logical atomicity"

Key idea: We capture a notion of *randomized logical atomicity*

Intuitively describes how modules commit to probabilistic choices in a logically atomic manner

We need to capture “randomized logical atomicity”

Key idea: We capture a notion of *randomized logical atomicity*

Intuitively describes how modules commit to probabilistic choices in a logically atomic manner

Here in Coneris, we capture this notion with a novel probabilistic update modality $⩪ P$;
it allows you to update error credits in addition to opening invariants

Case study

We implement a thread-safe idealized hash function:

$$\{\mathsf{True}\}\ \mathsf{hashInit}\ ()\ \left\{h.\ \mathop{\Large\ast}_{k \in K} \mathsf{hashKey}\ h\ k -\right\}$$

Case study

We implement a thread-safe idealized hash function:

$$\{\mathsf{True}\}\ \mathsf{hashInit}\ ()\ \left\{h.\ \mathop{\Huge\ast}_{k \in K} \mathsf{hashKey}\ h\ k\ -\right\}$$

$$\begin{aligned}&\mathsf{hashKey}\ h\ k\ - \ \twoheadrightarrow\ ↯(\varepsilon)\ \twoheadrightarrow \\ &\rightsquigarrow \exists v \in V.\ \mathsf{hashKey}\ h\ k\ v * (v \in X * ↯(\varepsilon_1)) \vee (v \notin X * ↯(\varepsilon_0))\end{aligned}$$

$$(\text{where } \varepsilon_1 \cdot |X| + \varepsilon_0 \cdot (|V| - |X|) \leqslant \varepsilon \cdot |V|)$$

Case study

We implement a thread-safe idealized hash function:

$$\{\mathsf{True}\}\ \mathsf{hashInit}\ ()\ \left\{h.\ \mathop{\Huge *}_{k \in K} \mathsf{hashKey}\ h\ k\ -\right\}$$

$$\mathsf{hashKey}\ h\ k\ - \ \mathrel{-\!\!*}\ ↯(\varepsilon)\ \mathrel{-\!\!*}$$
$$\Rrightarrow \exists v \in V.\ \mathsf{hashKey}\ h\ k\ v * (v \in X * ↯(\varepsilon_1)) \vee (v \notin X * ↯(\varepsilon_0))$$

$$(\text{where } \varepsilon_1 \cdot |X| + \varepsilon_0 \cdot (|V| - |X|) \leqslant \varepsilon \cdot |V|)$$

$$\{\mathsf{hashKey}\ h\ k\ v\}\ h\ k\ \{w.\ w = v\}$$

Case study

We implement a thread-safe idealized hash function:

$$\{\mathsf{True}\}\ \mathsf{hashInit}\ ()\ \left\{h.\ \mathop{\ast\!\!\!\ast}_{k \in K} \mathsf{hashKey}\ h\ k\ -\right\}$$

$$\begin{aligned}&\mathsf{hashKey}\ h\ k\ - \ \mathrel{-\!\!\ast}\ ↯(\varepsilon)\ \mathrel{-\!\!\ast}\\ &\rightsquigarrow \exists v \in V.\ \mathsf{hashKey}\ h\ k\ v \ast (v \in X \ast ↯(\varepsilon_1)) \lor (v \notin X \ast ↯(\varepsilon_0))\end{aligned}$$

$$(\text{where } \varepsilon_1 \cdot |X| + \varepsilon_0 \cdot (|V| - |X|) \leqslant \varepsilon \cdot |V|)$$

$$\{\mathsf{hashKey}\ h\ k\ v\}\ h\ k\ \{w.\ w = v\}$$

We use this to implement a concurrent Bloom filter and
we prove tight bound on probability of false positives (new result)

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs modularly

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs modularly by capturing randomized logical atomicity.

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs modularly by capturing randomized logical atomicity.

Other stuff in the paper:

- Details of the specifications and proofs (proving correctness of modules and clients, logic)

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs modularly by capturing randomized logical atomicity.

Other stuff in the paper:

- Details of the specifications and proofs (proving correctness of modules and clients, logic)
- Other examples: lazy sampler, concurrent amortized collision-free hash

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs
modularly by capturing randomized logical atomicity.

