Approximate Relational Reasoning for Higher-Order Probabilistic Programs

Philipp G. Haselwarter[1], Kwing Hei Li[1], Alejandro Aguirre[1], Simon Oddershede Gregersen[2], Joseph Tassarotti[2], and Lars Birkedal[1]

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

Our Contribution: Approxis

- Relational higher-order separation logic for **approximate program refinement** for **Randomized ML**!

Our Contribution: Approxis

- Relational higher-order separation logic for **approximate program refinement** for **Randomized ML**!
  - **error probability** ε represented as a first-class separation logic **resource** $\lightning(\varepsilon)$
  - full power of non-probabilistic reasoning preserved (invariants, ghost state, …)
  - expressive probabilistic reasoning rules manipulate $\lightning(\varepsilon)$ locally

Our Contribution: Approxis

- Relational higher-order separation logic for **approximate program refinement** for **Randomized ML**!
  - **error probability** ε represented as a first-class separation logic **resource** $\lightning(\varepsilon)$
  - full power of non-probabilistic reasoning preserved (invariants, ghost state, …)
  - expressive probabilistic reasoning rules manipulate $\lightning(\varepsilon)$ locally
- New **logical relation** for program approximation & contextual equivalence

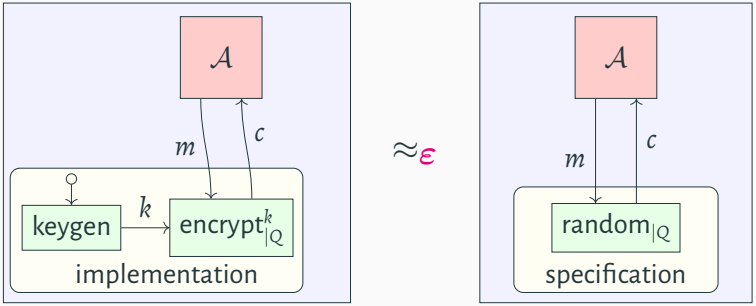
Our Contribution: Approxis

- Relational higher-order separation logic for **approximate program refinement** for **Randomized ML**!
  - **error probability** ε represented as a first-class separation logic **resource** $\lightning(\varepsilon)$
  - full power of non-probabilistic reasoning preserved (invariants, ghost state, …)
  - expressive probabilistic reasoning rules manipulate $\lightning(\varepsilon)$ locally
- New **logical relation** for program approximation & contextual equivalence
- Case studies
  - cryptographic security: symmetric key encryption, …
  - rejection samplers: sampling a node from a B+ tree, …

Our Contribution: Approxis

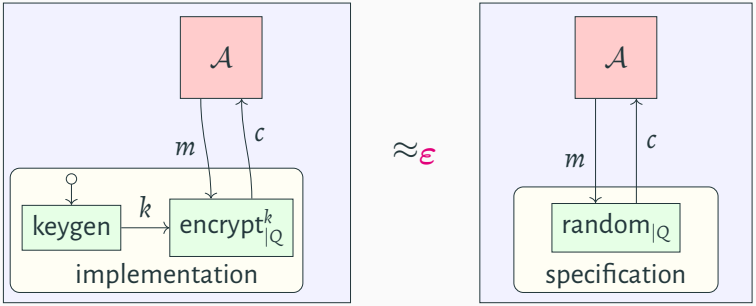
- Relational higher-order separation logic for **approximate program refinement** for **Randomized ML**!
  - **error probability** ε represented as a first-class separation logic **resource** $\lightning(\varepsilon)$
  - full power of non-probabilistic reasoning preserved (invariants, ghost state, …)
  - expressive probabilistic reasoning rules manipulate $\lightning(\varepsilon)$ locally
- New **logical relation** for program approximation & contextual equivalence
- Case studies
  - cryptographic security: symmetric key encryption, …
  - rejection samplers: sampling a node from a B+ tree, …
- Fully mechanised in  and 

Example from Cryptography



Security: "Encrypted messages c appear random if the key k is unknown to $\mathcal{A}$."

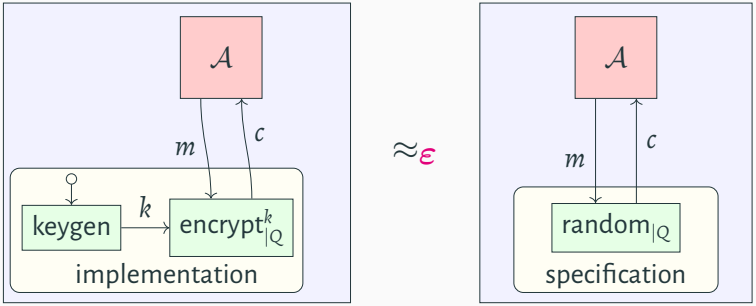
Example from Cryptography



Security: "Encrypted messages c appear random if the key k is unknown to $\mathcal{A}$."

Game: Adversary $\mathcal{A}$ can query an abstract encryption "oracle" with Q messages m.

Example from Cryptography

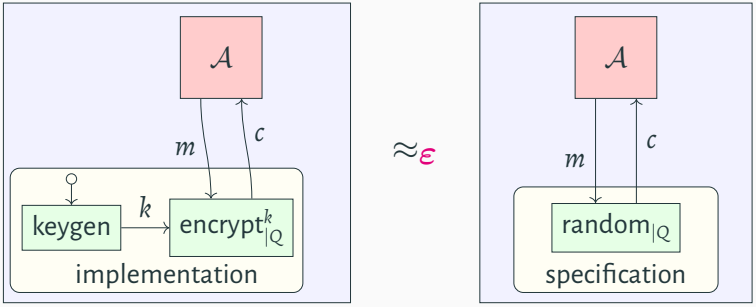


Security: "Encrypted messages c appear random if the key k is unknown to $\mathcal{A}$."

Game: Adversary $\mathcal{A}$ can query an abstract encryption "oracle" with Q messages m.

Goal: Return true for the implementation and false for the specification.

Example from Cryptography



Security: "Encrypted messages c appear random if the key k is unknown to $\mathcal{A}$."

Game: Adversary $\mathcal{A}$ can query an abstract encryption "oracle" with Q messages m.

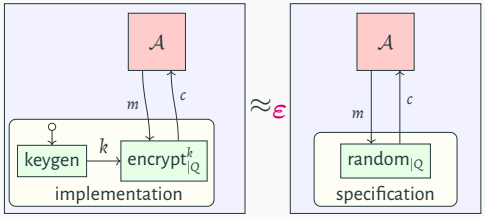
Goal: Return true for the implementation and false for the specification.

Claim: No adversary $\mathcal{A}$ can distinguish with probability better than ε !

Security in Approxis

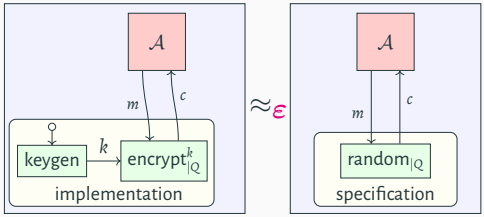
Intuition:

“encrypt behaves (almost) like the uniform distribution on ciphertexts”



Security in Approxis

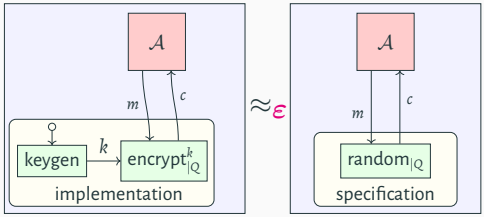
Intuition:
"encrypt behaves (almost) like the uniform distribution on ciphertexts"



Mathematically: $\Pr\left[\mathcal{A}(\text{encrypt}^k_{|Q}) \Downarrow \text{true}\right] = \Pr\left[\mathcal{A}(\text{random}_{|Q}) \Downarrow \text{true}\right] \pm \varepsilon$

Security in Approxis

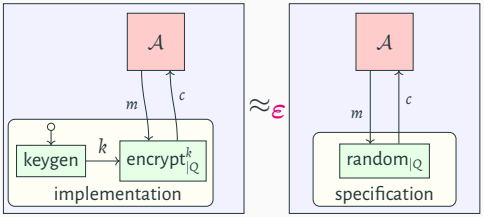
Intuition:
"encrypt behaves (almost) like the uniform distribution on ciphertexts"



