Probabilistic Reasoning in Higher-Order Separation Logic

Simon Oddershede Gregersen
CISPA Helmholtz Center for Information Security

joint work with
Alejandro Aguirre[1], Philipp G. Haselwarter[1], Kwing Hei-Li[1], Lars Birkedal[1],
Markus de Medeiros[2], Puming Liu[2], Alex Bai[2], Joseph Tassarotti[2]

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

Randomness is an essential tool

Randomness is an essential tool

Cryptography

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

Randomness is an essential tool

Cryptography

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

Differential privacy

Randomized noise: Laplace & Gaussian mechanisms, randomized response, ...

Randomness is an essential tool

Cryptography

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

Algorithms and data structures

Simpler and faster algorithms: hashing, Bloom filters, skip lists, Miller–Rabin, …

Differential privacy

Randomized noise: Laplace & Gaussian mechanisms, randomized response, …

Randomness is an essential tool

Cryptography

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

Algorithms and data structures

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

Differential privacy

Randomized noise: Laplace & Gaussian mechanisms, randomized response, ...

Machine learning

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

Randomness is an essential tool

Cryptography

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

Algorithms and data structures

Simpler and faster algorithms: hashing, Bloom filters, skip lists, Miller–Rabin, …

Differential privacy

Randomized noise: Laplace & Gaussian mechanisms, randomized response, …

Machine learning

Stochastic Gradient Descent, Markov Chain Monte Carlo, …

Program verification techniques need to handle randomness.

Probabilistic verification: Many successes!

- **Cryptographic security**
  - EasyCrypt / CryptHOL / … : OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC, HMAC, KEM-DEM, PRF-based encryption

Probabilistic verification: Many successes!

- **Cryptographic security**
  - EasyCrypt / CryptHOL / … : OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC, HMAC, KEM-DEM, PRF-based encryption
- **Differential privacy**
  - apRHL: Laplace & Gaussian mechanisms, Sparse Vector, Above Threshold
  - Fuzz / DFuzz / Duet / LightDP: types for DP, automated proofs

Probabilistic verification: Many successes!

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

Probabilistic verification: Many successes!

- **Cryptographic security**
  - EasyCrypt / CryptHOL / … : OAEP, Cramer–Shoup, FDH signatures, ElGamal, constant-time MEE-CBC, HMAC, KEM-DEM, PRF-based encryption
- **Differential privacy**
  - apRHL: Laplace & Gaussian mechanisms, Sparse Vector, Above Threshold
  - Fuzz / DFuzz / Duet / LightDP: types for DP, automated proofs
- **Randomized algorithms**
  - Union-bound logic (aHL): upper bound error probabilities
  - Ellora: polynomial identity testing, random walks, concentration bounds
  - Caesar / HeyVL: weakest pre-expectation reasoning for pGCL
- **Machine learning**
  - ePRHL: expected sensitivity properties, stability of stochastic gradient decent, …

Limitations… and a divide

Probabilistic reasoning

- small programs
- restricted languages
- not compositional

Limitations… and a divide

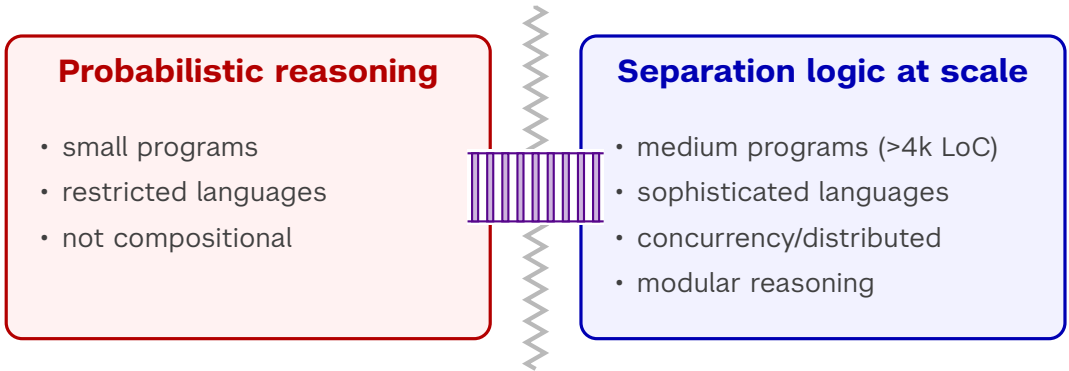
Probabilistic reasoning

- small programs
- restricted languages
- not compositional

Separation logic at scale

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

Limitations… and a divide



Can we bridge these two worlds?

The Clutch Project `clutch-project.org`

Approach: Extend modern separation logic with reasoning principles from probabilistic logics.