Other stuff in the paper:

- Details of the specifications and proofs (proving correctness of modules and clients, logic)
- Other examples: lazy sampler, concurrent amortized collision-free hash

P.S. If you have any research internship opportunity, please find me :)

Take home message

Motto: We can reason about error bounds of concurrent probabilistic programs
modularly by capturing randomized logical atomicity.

Other stuff in the paper:

- Details of the specifications and proofs (proving correctness of modules and clients, logic)
- Other examples: lazy sampler, concurrent amortized collision-free hash

P.S. If you have any research internship opportunity, please find me :)

P.P.S. My advisor, Lars, is recruiting interns / PhD students / postdocs :)

Presampling tapes I

$$\sigma \in \mathit{State} \triangleq (\mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}) \times (\mathit{Label} \xrightarrow{\text{fin}} \mathit{Tape})$$
$$t \in \mathit{Tape} \triangleq \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathbb{N}^{*}_{\leqslant N}\}$$

Presampling tapes I

$$\sigma \in \mathit{State} \triangleq (\mathit{Loc} \xrightharpoonup{\text{fin}} \mathit{Val}) \times (\mathit{Label} \xrightharpoonup{\text{fin}} \mathit{Tape})$$
$$t \in \mathit{Tape} \triangleq \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathbb{N}^{*}_{\leqslant N}\}$$

$$\mathrm{step}(\mathrm{tape}\ N, \sigma) = \mathrm{ret}(\kappa, \sigma[\kappa := (N, \epsilon)], []) \quad \text{(where } \kappa \text{ is fresh w.r.t. } \sigma\text{)}$$
$$\mathrm{step}(\mathrm{rand}\ \kappa\ N, \sigma) = \lambda(n, \sigma, []) \,.\ \tfrac{1}{N+1} \quad \text{if } \sigma[\kappa] = (N, \epsilon) \wedge n \in \{0, \ldots, N\} \quad \text{and} \quad 0 \text{ otherwise}$$

Presampling tapes I

$$\sigma \in \mathit{State} \triangleq (\mathit{Loc} \xrightharpoonup{\text{fin}} \mathit{Val}) \times (\mathit{Label} \xrightharpoonup{\text{fin}} \mathit{Tape})$$

$$t \in \mathit{Tape} \triangleq \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathbb{N}^{*}_{\leqslant N}\}$$

$$\text{step}(\text{tape } N, \sigma) = \text{ret}(\kappa, \sigma[\kappa := (N, \epsilon)], []) \quad \text{(where } \kappa \text{ is fresh w.r.t. } \sigma\text{)}$$

$$\text{step}(\text{rand } \kappa\, N, \sigma) = \lambda(n, \sigma, []) \,.\, \tfrac{1}{N+1} \quad \text{if } \sigma[\kappa] = (N, \epsilon) \wedge n \in \{0, \ldots, N\} \quad \text{and} \quad 0 \text{ otherwise}$$

There are no steps in operational semantics to *write* contents into a tape!

Rewriting randomized concurrent counter module

$$\begin{aligned} \mathit{create} &\triangleq \lambda\_.\ \ \mathsf{ref}\, 0 \\ \mathit{read} &\triangleq \lambda l.\ \ !\, l \\ \mathit{incr} &\triangleq \lambda l.\ \ \mathsf{faa}\, l\, (\mathsf{rand}\, 3) \end{aligned}$$

$$\begin{aligned} \mathit{conTwoAdd} \triangleq\ & \mathsf{let}\, l = \mathit{create}\,()\ \mathsf{in} \\ & (\mathit{incr}\, l \mid\mid\mid \mathit{incr}\, l)\, ; \\ & \mathit{read}\, l \end{aligned}$$