Mathematically: $\Pr\left[\mathcal{A}(\text{encrypt}^k_{|Q}) \Downarrow \text{true}\right] = \Pr\left[\mathcal{A}(\text{random}_{|Q}) \Downarrow \text{true}\right] \pm \varepsilon$

Security in Approxis

Intuition:
"encrypt behaves (almost) like the uniform distribution on ciphertexts"



Mathematically: $\Pr\left[\mathcal{A}(\mathsf{encrypt}^k_{|\mathrm{Q}}) \Downarrow \mathsf{true}\right] = \Pr\left[\mathcal{A}(\mathsf{random}_{|\mathrm{Q}}) \Downarrow \mathsf{true}\right] \pm \varepsilon$

In Approxis: $\lightning(\varepsilon) \multimap *\ \mathsf{rwp}\ \mathcal{A}(\mathsf{encrypt}^k_{|\mathrm{Q}}) \precsim \mathcal{A}(\mathsf{random}_{|\mathrm{Q}})\ \{b_1 = b_2\}$

Reasoning about Error Probability: Error Credits

- Error as first class **resource** [Eris, ICFP'24]: $↯(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $↯(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

Reasoning about Error Probability: Error Credits

- Error as first class **resource** [Eris, ICFP'24]: $↯(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $↯(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

Reasoning about Error Probability: Error Credits

- Error as first class **resource** [Eris, ICFP'24]: $↯(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $↯(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

- Laws: $↯(\varepsilon_1 + \varepsilon_2) \dashv\vdash ↯(\varepsilon_1) * ↯(\varepsilon_2)$ $↯(1) \vdash \bot$

Reasoning about Error Probability: Error Credits

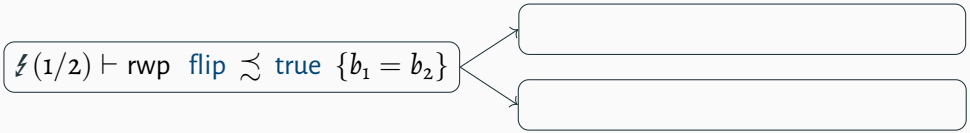
- Error as first class **resource** [Eris, ICFP'24]: $\lightning(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $\lightning(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

- Laws: $\lightning(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning(\varepsilon_1) * \lightning(\varepsilon_2)$ $\lightning(1) \vdash \bot$

- Error budget can increase during execution if expectation is preserved, *e.g.*:



Reasoning about Error Probability: Error Credits

- Error as first class **resource** [Eris, ICFP'24]: $\lightning(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $\lightning(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

- Laws: $\lightning(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning(\varepsilon_1) * \lightning(\varepsilon_2)$ $\lightning(1) \vdash \bot$

- Error budget can increase during execution if expectation is preserved, *e.g.*:

$$\lightning(1/2) \vdash \mathsf{rwp}\ \ \mathsf{flip} \precsim \mathsf{true}\ \ \{b_1 = b_2\}$$

$$\vdash \mathsf{rwp}\ \ \mathsf{true} \precsim \mathsf{true}\ \ \{b_1 = b_2\}$$

$$\vdash \mathsf{rwp}\ \ \mathsf{false} \precsim \mathsf{true}\ \ \{b_1 = b_2\}$$

Reasoning about Error Probability: Error Credits

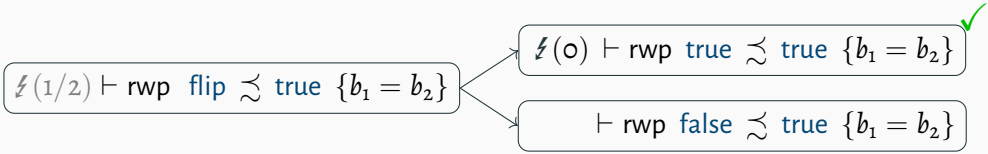
- Error as first class **resource** [Eris, ICFP'24]: $\lightning(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $\lightning(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

- Laws: $\lightning(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning(\varepsilon_1) * \lightning(\varepsilon_2)$ $\lightning(1) \vdash \bot$

- Error budget can increase during execution if expectation is preserved, *e.g.*:



Reasoning about Error Probability: Error Credits

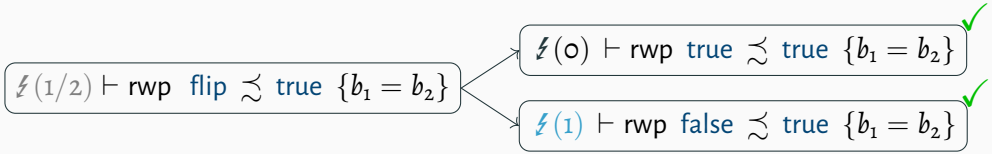
- Error as first class **resource** [Eris, ICFP'24]: $\lightning(\varepsilon)$ "up to ε" $\varepsilon \in [0, 1]$

  Budget intuition for $\lightning(\varepsilon) \vdash P$: "P holds **up to error** ε" *i.e.* $\Pr[\neg P] \leq \varepsilon$

- Approxis is flexible about when to "spend" this budget (*e.g.*, amortization)

- Laws: $\lightning(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning(\varepsilon_1) * \lightning(\varepsilon_2)$ $\lightning(1) \vdash \bot$

- Error budget can increase during execution if expectation is preserved, *e.g.*:



Reasoning about Known and Unknown Code

$$\lightning(\varepsilon) \twoheadrightarrow \mathsf{rwp}\ \mathcal{A}(\mathsf{encrypt}^{k}_{|Q}) \precsim \mathcal{A}(\mathsf{random}_{|Q})\ \{b_1 = b_2\}$$

Reasoning about Known and Unknown Code

$$\lightning(\varepsilon) \multimap * \text{ rwp } \mathcal{A}(\text{encrypt}^{k}_{|Q}) \precsim \mathcal{A}(\text{random}_{|Q}) \ \{b_1 = b_2\}$$

- Reason **separately** about $\mathcal{A}$ and oracles

Reasoning about Known and Unknown Code

$$\lightning(\varepsilon) \mathrel{-\!\!*} \text{rwp}\ \ \text{encrypt}^k_{|Q} \precsim \text{random}_{|Q}\ \ \{\phi\} \qquad (1)$$

$$(2)$$

$$\lightning(\varepsilon) \mathrel{-\!\!*} \text{rwp}\ \ \mathcal{A}(\text{encrypt}^k_{|Q}) \precsim \mathcal{A}(\text{random}_{|Q})\ \ \{b_1 = b_2\}$$

- Reason **separately** about $\mathcal{A}$ and oracles
- For $\text{encrypt}^k_{|Q}$ and $\text{random}_{|Q}$: use **refinement logic rules** to prove (1)
  invariants, local state, probabilistic rules for errors, ...

Reasoning about Known and Unknown Code

$$\lightning(\varepsilon) \multimap \mathsf{rwp}\ \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\} \tag{1}$$

$$\forall f_1, f_2.\ \phi(f_1, f_2) \multimap \mathsf{rwp}\ \mathcal{A}(f_1) \precsim \mathcal{A}(f_2)\ \{b_1 = b_2\} \tag{2}$$

$$\lightning(\varepsilon) \multimap \mathsf{rwp}\ \mathcal{A}(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}(\mathsf{random}_{|Q})\ \{b_1 = b_2\}$$

- Reason **separately** about $\mathcal{A}$ and oracles
- For $\mathsf{encrypt}^k_{|Q}$ and $\mathsf{random}_{|Q}$: use **refinement logic rules** to prove (1)
  invariants, local state, probabilistic rules for errors, ...
- But we don't know the code of $\mathcal{A}$

Reasoning about Known and Unknown Code

$$\lightning(\varepsilon) \multimap \mathsf{rwp}\ \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\} \qquad (1)$$

$$\forall f_1, f_2.\ \phi(f_1, f_2) \multimap \mathsf{rwp}\ \mathcal{A}(f_1) \precsim \mathcal{A}(f_2)\ \{b_1 = b_2\} \qquad (2)$$

$$\lightning(\varepsilon) \multimap \mathsf{rwp}\ \mathcal{A}(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}(\mathsf{random}_{|Q})\ \{b_1 = b_2\}$$

- Reason **separately** about $\mathcal{A}$ and oracles
- For $\mathsf{encrypt}^k_{|Q}$ and $\mathsf{random}_{|Q}$: use **refinement logic rules** to prove (1)
  invariants, local state, probabilistic rules for errors, ...
- But we don't know the code of $\mathcal{A}$ $\implies$ use **type** of $\mathcal{A}$ to derive (2) !