- **Relational Reasoning**: Clutch [POPL24], Approxis [POPL25], Foxtrot [PLDI26].
- **Differential Privacy**: Clutch-DP [PLDI26].
- **Expected Runtime**: Tachis [OOPSLA24].
- **Error Bounding**: Eris [ICFP24], Coneris [ICFP25], Continuous-Eris [LICS26]

All results mechanized in the Rocq proof assistant.

The Clutch Project `clutch-project.org`

Approach: Extend modern separation logic with reasoning principles from probabilistic logics.

- **Relational Reasoning**: Clutch [POPL24], Approxis [POPL25], Foxtrot [PLDI26].
- **Differential Privacy**: Clutch-DP [PLDI26].
- **Expected Runtime**: Tachis [OOPSLA24].
- **Error Bounding**: **Eris [ICFP24]**, Coneris [ICFP25], Continuous-Eris [LICS26]

All results mechanized in the Rocq proof assistant.

A bit of background …

Hoare Logic

In 1960s Floyd and Hoare pioneered what is now known as Hoare logic.

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

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

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

Separation logic

Separation logic adds the “separating conjunction” connective $*$ to Hoare logic:

$$P * Q$$

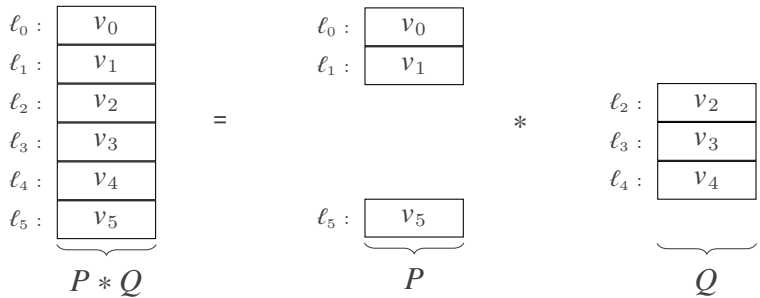
“the state can be split into disjoint pieces satisfying P and Q”

Separation logic

Separation logic adds the “separating conjunction” connective $*$ to Hoare logic:

$$P * Q$$

“the state can be split into disjoint pieces satisfying P and Q”



Points-to assertions and rules

The points-to assertion $\ell \mapsto v$ says that location ℓ stores value v, and confers a "permission to access and modify" or **ownership** of ℓ.

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

Points-to assertions and rules

The points-to assertion $\ell \mapsto v$ says that location ℓ stores value v, and confers a "permission to access and modify" or **ownership** of ℓ.

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

- **Allocation:** $\{\text{True}\}\ \mathsf{ref}(v)\ \{\ell.\, \ell \mapsto v\}$

Points-to assertions and rules

The points-to assertion $\ell \mapsto v$ says that location ℓ stores value v, and confers a "permission to access and modify" or **ownership** of ℓ.

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

- **Allocation:** $\{\text{True}\}\ \text{ref}(v)\ \{\ell.\, \ell \mapsto v\}$
- **Read:** $\{\ell \mapsto v\}\ !\ell\ \{x.\, x = v * \ell \mapsto v\}$

Points-to assertions and rules

The points-to assertion $\ell \mapsto v$ says that location ℓ stores value v, and confers a "permission to access and modify" or **ownership** of ℓ.

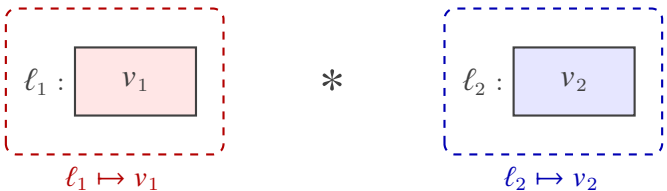
To read or write ℓ, the precondition must contain $\ell \mapsto -$.

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

Disjointness from points-to

Since $*$ requires splitting into **disjoint** heap pieces:

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



A single cell cannot belong to both pieces, so ℓ_1 and ℓ_2 must differ.

The frame rule

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

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

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

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

The Iris framework

Iris is a separation logic framework, mechanized in Rocq. Used for many projects:

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

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

Eris: Error credits for “up-to-bad” reasoning

The RandML language

An **OCaml-like language** with ..., and **probabilistic uniform sampling**.

$$e ::= \ldots \mid \mathsf{ref}(e) \mid\ !\, e \mid e_1 := e_2 \mid \mathsf{rand}(e)$$

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

Motivating example

$$
\begin{aligned}
\textit{collide} \quad \triangleq \quad & \mathsf{let}\, x = \mathsf{rand}(2^{64} - 1)\, \mathsf{in} \\
& \mathsf{let}\, y = \mathsf{rand}(2^{64} - 1)\, \mathsf{in} \\
& x = y
\end{aligned}
$$

The probability that *collide* returns true is 2^{-64}.