Rewriting randomized concurrent counter module

$$\begin{aligned}
\mathit{create} &\triangleq \lambda_.\ \mathsf{ref}\, 0\\
\mathit{read} &\triangleq \lambda l.\ !\, l\\
\mathit{incr} &\triangleq \lambda l.\ \mathsf{faa}\, l\, (\mathsf{rand}\, 3)
\end{aligned}$$

$$\begin{aligned}
\mathit{conTwoAdd} \triangleq\ & \mathsf{let}\ l = \mathit{create}\, ()\ \mathsf{in}\\
& (\mathit{incr}\, l \mid\mid\mid \mathit{incr}\, l)\, ;\\
& \mathit{read}\, l
\end{aligned}$$

$$\Rightarrow$$

$$\begin{aligned}
\mathit{create} &\triangleq \lambda_.\ \mathsf{ref}\, 0\\
\mathit{read} &\triangleq \lambda l.\ !\, l\\
\mathit{createCtape} &\triangleq \lambda ().\ \mathsf{tape}\, 3\\
\mathit{incr} &\triangleq \lambda l\, \kappa.\ \mathsf{faa}\, l\, (\mathsf{rand}\, \kappa\, 3)
\end{aligned}$$

$$\begin{aligned}
\mathit{conTwoAdd} \triangleq\ & \mathsf{let}\ c = \mathit{create}()\ \mathsf{in}\\
& \left(\begin{array}{l} \mathsf{let}\ \kappa = \mathit{createCtape}()\ \mathsf{in}\\ \mathit{incr}\, c\, \kappa \end{array} \mid\mid\mid \ \ldots \right);\\
& \mathit{read}\, c
\end{aligned}$$

Presampling tapes II

HT-ALLOC-TAPE

$$\frac{}{\{\mathsf{True}\}\ \texttt{tape}\ N\,\{\kappa.\ \kappa \hookrightarrow (N, \epsilon)\}}$$

Presampling tapes II

HT-ALLOC-TAPE

$$\frac{}{\{\mathsf{True}\}\ \mathsf{tape}\ N\,\{\kappa.\,\kappa \hookrightarrow (N, \epsilon)\}}$$

HT-RAND-TAPE

$$\frac{}{\{\kappa \hookrightarrow (N, n \cdot \vec{n})\}\ \mathsf{rand}\ \kappa\ N\,\{x.\,x = n * \kappa \hookrightarrow (N, \vec{n})\}}$$

Presampling tapes II

HT-ALLOC-TAPE

$$\frac{}{\{\mathsf{True}\}\ \mathsf{tape}\ N\,\{\kappa.\ \kappa \hookrightarrow (N, \epsilon)\}}$$

HT-RAND-TAPE

$$\frac{}{\{\kappa \hookrightarrow (N, n \cdot \vec{n})\}\ \mathsf{rand}\ \kappa\, N\,\{x.\, x = n * \kappa \hookrightarrow (N, \vec{n})\}}$$

Wait?! How do you presample onto a tape in the logic?

Presampling tapes II

HT-ALLOC-TAPE

$$\frac{}{\{\mathsf{True}\}\ \mathsf{tape}\ N\,\{\kappa.\,\kappa \hookrightarrow (N, \epsilon)\}}$$

HT-RAND-TAPE

$$\frac{}{\{\kappa \hookrightarrow (N, n \cdot \vec{n})\}\ \mathsf{rand}\,\kappa\,N\,\{x.\,x = n * \kappa \hookrightarrow (N, \vec{n})\}}$$

Wait?! How do you presample onto a tape in the logic?

We do it with the probabilistic update modality!

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}{\Rrightarrow}_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}\!\rightsquigarrow\!\!\!\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

$$\frac{P}{\rightsquigarrow\!\!\!\!\rightsquigarrow_{\mathcal{E}} P}\ \textsc{pupd-ret}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

PUPD-RET
$$\frac{P}{\rightsquigarrow_{\mathcal{E}} P}$$

PUPD-BIND
$$\frac{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P \qquad P \twoheadrightarrow {}_{\mathcal{E}_2}\!\rightsquigarrow_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_3} Q}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}{\mid\!\!\rightsquigarrow}_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

PUPD-RET

$$\frac{P}{{\mid\!\!\rightsquigarrow}_{\mathcal{E}} P}$$

PUPD-BIND

$$\frac{{}_{\mathcal{E}_1}{\mid\!\!\rightsquigarrow}_{\mathcal{E}_2} P \qquad P \mathrel{-\!\!*} {}_{\mathcal{E}_2}{\mid\!\!\rightsquigarrow}_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}{\mid\!\!\rightsquigarrow}_{\mathcal{E}_3} Q}$$