Approximate Refinement: Errors & Invariants

$$\lightning(\varepsilon) \multimap \mathsf{rwp}\ \ \mathsf{encrypt}^{k}_{\upharpoonright Q} \precsim \mathsf{random}_{\upharpoonright Q}\ \{\phi\}$$

Approximate Refinement: Errors & Invariants

$$\lightning(\varepsilon) \multimap * \ \mathsf{rwp}\ \ \mathsf{encrypt}^{k}_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\}$$

- The oracles satisfy an invariant. Initially, $i = 0$ and $\varepsilon = \sum_{k=0}^{Q-1} \frac{k}{2N} = \frac{(Q-1)Q}{2N}$.

$$Inv \triangleq \quad \exists\, (i \leq Q)\ .\ \mathsf{counter} \mapsto i\ *\ \mathsf{counter}' \mapsto_s i\ *\ \lightning\left(\frac{(Q-1)Q}{2N} - \frac{(i-1)i}{2N}\right)$$

Approximate Refinement: Errors & Invariants

$$↯(\varepsilon) \twoheadrightarrow \mathsf{rwp}\ \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\}$$

- The oracles satisfy an invariant. Initially, $i = 0$ and $\varepsilon = \sum_{k=0}^{Q-1} \frac{k}{2N} = \frac{(Q-1)Q}{2N}$.

$$Inv \triangleq \exists (i \leq Q)\ .\ \mathsf{counter} \mapsto i\ *\ \mathsf{counter}' \mapsto_s i\ *\ ↯\left(\frac{(Q-1)Q}{2N} - \frac{(i-1)i}{2N}\right)$$

- We transfer ownership of the initial error credits into the invariant

$$\boxed{Inv}^{\gamma} \twoheadrightarrow \mathsf{rwp}\ \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\}$$

Approximate Refinement: Error Resource Management

Every invocation of the oracles preserves the invariant.

- Increment counters: $\text{counter} \mapsto i{+}1 \;*\; \text{counter}' \mapsto_s i{+}1 \;*\; \lightning(\varepsilon_i)$

Approximate Refinement: Error Resource Management

Every invocation of the oracles preserves the invariant.

- Increment counters: $\text{counter} \mapsto i{+}1 \;*\; \text{counter}' \mapsto_s i{+}1 \;*\; \lightning(\varepsilon_i)$

$$\frac{\text{counter} \mapsto i \qquad \text{counter} \mapsto i \;\mathrel{-\!\!*}\; \text{rwp}\ i \precsim e\ \{\Phi\}}{\text{rwp}\ \ !\,\text{counter} \precsim e\ \{\Phi\}}$$

Approximate Refinement: Error Resource Management

Every invocation of the oracles preserves the invariant.

- Increment counters: $\text{counter} \mapsto i{+}1 \;*\; \text{counter}' \mapsto_s i{+}1 \;*\; \lightning(\varepsilon_i)$
- Split credits: $\lightning(\varepsilon_i) \;\;\multimap\!\!*\;\; \lightning(\varepsilon_{i+1}) \;*\; \lightning(i/N)$

$$\frac{\text{counter} \mapsto i \qquad \text{counter} \mapsto i \;\multimap\!\!*\; \text{rwp}\; i \precsim e\; \{\Phi\}}{\text{rwp}\;\; !\,\text{counter} \precsim e\; \{\Phi\}}$$

$$\lightning(\varepsilon + \varepsilon') \dashv\vdash \lightning(\varepsilon) * \lightning(\varepsilon')$$

$$\varepsilon_i = \varepsilon_{i+1} + i/N$$

Approximate Refinement: Error Resource Management

Every invocation of the oracles preserves the invariant.

- Increment counters: $\text{counter} \mapsto i{+}1 \;*\; \text{counter}' \mapsto_s i{+}1 \;*\; \lightning(\varepsilon_i)$
- Split credits: $\lightning(\varepsilon_i) \;\; \multimap\!\!* \;\; \lightning(\varepsilon_{i+1}) \;*\; \lightning(i/N)$
- Spend $\lightning(\varepsilon_{i+1})$ credits to re-establish invariant with $i+1$

$$\frac{\text{counter} \mapsto i \qquad \text{counter} \mapsto i \;\multimap\!\!*\; \text{rwp}\; i \precsim e \;\{\Phi\}}{\text{rwp}\;\; !\,\text{counter} \precsim e \;\{\Phi\}}$$

$$\lightning(\varepsilon + \varepsilon') \dashv\vdash \lightning(\varepsilon) \;*\; \lightning(\varepsilon')$$

$$\varepsilon_i = \varepsilon_{i+1} + i/N$$

Approximate Refinement: Error Resource Management

Every invocation of the oracles preserves the invariant.

- Increment counters: $\text{counter} \mapsto i{+}1 \;*\; \text{counter}' \mapsto_s i{+}1 \;*\; \lightning(\varepsilon_i)$
- Split credits: $\lightning(\varepsilon_i) \;\; \multimap\!\!* \;\; \lightning(\varepsilon_{i+1}) \;*\; \lightning(i/N)$
- Spend $\lightning(\varepsilon_{i+1})$ credits to re-establish invariant with $i+1$
- Spend $\lightning(i/N)$ credits to sample a fresh nonce $n \notin \mathit{nonces}$

$$\frac{\text{counter} \mapsto i \qquad \text{counter} \mapsto i \;\multimap\!\!*\; \text{rwp}\; i \precsim e\; \{\Phi\}}{\text{rwp}\;\; !\,\text{counter} \precsim e\; \{\Phi\}}$$

$$\lightning(\varepsilon + \varepsilon') \dashv\vdash \lightning(\varepsilon) \;*\; \lightning(\varepsilon')$$

$$\varepsilon_i = \varepsilon_{i+1} + i/N$$

Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \mathrel{-\!\!*} \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$

Approximate Refinement: Errors Credits for Random Sampling

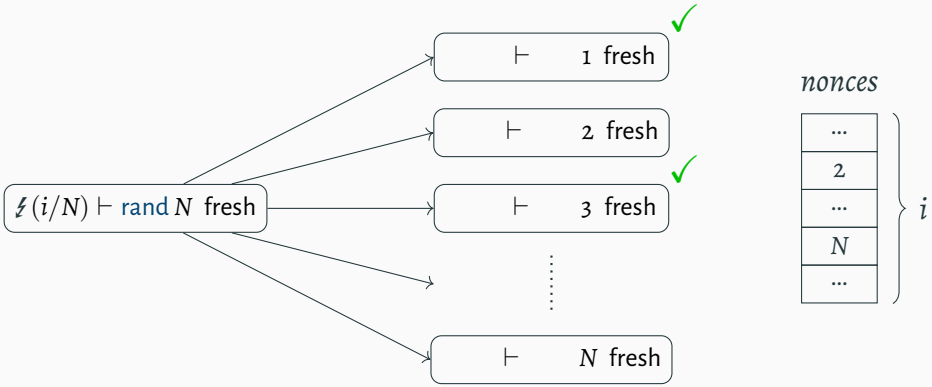
$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \mathrel{-\!\!*} \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$

Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \mathrel{-\!\!*} \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$