(Almost) specifications

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

$$\{\text{True}\}\ \textit{collide}\ \{x.\, x = \text{false}\}_{\approx}$$

meaning the postcondition is almost always true?

Barthe et al.'s approximate Hoare Logic

Annotate triples with a non-negative real number ε:

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

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

Set $\varepsilon = 0$ for the usual Hoare triple.

Barthe et al.'s approximate Hoare Logic

Annotate triples with a non-negative real number ε:

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

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

Set $\varepsilon = 0$ for the usual Hoare triple.

Our example would be:

$$\{\text{True}\}\ \textit{collide}\ \{x.\, x = \text{false}\}_{2^{-64}}$$

Basic aHL Rules (adapted)

Basic aHL Rules (adapted)

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

Basic aHL Rules (adapted)

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

$$\frac{\{P\}\, e\, \{Q\}_{\varepsilon_1} \qquad \varepsilon_1 \leq \varepsilon_2}{\{P\}\, e\, \{Q\}_{\varepsilon_2}}$$

Basic aHL Rules (adapted)

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}} \qquad \frac{\{P\}\, e\, \{Q\}_{\varepsilon_1} \qquad \varepsilon_1 \leq \varepsilon_2}{\{P\}\, e\, \{Q\}_{\varepsilon_2}}$$

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

Basic aHL Rules (adapted)

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}} \qquad \frac{\{P\}\, e\, \{Q\}_{\varepsilon_1} \qquad \varepsilon_1 \leq \varepsilon_2}{\{P\}\, e\, \{Q\}_{\varepsilon_2}}$$

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

Example: use $\varepsilon = 1/2$ to "avoid" heads or tails when evaluating $\mathsf{rand}(1)$.

Limitation 1: No error dependency

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

What if the error for e_2 **depends** on what e_1 did?

Limitation 1: No error dependency

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

What if the error for e_2 **depends** on what e_1 did?

$$\begin{aligned} \mathit{collideAlt} \triangleq\ & \mathsf{let}\, l = \mathsf{rand}(128)\, \mathsf{in} \\ & \mathsf{let}\, x = \mathsf{rand}(2^l - 1)\, \mathsf{in} \\ & \mathsf{let}\, y = \mathsf{rand}(2^l - 1)\, \mathsf{in} \\ & x = y \end{aligned}$$

Limitation 2: Errors propagate everywhere

Consider a standard specification for list iter:

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

Limitation 2: Errors propagate everywhere

Consider a standard specification for list iter:

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

Now consider an “error”-aware version:

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

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

Limitation 3: Errors expose internals

Consider a Random-Oracle model of a hash function:

$$
\begin{array}{lll}
\mathit{genhash} & \triangleq & \mathsf{let}\ m = \mathit{init\_map}\ () \ \mathsf{in} \\
 & & (\mathsf{fun}\ k \ \longrightarrow \ \mathsf{match}\ \mathit{get}\ m\ k\ \mathsf{with} \\
 & & \qquad \mathrm{Some}(x) \Longrightarrow x \\
 & & \qquad \mid \mathrm{None} \quad \Longrightarrow \mathsf{let}\ x = \mathit{rand256}() \ \mathsf{in} \\
 & & \qquad\qquad\qquad\quad \mathit{set}\ m\ x; x \\
 & & \qquad \mathsf{end})
\end{array}
$$

Collisions are unlikely if few hashes are made.

Limitation 3: Errors expose internals

We want a specification like

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

But what $\mathcal{E}$ should we pick?

Limitation 3: Errors expose internals

We want a specification like

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

But what ε should we pick?

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

Credits

We want a better way to track the error.

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

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

Credits

We want a better way to track the error.

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

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

The specification

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

implies e does at most n calls to $\mathsf{tick}()$.

Atkey's motivation: amortized reasoning

Operations in some data structures do not have **constant** costs:

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

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

Atkey's motivation: amortized reasoning

Operations in some data structures do not have **constant** costs:

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

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

With amortized point of view:

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

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

Amortized reasoning

Consider a dynamic, resizing array:

$$\{\$3 * \text{dynarray}(a, l)\}$$
$$\quad \text{arrayInsert } a\ v$$
$$\{\text{dynarray}(a, l +\!\!+ [v])\}$$

Idea: "bank" excess credits in $\text{dynarray}(a, l)$ predicate in postcondition.

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

Eris

Eric introduces **error credits** for approximate bounds. The error credit assertion $↯(\varepsilon)$ represents a “permission” to incur ε approximation error.