PUPD-FUPD

$$\frac{{}_{\mathcal{E}_1}{\mid\!\!\Rrightarrow}_{\mathcal{E}_2} P}{{}_{\mathcal{E}_1}{\mid\!\!\rightsquigarrow}_{\mathcal{E}_2} P}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}{\rightsquigarrow}_{\mathcal{E}_2}\ P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

$$\textsc{pupd-ret}\quad \frac{P}{{\rightsquigarrow}_{\mathcal{E}}\ P}$$

$$\textsc{pupd-bind}\quad \frac{{}_{\mathcal{E}_1}{\rightsquigarrow}_{\mathcal{E}_2}\ P \qquad P \mathrel{-\!\!*} {}_{\mathcal{E}_2}{\rightsquigarrow}_{\mathcal{E}_3}\ Q}{{}_{\mathcal{E}_1}{\rightsquigarrow}_{\mathcal{E}_3}\ Q}$$

$$\textsc{pupd-fupd}\quad \frac{{}_{\mathcal{E}_1}{\Rrightarrow}_{\mathcal{E}_2}\ P}{{}_{\mathcal{E}_1}{\rightsquigarrow}_{\mathcal{E}_2}\ P}$$

$$\textsc{pupd-presample-exp}\quad \frac{\kappa \hookrightarrow (N, \vec{n}) \qquad ⚡(\varepsilon) \qquad \mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon}{{\rightsquigarrow}_{\mathcal{E}}\ (\exists n.\ \kappa \hookrightarrow (N, \vec{n} \cdot n) * ⚡(\mathcal{F}(n)))}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

PUPD-RET

$$\frac{P}{\rightsquigarrow_{\mathcal{E}} P}$$

PUPD-BIND

$$\frac{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P \qquad P \twoheadrightarrow {}_{\mathcal{E}_2}\!\rightsquigarrow_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_3} Q}$$

PUPD-FUPD

$$\frac{{}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} P}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P}$$

PUPD-PRESAMPLE-EXP

$$\frac{\kappa \hookrightarrow (N, \vec{n}) \qquad \lightning(\varepsilon) \qquad \mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon}{\rightsquigarrow_{\mathcal{E}} (\exists n.\, \kappa \hookrightarrow (N, \vec{n} \cdot n) * \lightning(\mathcal{F}(n)))}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

$$\textsc{pupd-ret}\quad \frac{P}{\rightsquigarrow_{\mathcal{E}} P}$$

$$\textsc{pupd-bind}\quad \frac{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P \qquad P \mathrel{-\!\!*} {}_{\mathcal{E}_2}\!\rightsquigarrow_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_3} Q}$$

$$\textsc{pupd-fupd}\quad \frac{{}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} P}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P}$$

$$\textsc{pupd-presample-exp}\quad \frac{\kappa \hookrightarrow (N, \vec{n}) \qquad \lightning(\varepsilon) \qquad \mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon}{\rightsquigarrow_{\mathcal{E}} (\exists n.\ \kappa \hookrightarrow (N, \vec{n} \cdot n) * \lightning(\mathcal{F}(n)))}$$

Rules of the probabilistic update modality

${}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

PUPD-RET
$$\frac{P}{\rightsquigarrow_{\mathcal{E}} P}$$

PUPD-BIND
$$\frac{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P \qquad P \mathrel{-\!\!*} {}_{\mathcal{E}_2}\!\rightsquigarrow_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_3} Q}$$

PUPD-FUPD
$$\frac{{}_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} P}{{}_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P}$$

PUPD-PRESAMPLE-EXP
$$\frac{\kappa \hookrightarrow (N, \vec{n}) \qquad ↯(\varepsilon) \qquad \mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon}{\rightsquigarrow_{\mathcal{E}} (\exists n.\, \kappa \hookrightarrow (N, \vec{n} \cdot n) * ↯(\mathcal{F}(n)))}$$