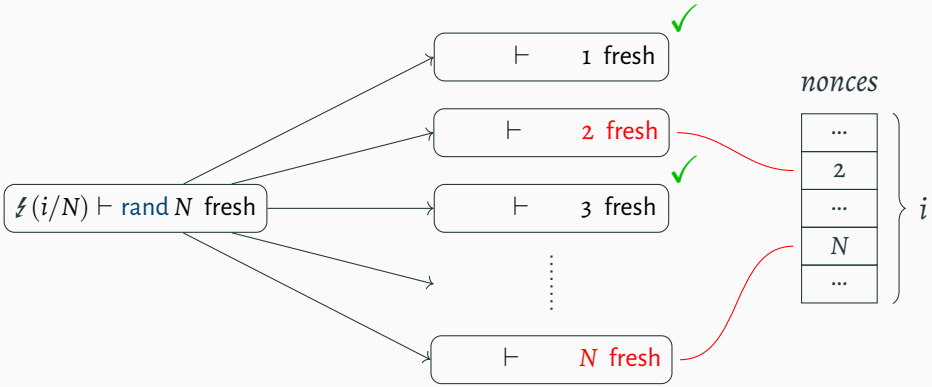
Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \twoheadrightarrow \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$



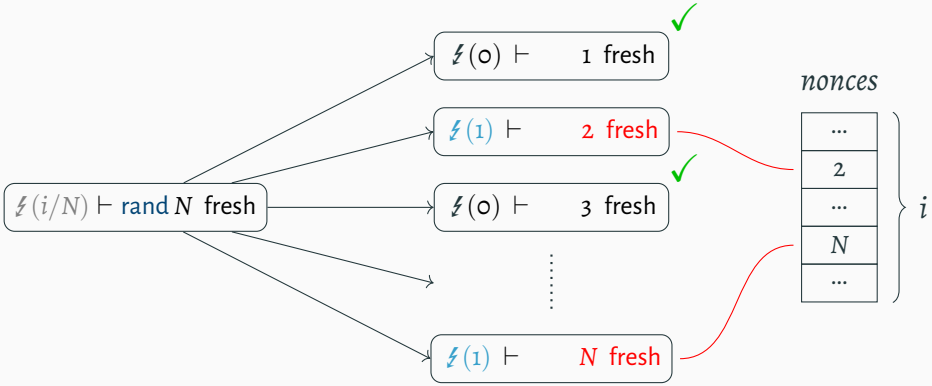
Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \twoheadrightarrow \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$



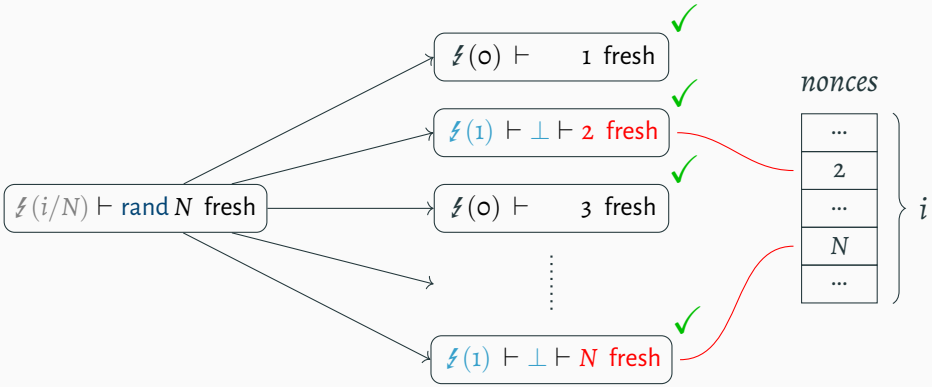
Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \twoheadrightarrow \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$



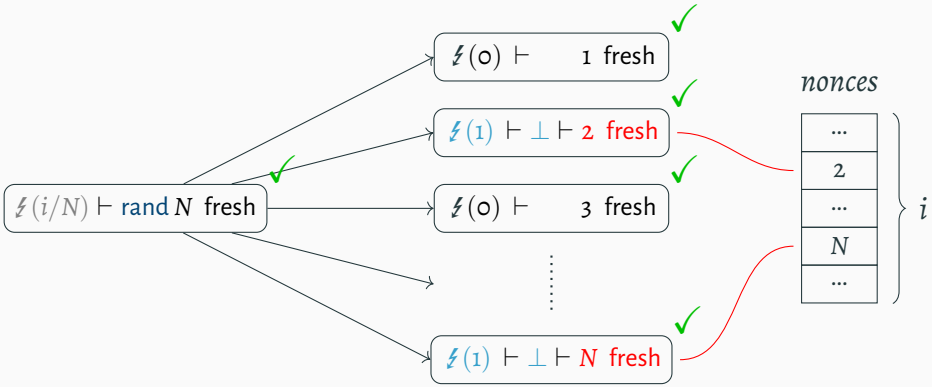
Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \text{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \twoheadrightarrow \text{rwp}\ n \precsim n\ \{\phi\}}{\text{rwp}\ \ \text{rand}\, N \precsim \text{rand}\, N\ \{\phi\}}$$



Approximate Refinement: Errors Credits for Random Sampling

$$\frac{↯(i/N) \qquad i = \mathsf{length}(\mathit{nonces}) \qquad \forall n < N.\, n \notin \mathit{nonces} \twoheadrightarrow \mathsf{rwp}\ n \precsim n\ \{\phi\}}{\mathsf{rwp}\ \mathsf{rand}\, N \precsim \mathsf{rand}\, N\ \{\phi\}}$$



Refinement via Logical Relation

$$\frac{}{\mathsf{rwp}\ \mathcal{A} \precsim \mathcal{A}\ \{\psi\}}$$

- Code of $\mathcal{A}$ unknown

Refinement via Logical Relation

$$\frac{}{\mathsf{rwp}\ \mathcal{A} \precsim \mathcal{A}\ \{\psi\}}$$

- Code of $\mathcal{A}$ unknown
- Language property: all well-typed programs respect abstraction

Refinement via Logical Relation

$$\frac{\vDash \mathcal{A} \precsim \mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}}{\mathsf{rwp}\ \mathcal{A} \precsim \mathcal{A}\ \{\psi\}}$$

- Code of $\mathcal{A}$ unknown
- Language property: all well-typed programs respect abstraction
- Captured via a type-indexed logical relation: $\vDash e_1 \precsim e_2 : \tau$

Refinement via Logical Relation

$$\frac{\vDash \mathcal{A} \precsim \mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}}{\mathsf{rwp}\ \mathcal{A} \precsim \mathcal{A}\ \{\psi\}}$$

- Code of $\mathcal{A}$ unknown
- Language property: all well-typed programs respect abstraction
- Captured via a type-indexed logical relation: $\vDash e_1 \precsim e_2 : \tau$
- Logical relations model of the type system defined in terms of refinement

Refinement via Logical Relation

$$\frac{\vDash \mathcal{A} \precsim \mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}}{\mathsf{rwp}\ \ \mathcal{A} \precsim \mathcal{A}\ \{[\![(\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}]\!]\}}$$

- Code of $\mathcal{A}$ unknown
- Language property: all well-typed programs respect abstraction
- Captured via a type-indexed logical relation: $\quad \vDash e_1 \precsim e_2 : \tau$
- Logical relations model of the type system defined in terms of refinement
- **Derive a rwp for $\mathcal{A}$ purely from its type signature**

  $\mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}$

Summary: Approximate Refinement

- Approximate refinement of randomized programs

$$\lightning(\varepsilon) \multimap * \ \mathsf{rwp}\ \ \mathsf{encrypt}^{k}_{|\mathsf{Q}} \precsim \mathsf{random}_{|\mathsf{Q}}\ \ \{\phi\}$$

Summary: Approximate Refinement

- Approximate refinement of randomized programs

$$\lightning(\varepsilon) \multimap * \ \mathsf{rwp}\ \ \mathsf{encrypt}^{k}_{|\mathbb{Q}} \precsim \mathsf{random}_{|\mathbb{Q}}\ \ \{\phi\}$$

- Typed logical relation for unknown code

$$\vDash \mathcal{A} \precsim \mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}$$

Summary: Approximate Refinement

- Approximate refinement of randomized programs

$$\lightning(\varepsilon) \multimap * \ \mathsf{rwp}\ \ \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q}\ \{\phi\}$$