The triple

$$\{↯(\varepsilon) * P\}\ e\ \{Q\}$$

is then analogous to

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

Error credit rules: Spending

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

Error credit rules: Splitting

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

Implies sequencing:

$$\frac{\{\lightning(\varepsilon_1) * P\}\, e_1\, \{Q\} \qquad \{\lightning(\varepsilon_2) * Q\}\, e_2\, \{R\}}{\{\lightning(\varepsilon_1 + \varepsilon_2) * P\}\, e_1; e_2\, \{R\}}$$

Error credit rules: Max error

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

$$↯(1) \vdash \text{False}$$

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

Error credit rules: Averaging

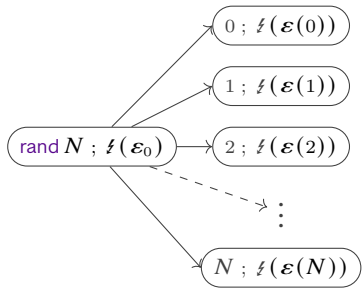
Increase error credits along branches of a sample, as long as **expected value** is the same.

$$\frac{\mathbb{E}_{x\sim\mathrm{Unif}(N)}[\varepsilon(x)] = \varepsilon_0}{\{\lightning(\varepsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ \lightning(\varepsilon(x))\}}$$

Error credit rules: Averaging

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

$$\frac{\mathbb{E}_{x\sim\mathrm{Unif}(N)}[\varepsilon(x)] = \varepsilon_0}{\{\lightning(\varepsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ \lightning(\varepsilon(x))\}}$$

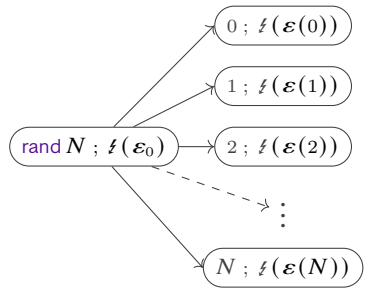


Error credit rules: Averaging

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

$$\frac{\mathbb{E}_{x\sim \mathrm{Unif}(N)}[\varepsilon(x)] = \varepsilon_0}{\{\lightning(\varepsilon_0)\}\ \mathsf{rand}(N)\ \{x.\ \lightning(\varepsilon(x))\}}$$

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



Resolves limitation 1: Error dependency

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

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

Resolves limitation 1: Error dependency

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

$$\frac{\{P\}\, e_1\, \{Q\}_{\varepsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\varepsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\varepsilon_1 + \varepsilon_2}}$$

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

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

Q can refer to state after e_1

Resolves limitation 2: Error propagation

Recall specification for list iter:

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

Resolves limitation 2: Error propagation

Recall specification for list iter:

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

Can **derive** error credit version:

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

Resolves limitation 3: Can amortize costs

Amortize by putting error credits in the representation predicate.

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

$$\{\lightning(\varepsilon) * \mathsf{hash}(f, m) * k \notin \mathsf{dom}(m)\}$$
$$f\ k$$
$$\{v.\ \mathsf{hash}(f, m[k := v]) * v \notin \mathsf{cod}(m)\}$$

* up to a fixed maximum number of insertions

Retains expressivity of separation logic

Eris **preserves** all the standard rules of modern separation logic.

Retains expressivity of separation logic

Eris **preserves** all the standard rules of modern separation logic.

Generalizations:

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

Retains expressivity of separation logic

Eris **preserves** all the standard rules of modern separation logic.

Generalizations:

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

Example case studies:

- Merkle tree
- Resizing hash tables
- Concurrent Bloom Filter

Conclusion

The Clutch project is an ongoing effort to extend modern separation logic with probabilistic reasoning principles.

- **Relational Reasoning**: Clutch [POPL24], Approxis [POPL25], Foxtrot [PLDI26].
- **Differential Privacy**: Clutch-DP [PLDI26].
- **Expected Runtime**: Tachis [OOPSLA24].
- **Error Bounding**: Eris [ICFP24], Coneris [ICFP25], Continuous-Eris [LICS26]

See more at `clutch-project.org`.

Conclusion

The Clutch project is an ongoing effort to extend modern separation logic with probabilistic reasoning principles.

- **Relational Reasoning**: Clutch [POPL24], Approxis [POPL25], Foxtrot [PLDI26].
- **Differential Privacy**: Clutch-DP [PLDI26].
- **Expected Runtime**: Tachis [OOPSLA24].
- **Error Bounding**: Eris [ICFP24], Coneris [ICFP25], Continuous-Eris [LICS26]

See more at `clutch-project.org`.

Thank you!