Rules of the probabilistic update modality

$_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P$ denotes a resource together with the invariants in $\mathcal{E}_1$, can perform *a randomized logical atomic* operation and split into two parts: P and one satisfying invariants in $\mathcal{E}_2$

$$\frac{P}{\rightsquigarrow_{\mathcal{E}} P}\ \textsc{pupd-ret} \qquad \frac{_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P \qquad P \mathrel{-\!\!*}\ _{\mathcal{E}_2}\!\rightsquigarrow_{\mathcal{E}_3} Q}{_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_3} Q}\ \textsc{pupd-bind} \qquad \frac{_{\mathcal{E}_1}\!\Rrightarrow_{\mathcal{E}_2} P}{_{\mathcal{E}_1}\!\rightsquigarrow_{\mathcal{E}_2} P}\ \textsc{pupd-fupd}$$

$$\frac{\kappa \hookrightarrow (N, \vec{n}) \qquad ⚡(\varepsilon) \qquad \mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon}{\rightsquigarrow_{\mathcal{E}} (\exists n.\ \kappa \hookrightarrow (N, \vec{n} \cdot n) * ⚡(\mathcal{F}(n)))}\ \textsc{pupd-presample-exp}$$

New specification that exposes presampling

$$\forall \iota, c.\ \{\mathit{counter}\ \iota\ c\}\ \mathit{createCtape}(\,)\ \left\{\kappa.\ \mathit{ctape}\ \kappa\ \epsilon\right\}$$

New specification that exposes presampling

$$\forall \iota, c.\ \{\mathit{counter}\ \iota\ c\}\ \mathit{createCtape}(\,)\ \left\{\kappa.\ \mathit{ctape}\ \kappa\ \epsilon\right\}$$

$$\begin{array}{l}
\forall \mathcal{E}, \iota, c, n, \vec{n}, Q. \\
\left\{\begin{array}{l} \mathit{counter}\ \iota\ c \ast \mathit{ctape}\ \kappa\ (n \cdot \vec{n}) \ast \\ \ (\forall z.\ \mathit{cauth}\ z \mathrel{-\!\!\ast} \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ (z + n) \ast Q\, z) \end{array}\right\} \\
\quad \mathit{incr}\ c\ \kappa \\
\left\{z. \mathit{ctape}\ \kappa\ \vec{n} \ast Q\, z\right\}_{\mathcal{E} \uplus \{\iota\}}
\end{array}$$

New specification that exposes presampling

$$\forall \iota, c.\ \{\mathit{counter}\ \iota\ c\}\ \mathit{createCtape}(\,)\ \left\{\kappa.\ \mathit{ctape}\ \kappa\ \epsilon\right\}$$

$$\forall \mathcal{E}, \iota, c, n, \vec{n}, Q.$$

$$\left\{\begin{array}{l} \mathit{counter}\ \iota\ c * \mathit{ctape}\ \kappa\ (n \cdot \vec{n}) * \\ \quad (\forall z.\ \mathit{cauth}\ z \mathrel{-\!\!*} \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ (z+n) * Q\ z) \end{array}\right\}$$

$$\quad \mathit{incr}\ c\ \kappa$$

$$\left\{z. \mathit{ctape}\ \kappa\ \vec{n} * Q\ z\right\}_{\mathcal{E} \uplus \{\iota\}}$$

$$\forall \mathcal{E}, \varepsilon, \mathcal{F}, \vec{n}, \kappa.$$

$$(\lightning(\varepsilon) * (\mathbb{E}_{\mathcal{U}3}[\mathcal{F}] \leqslant \varepsilon) * \mathit{ctape}\ \kappa\ \vec{n} \mathrel{-\!\!*} \rightsquigarrow_{\mathcal{E}} \exists n \in \{0..3\}.\ \lightning(\mathcal{F}(n)) * \mathit{ctape}\ \kappa\ (\vec{n} \cdot [n]))$$

HOCAP specification of *create* and *read*

$$\{\mathsf{True}\}\ \mathit{create}(\,)\ \{c.\ \exists\iota.\ \mathit{counter}\ \iota\ c * \mathit{cfrag}\ 1\ 0\}$$

$$\begin{aligned}&\forall \mathcal{E}, \iota, c, Q.\\ &\quad \{\mathit{counter}\ \iota\ c * (\forall z.\ \mathit{cauth}\ z \twoheadrightarrow\!* \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ z * Q\,z)\}\\ &\qquad \mathit{read}\ c\\ &\quad \{z.\ Q\,z\}_{\mathcal{E} \uplus \{\iota\}}\end{aligned}$$