- Typed logical relation for unknown code

$$\vDash \mathcal{A} \precsim \mathcal{A} : (\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool}$$

- By the adequacy theorem: encryption is secure

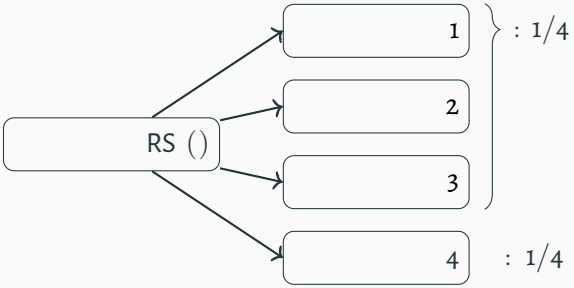
$$\Pr\left[\mathcal{A}(\mathsf{encrypt}^k_{|Q}) \Downarrow \mathsf{true}\right] = \Pr\left[\mathcal{A}(\mathsf{random}_{|Q}) \Downarrow \mathsf{true}\right] \pm \varepsilon$$

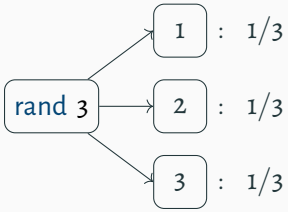
Exact Contextual Refinement by Approximation in the Limit

$$\vDash \quad \mathsf{rec}\,\mathsf{RS}\ \mathsf{x} = \mathsf{let}\,\mathsf{k} = \mathsf{rand}\,4\,\mathsf{in}\ \mathsf{if}\ \mathsf{k} < 4\ \mathsf{then}\ \mathsf{k}\ \mathsf{else}\ \mathsf{RS}\,() \quad \precsim \quad \mathsf{rand}\,3 : \mathsf{int}$$

Exact Contextual Refinement by Approximation in the Limit

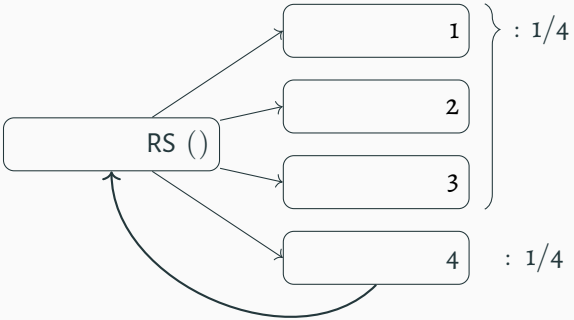
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \\ \text{if k} < 4 \text{ then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

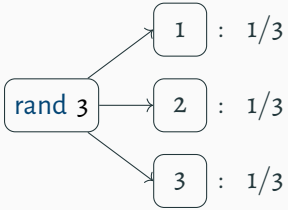




Exact Contextual Refinement by Approximation in the Limit

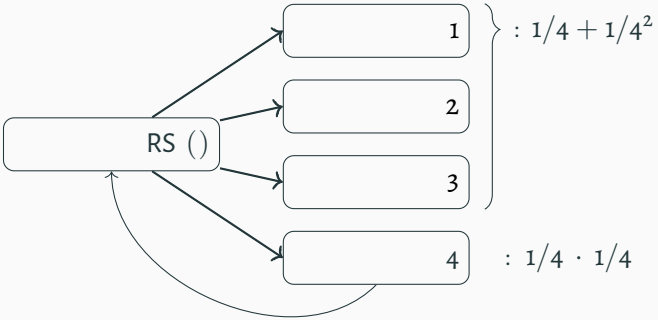
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \\ \text{if k} < 4 \text{ then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

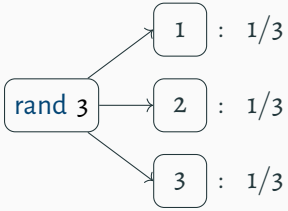




Exact Contextual Refinement by Approximation in the Limit

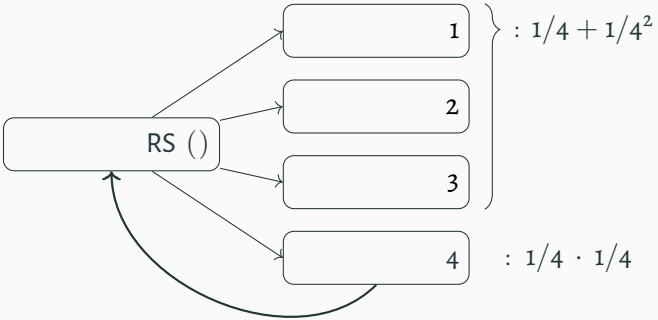
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \quad \text{if k} < 4 \text{ then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

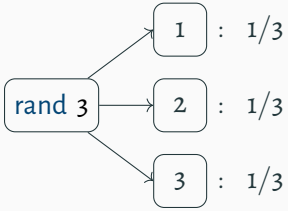




Exact Contextual Refinement by Approximation in the Limit

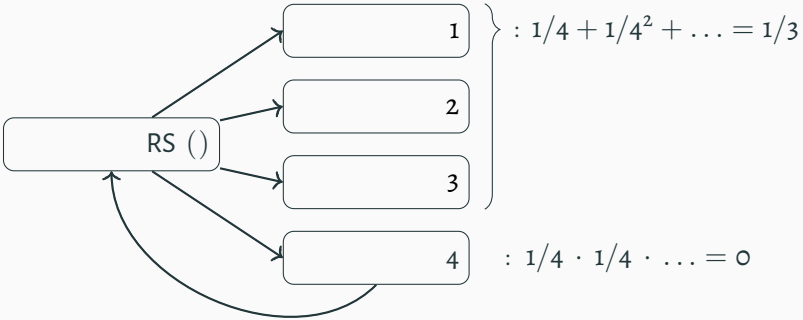
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \quad \text{if k} < 4 \text{ then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

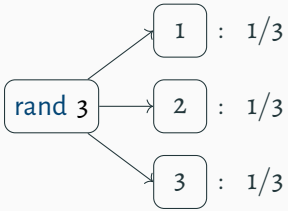




Exact Contextual Refinement by Approximation in the Limit

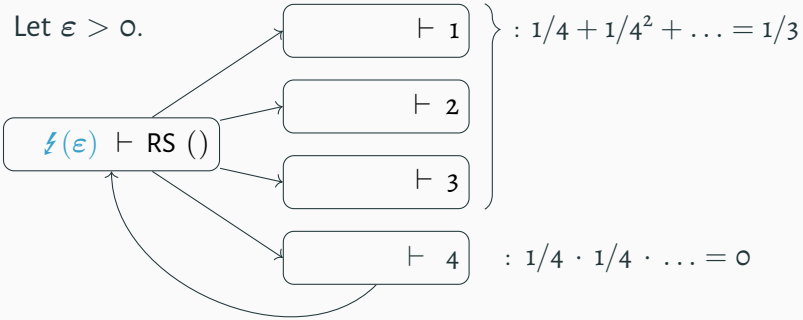
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \\ \text{if k} < 4 \text{ then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

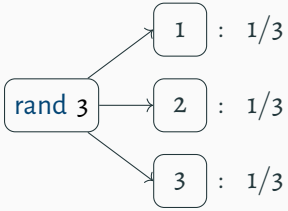




Exact Contextual Refinement by Approximation in the Limit

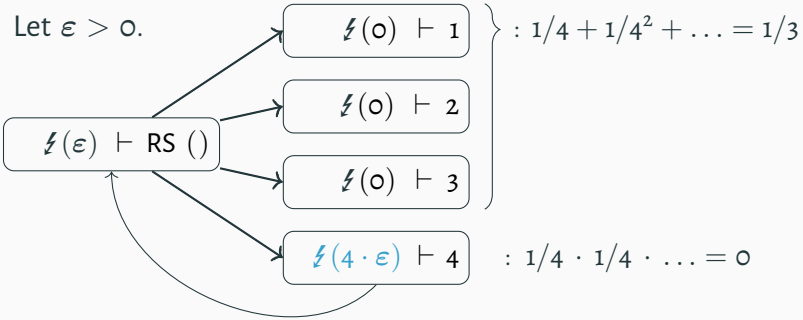
$$\vDash \quad \mathsf{rec}\ \mathsf{RS}\ \mathsf{x} = \mathsf{let}\ \mathsf{k} = \mathsf{rand}\ 4\ \mathsf{in}\ \mathsf{if}\ \mathsf{k} < 4\ \mathsf{then}\ \mathsf{k}\ \mathsf{else}\ \mathsf{RS}\,() \quad \precsim \quad \mathsf{rand}\ 3 \ : \ \mathsf{int}$$

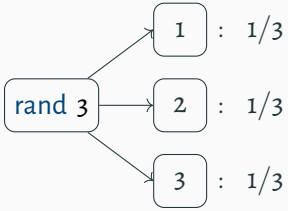




Exact Contextual Refinement by Approximation in the Limit

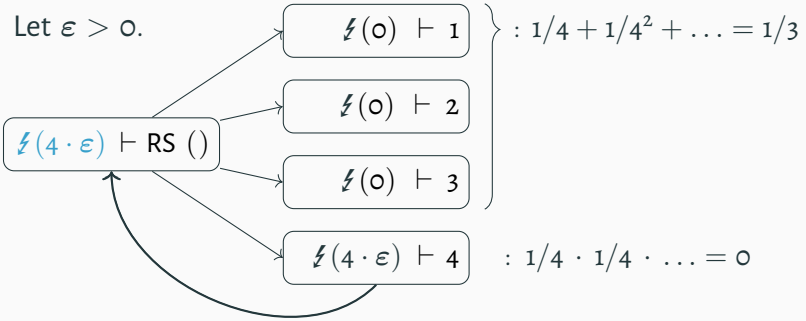
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \; \text{if k} < 4 \; \text{then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

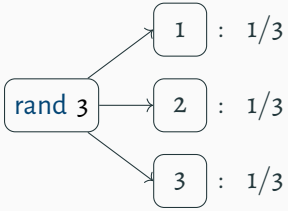




Exact Contextual Refinement by Approximation in the Limit

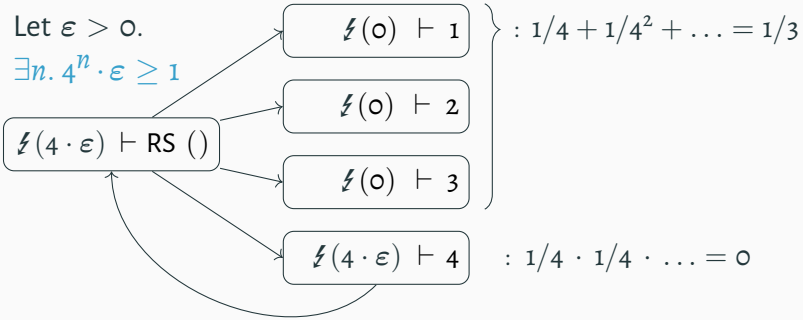
$$\vDash \quad \texttt{rec RS x} = \texttt{let k} = \texttt{rand 4 in}\ \texttt{if k} < 4\ \texttt{then k else RS ()} \quad \precsim \quad \texttt{rand 3} : \texttt{int}$$

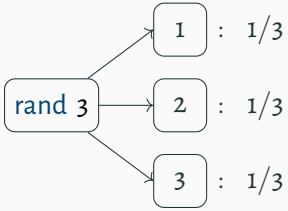




Exact Contextual Refinement by Approximation in the Limit

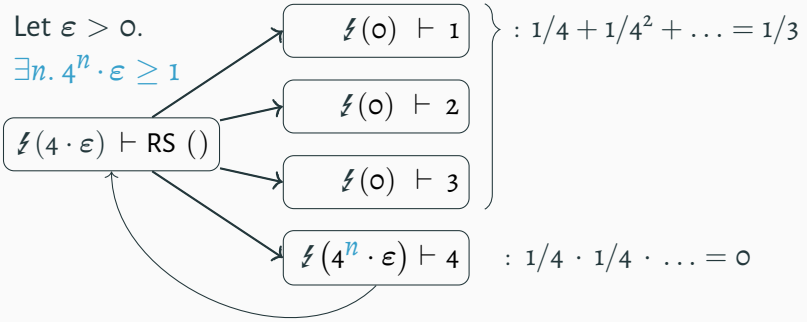
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \; \text{if k} < 4 \; \text{then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

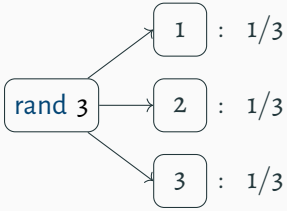




Exact Contextual Refinement by Approximation in the Limit

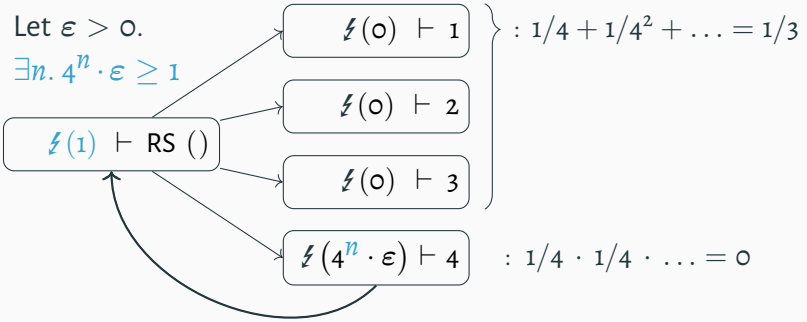
$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \; \text{if k} < 4 \; \text{then k else RS ()} \quad \precsim \quad \text{rand 3} : \text{int}$$

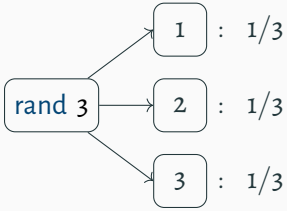




Exact Contextual Refinement by Approximation in the Limit

$$\vDash \quad \text{rec RS x} = \text{let k} = \text{rand 4 in} \; \text{if k} < 4 \; \text{then k else RS ()} \quad \precsim \quad \text{rand 3} \; : \; \text{int}$$





The Bigger Picture

Related projects:

- Clutch: refinement, asynchronous sampling via tapes
- Eris: unary, bound probability of "bad events" via error credits
- Caliper: termination-preserving refinement
- Tachis: expected cost (e.g., time or entropy) via cost credits

The Bigger Picture

Related projects:

- Clutch: refinement, asynchronous sampling via tapes
- Eris: unary, bound probability of "bad events" via error credits
- Caliper: termination-preserving refinement
- Tachis: expected cost (e.g., time or entropy) via cost credits

Ongoing work:

- Concurrency
- Continuous distributions
- Differential privacy

The Bigger Picture

Related projects:

- Clutch: refinement, asynchronous sampling via tapes
- Eris: unary, bound probability of "bad events" via error credits
- Caliper: termination-preserving refinement
- Tachis: expected cost (e.g., time or entropy) via cost credits

Ongoing work:

- Concurrency
- Continuous distributions
- Differential privacy

Future work:

- Distributed systems
- Tail bounds for complexity
- Realistic languages & cross-language reasoning
- Unifying framework, more security applications, ...

Summary

- **Approxis**: higher-order approximate refinement separation logic
- **Typed logical relation** for unknown code & limiting arguments
- Fully mechanized in Rocq and Iris
- More in the paper:
  - applications, *e.g.*, rejection samplers for data structures (B+ trees from DBs/FSs)
  - semantic model (based on new coupling laws)

Thank you!