- *counter* ι c captures the fact that c is a counter with invariant name ι

HOCAP specification of *create* and *read*

$$\{\mathsf{True}\}\ \mathit{create}()\ \{c.\ \exists\iota.\ \mathit{counter}\ \iota\ c * \mathit{cfrag}\ 1\ 0\}$$

$$\begin{array}{l}\forall \mathcal{E}, \iota, c, Q. \\ \quad \{\mathit{counter}\ \iota\ c * (\forall z.\ \mathit{cauth}\ z \mathrel{-\!\!*} \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ z * Q\ z)\} \\ \qquad \mathit{read}\ c \\ \quad \{z.\ Q\ z\}_{\mathcal{E} \uplus \{\iota\}}\end{array}$$

- *counter* ι c captures the fact that c is a counter with invariant name ι
- *cauth* and *cfrag* provides *authoritative* and *fragmental* views of the counter

HOCAP specification of *create* and *read*

$$\{\mathsf{True}\}\ \mathit{create}(\,)\ \{c.\ \exists\iota.\ \mathit{counter}\ \iota\ c * \mathit{cfrag}\ 1\ 0\}$$

$$\begin{array}{l} \forall \mathcal{E}, \iota, c, Q. \\ \quad \{\mathit{counter}\ \iota\ c * (\forall z.\ \mathit{cauth}\ z \mathrel{-\!\!*} \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ z * Q\ z)\} \\ \qquad \mathit{read}\ c \\ \quad \{z.\ Q\ z\}_{\mathcal{E} \uplus \{\iota\}} \end{array}$$

- $\mathit{counter}\ \iota\ c$ captures the fact that c is a counter with invariant name ι
- *cauth* and *cfrag* provides *authoritative* and *fragmental* views of the counter
- Many conditions of these abstract predicates not shown, e.g. $\mathit{cauth}\ n * \mathit{cfrag}\ q\ m \vdash \Rrightarrow_{\mathcal{E}} \mathit{cauth}\ (n+p) * \mathit{cfrag}\ q\ (m+p)$

Second attempt in verifying *conTwoAdd*

$$\{ \lightning(1/16) \ * l \mapsto 0 \}$$

$$(\mathsf{faa}\ l\ (\mathsf{rand}\ 3) \ ||| \ \mathsf{faa}\ l\ (\mathsf{rand}\ 3))\ ;$$
$$!\, l$$

$$\{ v.v > 0 \}$$

Where we left off...

Second attempt in verifying *conTwoAdd*

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_0}^{\gamma_1} * \boxed{\circ S_0}^{\gamma_2} \right\}$$

$$(\mathsf{faa}\ l\ (\mathsf{rand}\ 3) \mathbin{|||} \mathsf{faa}\ l\ (\mathsf{rand}\ 3))\ ;$$
$$!\,l$$

$$\{v.v > 0\}$$

Allocating invariants and resources:

$$\lightning(1/16) * l \mapsto 0 \twoheadrightarrow$$
$$\Rrightarrow \exists \gamma_1\, \gamma_2.\ I(\gamma_1, \gamma_2) * \boxed{\circ S_0}^{\gamma_1} * \boxed{\circ S_0}^{\gamma_2}$$

Second attempt in verifying *conTwoAdd*

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_0}^{\gamma_1} \right\} \ \mathsf{faa}\, l\, (\mathsf{rand}\, 3) \ \left\{ \exists n.\ \boxed{\circ S_2(n)}^{\gamma_1} \right\}$$

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_0}^{\gamma_2} \right\} \ \mathsf{faa}\, l\, (\mathsf{rand}\, 3) \ \left\{ \exists n.\ \boxed{\circ S_2(n)}^{\gamma_2} \right\}$$

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_2(n_1)}^{\gamma_1} * \boxed{\circ S_2(n_2)}^{\gamma_2} \right\} \ !\, l\, \{v.\ v > 0\}$$