E-mail pgh@cs.au.dk

Website github.com/logsem/clutch

Encore

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

Program refinement: $e_1 \precsim e_2 \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \mathsf{v}\ ] \leq \Pr[\mathcal{C}[e_2] \Downarrow \mathsf{v}\ ]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

Contextual refinement: $\mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \mathsf{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \mathsf{true}]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

Refinement Logic: $\vdash \mathsf{rwp}\ \mathcal{C}[e_1] \precsim \mathcal{C}[e_2]\ \{[\![\mathsf{bool}]\!]\}$

$\Downarrow$

Contextual refinement: $\mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \mathsf{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \mathsf{true}]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

Logical relation: $\models \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] : \text{bool}$

$\Downarrow$

Refinement Logic: $\vdash \text{rwp}\ \mathcal{C}[e_1] \precsim \mathcal{C}[e_2]\ \{[\![\text{bool}]\!]\}$

$\Downarrow$

Contextual refinement: $\mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \text{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \text{true}]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = \circ$:

$$\Downarrow \text{ stable under typed ctx } (\mathcal{C} : \tau \Rightarrow \text{bool})$$

Logical relation: $\quad \vDash \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] : \text{bool}$

$$\Downarrow$$

Refinement Logic: $\quad \vdash \text{rwp } \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \ \{[\![\text{bool}]\!]\}$

$$\Downarrow$$

Contextual refinement: $\quad \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \text{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \text{true}]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

$$\vDash e_1 \precsim e_2 : \tau$$

$$\Downarrow \text{ stable under typed ctx } (\mathcal{C} : \tau \Rightarrow \text{bool})$$

Logical relation: $\vDash \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] : \text{bool}$

$$\Downarrow$$

Refinement Logic: $\vdash \text{rwp } \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \ \{[\![\text{bool}]\!]\}$

$$\Downarrow$$

Contextual refinement: $\mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \text{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \text{true}]$

Methodology for Contextual Refinement

Consider the case $\varepsilon = 0$:

$$\forall \varepsilon' > 0.\ \lightning(\varepsilon') \vDash e_1 \precsim e_2 : \tau \quad \text{Can approximate via "error induction"!}$$

$$\Downarrow \text{ stable under typed ctx } (\mathcal{C} : \tau \Rightarrow \text{bool})$$

Logical relation: $\quad \vDash \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] : \text{bool}$

$$\Downarrow$$

Refinement Logic: $\quad \vdash \text{rwp } \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \ \{[\![\text{bool}]\!]\}$

$$\Downarrow$$

Contextual refinement: $\quad \mathcal{C}[e_1] \precsim \mathcal{C}[e_2] \triangleq \Pr[\mathcal{C}[e_1] \Downarrow \text{true}] \leq \Pr[\mathcal{C}[e_2] \Downarrow \text{true}]$

IND$-CPA Security in Approxis

Logical Relation: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\mathsf{random}_{|Q}) : \mathsf{bool}$

$\Downarrow$

Refinement Logic: $\lightning(\varepsilon) \vdash \mathsf{rwp}\ \mathcal{A}\,(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\mathsf{random}_{|Q})\ \ \{b_1\ b_2\ .\ b_1 = b_2\}$

$\Downarrow$

Approximation: $\Pr\left[\mathcal{A}\,\mathsf{encrypt}^k_{|Q} \Downarrow \mathsf{true}\right] \leq \Pr\left[\mathcal{A}(\mathsf{random}_{|Q}) \Downarrow \mathsf{true}\right] + \varepsilon$

IND$-CPA proof I: Logical Relation

Let $\tau_{\mathcal{A}} \triangleq ((\mathsf{msg} \to \mathsf{cipher\,option}) \to \mathsf{bool})$ and assume $\vdash \mathcal{A} : \tau_{\mathcal{A}}$.

- To show: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\mathsf{random}_{|Q}) : \mathsf{bool}$

IND$-CPA proof I: Logical Relation

Let $\tau_{\mathcal{A}} \triangleq ((\mathsf{msg} \to \mathsf{cipher\,option}) \to \mathsf{bool})$ and assume $\vdash \mathcal{A} : \tau_{\mathcal{A}}$.

- To show: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\mathsf{encrypt}^{k}_{|Q}) \precsim \mathcal{A}\,(\mathsf{random}_{|Q}) : \mathsf{bool}$
- By Fundamental Lemma of the logical relation, we get $\vDash \mathcal{A} \precsim \mathcal{A} : \tau_{\mathcal{A}}$

IND$-CPA proof I: Logical Relation

Let $\tau_{\mathcal{A}} \triangleq ((\text{msg} \to \text{cipher option}) \to \text{bool})$ and assume $\vdash \mathcal{A} : \tau_{\mathcal{A}}$.

- To show: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\text{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\text{random}_{|Q}) : \text{bool}$
- By Fundamental Lemma of the logical relation, we get $\vDash \mathcal{A} \precsim \mathcal{A} : \tau_{\mathcal{A}}$
- By the rule for application, we can "drop" $\mathcal{A}$ and focus on the oracles :

$$\lightning(\varepsilon) \vDash \text{encrypt}^k_{|Q} \precsim \text{random}_{|Q} : \text{msg} \to \text{cipher option}$$

IND$-CPA proof I: Logical Relation

Let $\tau_{\mathcal{A}} \triangleq ((\text{msg} \to \text{cipher option}) \to \text{bool})$ and assume $\vdash \mathcal{A} : \tau_{\mathcal{A}}$.

- To show: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\text{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\text{random}_{|Q}) : \text{bool}$
- By Fundamental Lemma of the logical relation, we get $\vDash \mathcal{A} \precsim \mathcal{A} : \tau_{\mathcal{A}}$
- By the rule for application, we can "drop" $\mathcal{A}$ and focus on the oracles :

$$\lightning(\varepsilon) \vDash \text{encrypt}^k_{|Q} \precsim \text{random}_{|Q} : \text{msg} \to \text{cipher option}$$

- The oracles satisfy an *invariant*. Initially, $i = 0$ and $\varepsilon = \sum_{k=0}^{Q-1} \frac{k}{2N} = \frac{(Q-1)Q}{2N}$.

$$Inv \triangleq \exists i.\ \text{counter} \mapsto i \ *\ \text{counter}' \mapsto_s i \ *\ \lightning\left(\frac{(Q-1)Q}{2N} - \frac{(i-1)i}{2N}\right)$$

IND$-CPA proof I: Logical Relation

Let $\tau_{\mathcal{A}} \triangleq ((\mathsf{msg} \to \mathsf{cipher\ option}) \to \mathsf{bool})$ and assume $\vdash \mathcal{A} : \tau_{\mathcal{A}}$.

- To show: $\lightning(\varepsilon) \vDash \mathcal{A}\,(\mathsf{encrypt}^k_{|Q}) \precsim \mathcal{A}\,(\mathsf{random}_{|Q}) : \mathsf{bool}$
- By Fundamental Lemma of the logical relation, we get $\vDash \mathcal{A} \precsim \mathcal{A} : \tau_{\mathcal{A}}$
- By the rule for application, we can "drop" $\mathcal{A}$ and focus on the oracles:

$$\lightning(\varepsilon) \vDash \mathsf{encrypt}^k_{|Q} \precsim \mathsf{random}_{|Q} : \mathsf{msg} \to \mathsf{cipher\ option}$$

- The oracles satisfy an *invariant*. Initially, $i = 0$ and $\varepsilon = \sum_{k=0}^{Q-1} \frac{k}{2N} = \frac{(Q-1)Q}{2N}$.

$$Inv \triangleq \exists i.\ \mathsf{counter} \mapsto i \ * \ \mathsf{counter}' \mapsto_s i \ * \ \lightning\left(\frac{(Q-1)Q}{2N} - \frac{(i-1)i}{2N}\right)$$

- By the rule for functions, show refinement for an argument $(\mathsf{m} : \mathsf{msg})$

$$\boxed{Inv}^{\gamma} \vDash \mathsf{encrypt}^k_{|Q}\ \mathsf{m} \precsim \mathsf{random}_{|Q}\ \mathsf{m} : \mathsf{cipher\ option}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- No more unknown code, a program at ground type

$$\boxed{Inv}^{\gamma} \vDash \mathsf{encrypt}^{k}_{|\mathcal{Q}}\ \mathsf{m} \precsim \mathsf{random}_{|\mathcal{Q}}\ \mathsf{m} : \mathsf{cipher\ option}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- No more unknown code, a program at ground type

$$\boxed{Inv}^{\gamma} \vDash \mathsf{encrypt}^{k}_{|Q}\ \mathsf{m} \precsim \mathsf{random}_{|Q}\ \mathsf{m} : \mathsf{cipher\ option}$$