Applying HT-PAR-COMP

Second attempt in verifying *conTwoAdd* – First two Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_\circ}^{\gamma_1} \right\}$$

$$\text{faa}\ l\ (\text{rand}\ 3)$$

$$\left\{ \exists n.\ \boxed{\circ S_2(n)}^{\gamma_1} \right\}$$

First Hoare triple

(second Hoare triple is proven similarly)

Recall invariant opening rule:

HT-INV-OPEN

$$\frac{e \text{ atomic} \qquad \{I * P\}\, e\, \{I * Q\}}{\left\{ \boxed{I} * P \right\} e \left\{ \boxed{I} * Q \right\}}$$

Second attempt in verifying *conTwoAdd* – First two Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_1(n)}^{\gamma_1} \right\}$$

$$\quad \mathsf{faa}\ l\ n$$

$$\left\{ \exists n.\ \boxed{\circ S_2(n)}^{\gamma_1} \right\}$$

rand 3 is atomic

We can open invariants temporarily and update ghost resources to track n sampled

Second attempt in verifying *conTwoAdd* – First two Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_1(n)}^{\gamma_1} \right\}$$

$$\texttt{faa}\ l\ n$$

$$\left\{ \exists n.\ \boxed{\circ S_2(n)}^{\gamma_1} \right\}$$

rand 3 is atomic

We can open invariants temporarily and update ghost resources to track n sampled

We can do the same again with faa $l\ n$

Second attempt in verifying *conTwoAdd* – last Hoare triples

Last Hoare triple

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_2(n_1)}^{\gamma_1} * \boxed{\circ S_2(n_2)}^{\gamma_2} \right\}$$

$$!\, l$$

$$\{v.\, v > 0\}$$

Second attempt in verifying *conTwoAdd* – last Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_2(n_1)}^{\gamma_1} * \boxed{\circ S_2(n_2)}^{\gamma_2} \right\}$$

$$!\, l$$

$$\{v.\, v > 0\}$$

Last Hoare triple

- But nothing too surprising!

Second attempt in verifying *conTwoAdd* – last Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_2(n_1)}^{\gamma_1} * \boxed{\circ S_2(n_2)}^{\gamma_2} \right\}$$

$$! \, l$$

$$\{v.\, v > 0\}$$

Last Hoare triple

- But nothing too surprising!
- $!\,l$ is atomic, so we can open invariants and do a case split on value of n_1 and n_2.

Second attempt in verifying *conTwoAdd* – last Hoare triples

$$\left\{ \boxed{I(\gamma_1, \gamma_2)}^{\iota} * \boxed{\circ S_2(n_1)}^{\gamma_1} * \boxed{\circ S_2(n_2)}^{\gamma_2} \right\}$$

$$!\,l$$

$$\{v.\, v > 0\}$$

Last Hoare triple

- But nothing too surprising!
- $!\,l$ is atomic, so we can open invariants and do a case split on value of n_1 and n_2.
- If both are 0, we get $\lightning(1)$ and can derive $\bot$!

Weakest-pre

$$
\begin{aligned}
\mathsf{wp}\ e_1\ \{\Phi\} \triangleq \forall \sigma_1, \varepsilon_1.\ S(\sigma_1, \varepsilon_1) \mathrel{-\!\!*}\ {}_{\top}\!\Rrightarrow_{\emptyset}\ \mathsf{sstep}\ \sigma_1\ \varepsilon_1\ \{\sigma_2, \varepsilon_2.\\
&\big(e_1 \in \mathit{Val} * {}_{\emptyset}\!\Rrightarrow_{\top} S(\sigma_2, \varepsilon_2) * \Phi(e_1)\big) \vee \\
&\big(e_1 \notin \mathit{Val} * \mathsf{pstep}\ (e_1, \sigma_2)\ \varepsilon_2\ \{e_2, \sigma_3, l, \varepsilon_3.\\
&\quad \triangleright\, \mathsf{sstep}\ \sigma_3\ \varepsilon_3\ \{\sigma_4, \varepsilon_4.\ {}_{\emptyset}\!\Rrightarrow_{\top} S(\sigma_4, \varepsilon_4) * \mathsf{wp}\ e_2\ \{\Phi\} * \mathop{\bigstar}_{e' \in l} \mathsf{wp}\ e'\ \{\mathsf{True}\}\}\}\big)\}
\end{aligned}
$$

State and program step precondition

$$\textsc{State-step-err-1}\quad \frac{1 \leqslant \varepsilon}{\mathsf{sstep}\ \sigma\ \varepsilon\ \{\Phi\}}$$

$$\textsc{State-step-ret}\quad \frac{\Phi(\sigma, \varepsilon)}{\mathsf{sstep}\ \sigma\ \varepsilon\ \{\Phi\}}$$

$$\textsc{State-step-continuous}\quad \frac{\forall \varepsilon'.\ \varepsilon < \varepsilon' \twoheadrightarrow \mathsf{sstep}\ \sigma\ \varepsilon'\ \{\Phi\}}{\mathsf{sstep}\ \sigma\ \varepsilon\ \{\Phi\}}$$

$$\textsc{State-step-exp}\quad \frac{\mathbb{E}_{\mu}[\mathcal{F}] \leqslant \varepsilon \qquad \mathsf{schErasable}(\mu, \sigma_1) \qquad \forall \sigma_2.\ 0 < \mu(\sigma_2) \twoheadrightarrow \mathsf{sstep}\ \sigma_2\ (\mathcal{F}(\sigma_2))\ \{\Phi\}}{\mathsf{sstep}\ \sigma_1\ \varepsilon\ \{\Phi\}}$$

$$\textsc{Prog-step-exp}\quad \frac{\begin{array}{c}\mathsf{red}(e_1, \sigma_1) \qquad \mathbb{E}_{\mathsf{step}(e_1,\sigma_1)}[\mathcal{F}] \leqslant \varepsilon \\ \forall (e_2, \sigma_2, l).\ 0 < \mathsf{step}(e_1, \sigma_1)(e_2, \sigma_2, l) \twoheadrightarrow \Phi(e_2, \sigma_2, l, \mathcal{F}(e_2, \sigma_2, l))\end{array}}{\mathsf{pstep}\ (e_1, \sigma_1)\ \varepsilon\ \{\Phi\}}$$

Probabilistic update modality

$$ {}_{\mathcal{E}_1}\!\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}_2} P \triangleq \forall \sigma_1, \varepsilon_1.\, S(\sigma_1, \varepsilon_1) \mathrel{-\!\!*} {}_{\mathcal{E}_1}\!\mathrel{|\!\!\Rrightarrow}_{\emptyset} \mathsf{sstep}\ \sigma_1\ \varepsilon_1 \{\sigma_2, \varepsilon_2.\ {}_{\emptyset}\!\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_2} S(\sigma_2, \varepsilon_2) * P\} $$

PUPD-ELIM

$$ \frac{\{P * Q\}\, e\, \{R\}_{\mathcal{E}}}{\{(\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}} P) * Q\}\, e\, \{R\}_{\mathcal{E}}} $$

PUPD-RET

$$ \frac{P}{\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}} P} $$

PUPD-BIND

$$ \frac{{}_{\mathcal{E}_1}\!\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}_2} P \qquad P \mathrel{-\!\!*} {}_{\mathcal{E}_2}\!\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}_3} Q}{{}_{\mathcal{E}_1}\!\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}_3} Q} $$

PUPD-FUPD

$$ \frac{{}_{\mathcal{E}_1}\!\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_2} P}{{}_{\mathcal{E}_1}\!\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}_2} P} $$

PUPD-PRESAMPLE-EXP

$$ \frac{\mathbb{E}_{\mathfrak{U}N}[\mathcal{F}] \leqslant \varepsilon \qquad ⚡(\varepsilon) \qquad \kappa \hookrightarrow (N, \vec{n})}{\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}} (\exists n.\, \kappa \hookrightarrow (N, \vec{n} \cdot n) * ⚡(\mathcal{F}(n)))} $$

PUPD-ERR

$$ \frac{}{\mathrel{|\!\!\rightsquigarrow}_{\mathcal{E}} (\exists \varepsilon.\, 0 < \varepsilon * ⚡(\varepsilon))} $$