- Switch to rwp, open invariant, symbolically execute:

$$Inv_i \vdash \mathsf{rwp} \begin{pmatrix} \mathsf{if}\ !\,\mathsf{counter} < Q\ \mathsf{then} \\ \quad \mathsf{incr\ counter}\,; \\ \quad \mathsf{Some}\,(\mathsf{encrypt}\,k\,\mathsf{m}) \\ \mathsf{else\ None} \end{pmatrix} \precsim \begin{pmatrix} \mathsf{if}\ !\,\mathsf{counter}' < Q\ \mathsf{then} \\ \quad \mathsf{incr\ counter}'\,; \\ \quad \mathsf{Some}\,(\mathsf{random\ m}) \\ \mathsf{else\ None} \end{pmatrix} \{\![\![\mathsf{cipher\ option}]\!]\!\}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- Switch to rwp, open invariant, symbolically execute:

$$Inv_i \vdash \text{rwp} \begin{pmatrix} \text{if } !\,\text{counter} < Q \text{ then} \\ \quad \text{incr counter} ; \\ \quad \text{Some (encrypt } k \text{ m)} \\ \text{else None} \end{pmatrix} \precsim \begin{pmatrix} \text{if } !\,\text{counter}' < Q \text{ then} \\ \quad \text{incr counter}' ; \\ \quad \text{Some (random m)} \\ \text{else None} \end{pmatrix} \{[\![\text{cipher option}]\!]\}$$

- Open invariant, case on the if-statement, compute:

$$\ldots * \lightning(\varepsilon_i) \vdash \text{rwp}_{\top \setminus \gamma} \begin{pmatrix} \text{let r} = \text{rand } N \text{ in} \\ \text{let nonce} = prf\ k\ \text{r in} \\ \text{let c} = \text{xor m nonce in} \\ (\text{r, c}) \end{pmatrix} \precsim \begin{pmatrix} \text{let r} = \text{rand } N \text{ in} \\ \\ \text{let c} = \text{rand } N \text{ in} \\ (\text{r, c}) \end{pmatrix} \{[\![\text{cipher option}]\!]\}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- Open invariant, case on the if-statement, compute:

$$\ldots * \lightning(\varepsilon_i) \vdash \mathsf{rwp}_{\top\setminus\gamma} \begin{pmatrix} \mathsf{let\ r} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ nonce} = \mathit{prf}\ k\ \mathsf{r\ in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \mathsf{let\ r} = \mathsf{rand}\ N\ \mathsf{in} \\ \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

- Spend some error credits on sampling a fresh nonce

$$\frac{\lightning\left(\frac{\mathsf{length}(l)}{N+1}\right) \qquad \forall n \leq N.\, n \notin l \mathrel{-\!\!*} \mathsf{rwp}\ \ n \precsim n\ \{\Phi\}}{\mathsf{rwp}\ \ \mathsf{rand}\ N \precsim \mathsf{rand}\ N\ \{\Phi\}}$$

where l is the list of previously generated nonces and $\mathsf{length}(l) = i$.

IND$-CPA proof II: refinement weakes-pre with error credits

- Open invariant, case on the if-statement, compute:

$$\ldots * \lightning(\varepsilon_i) \vdash \mathsf{rwp}_{\top\setminus\gamma} \begin{pmatrix} \mathsf{let\ r} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ nonce} = \mathit{prf}\ k\ \mathsf{r\ in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \mathsf{let\ r} = \mathsf{rand}\ N\ \mathsf{in} \\ \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

- Spend some error credits on sampling a fresh nonce, close invariant:

$$\mathit{Inv}_{i+1} * \mathsf{r\ fresh} \vdash \mathsf{rwp}_{\top\setminus\gamma} \begin{pmatrix} \mathsf{let\ nonce} = \mathit{prf}\ k\ \mathsf{r\ in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- Spend some error credits on sampling a fresh nonce, close invariant:

$$Inv_{i+1} * \mathsf{r\ fresh} \vdash \mathsf{rwp}_{\top \setminus \gamma} \begin{pmatrix} \mathsf{let\ nonce} = prf\ k\ \mathsf{r\ in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{\llbracket \mathsf{cipher\ option} \rrbracket\}$$

- PRF assumption: $prf\ k\ \mathsf{r}$ behaves randomly if r is fresh!

$$\mathsf{r\ fresh} \vdash \mathsf{rwp} \begin{pmatrix} \mathsf{let\ nonce} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{\llbracket \mathsf{cipher\ option} \rrbracket\}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- PRF assumption: $prf\ k\ \mathsf{r}$ behaves randomly if r is fresh!

$$\mathsf{r\ fresh} \vdash \mathsf{rwp} \begin{pmatrix} \mathsf{let\ nonce} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

- xor acts as one-time pad because nonce is random:

$$Inv_{i+1} \vdash \mathsf{rwp}_{\top \setminus \gamma} \begin{pmatrix} \mathsf{let\ nonce} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \precsim \begin{pmatrix} \\ \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

IND$-CPA proof II: refinement weakes-pre with error credits

- xor acts as one-time pad because nonce is random:

$$Inv_{i+1} \vdash \mathsf{rwp}_{\top \setminus \gamma} \begin{pmatrix} \mathsf{let\ nonce} = \mathsf{rand}\ N\ \mathsf{in} \\ \mathsf{let\ c} = \mathsf{xor\ m\ nonce\ in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \preceq \begin{pmatrix} \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

- done:

$$\vdash \mathsf{rwp} \begin{pmatrix} \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \preceq \begin{pmatrix} \mathsf{let\ c} = \mathsf{rand}\ N\ \mathsf{in} \\ (\mathsf{r}, \mathsf{c}) \end{pmatrix} \{[\![\mathsf{cipher\ option}]\!]\}$$

Overview: Reasoning about Known and Unknown Code

"(Observable) behaviours of e_1 are *included* in those of e_2."

Program refinement: $e_1 \precsim e_2$

Overview: Reasoning about Known and Unknown Code

"(Observable) behaviours of e_1 are *included* in those of e_2."

Program refinement: $e_1 \precsim e_2 \triangleq \forall(\sigma : State)(v : Val).\ \Pr[(e_1, \sigma) \Downarrow v] \leq \Pr[(e_2, \sigma) \Downarrow v]$

Overview: Reasoning about Known and Unknown Code

"(Observable) behaviours of e_1 are *included* in those of e_2, up to probability ε."

Program approximation: $e_1 \precsim^{\varepsilon} e_2 \triangleq \forall(\sigma : State)(v : Val).\ \Pr[(e_1, \sigma) \Downarrow v] \leq \Pr[(e_2, \sigma) \Downarrow v] + \varepsilon$

Overview: Reasoning about Known and Unknown Code

Refinement Logic: $\lightning(\varepsilon) \vdash \text{rwp}\ e_1 \precsim e_2\ \{\phi\}$

$\Downarrow$ Adequacy

Program approximation: $e_1 \precsim^{\varepsilon} e_2 \triangleq \forall(\sigma : \mathit{State})(v : \mathit{Val}).\ \Pr[(e_1, \sigma) \Downarrow v] \leq \Pr[(e_2, \sigma) \Downarrow v] + \varepsilon$

Overview: Reasoning about Known and Unknown Code

Typed Logical Relation: $\lightning(\varepsilon) \vDash e_1 \precsim e_2 : \tau$

$\Downarrow$ Congruence

Refinement Logic: $\lightning(\varepsilon) \vdash \mathsf{rwp}\ e_1 \precsim e_2\ \{\phi\}$

$\Downarrow$ Adequacy

Program approximation: $e_1 \precsim^{\varepsilon} e_2 \triangleq \forall(\sigma : \mathit{State})(v : \mathit{Val}).\ \Pr[(e_1, \sigma) \Downarrow v] \leq \Pr[(e_2, \sigma) \Downarrow v] + \varepsilon$