ASYNCHRONOUS PROBABILISTIC COUPLINGS in Higher-Order Separation Logic

Simon Gregersen  Alejandro Aguirre  Philipp Haselwarter
Joseph Tassarotti  Lars Birkedal

alejandro@cs.au.dk

May 23, 2023
We consider a real and an ideal implementation of the OTP encryption

\[
\text{real} \triangleq \lambda (m : \text{bool}). \text{let } k = \text{flip in } (k \text{ xor } m) \\
\text{ideal} \triangleq \lambda (m : \text{bool}). \text{flip}
\]

Any adversary \(A\) should not be able to distinguish \(\text{real}\) from \(\text{ideal}\), i.e.

\[ A(\text{real})' = A(\text{ideal}). \]

This is captured by contextual equivalence.
One-time pad

We consider a real and an ideal implementation of the OTP encryption

\[
\text{real} \triangleq \lambda(m : \text{bool}). \text{let } k = \text{flip} \text{ in } (k \text{ xor } m)
\]

\[
\text{ideal} \triangleq \lambda(m : \text{bool}). \text{flip}
\]

Any adversary \(A\) should not be able to distinguish real from ideal, i.e.

\[
\forall A. A(\text{real}) \sim A(\text{ideal}).
\]
We consider a real and an ideal implementation of the OTP encryption

\[
\text{real} \triangleq \lambda (m: \text{bool}). \text{let } k = \text{flip in } (k \text{ xor } m) \\
\text{ideal} \triangleq \lambda (m: \text{bool}). \text{flip}
\]

Any adversary \( \mathcal{A} \) should not be able to distinguish real from ideal, i.e.
\[\forall \mathcal{A}. \mathcal{A}(\text{real}) \simeq \mathcal{A}(\text{ideal}).\]

This is captured by contextual equivalence
Contextual equivalence

Two programs are contextually equivalent if they have the same behavior under any context.
Contextual equivalence

Two programs are contextually equivalent if they have the same behavior under any context.

Often hard to reason about directly, instead we use a logical relation.

$$\models e \preceq e' : \tau$$
Two programs are contextually equivalent if they have the same behavior under any context.

Often hard to reason about directly, instead we use a logical relation.

\[ \vdash e \preceq e' : \tau \]

Multiple examples of this in Iris, e.g. ReLoC
In this work we develop Clutch\(^1\), which consists of

- A **probabilistic operational semantics** for sequential probabilistic languages
- A **unary coupling-based WP** to prove relations between probabilistic programs
- A **ReLoC-style logical relation** to prove contextual refinement of probabilistic programs
- A **ghost resource** to reason about samples that happen in the future

\(^1\)https://github.com/logsem/clutch
# Structure of Clutch

## ReLoC

<table>
<thead>
<tr>
<th>Logical Refinement</th>
<th>Type Interpretation</th>
<th>HeapLang WP Rules</th>
<th>HeapLang</th>
<th>Iris WP</th>
<th>Iris Base Language</th>
<th>Iris Base Logic</th>
</tr>
</thead>
</table>

## Clutch

<table>
<thead>
<tr>
<th>Logical Refinement</th>
<th>Type Interpretation</th>
<th>WP Rules</th>
<th>Base Language</th>
<th>Base Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris + Clutch</td>
<td>Iris + Clutch</td>
<td>$F_{\mu,ref}^\text{rand}$</td>
<td>Clutch</td>
<td>Iris</td>
</tr>
</tbody>
</table>

Note: The table represents the structure of Clutch, showing the integration of ReLoC and Clutch at both logical refinement and type interpretation levels, along with the associated WP rules and base languages.
A proof in Clutch

\[ \forall m. \text{let } k = \text{flip in } (k \text{ xor } m) \preceq \lambda m. \text{flip} : \text{bool} \rightarrow \text{bool} \]
A proof in Clutch

\[ m : \text{bool} \models \text{let } k = \text{flip in } (k \text{ xor } m) \preceq \text{flip : bool} \]

\[ \models \lambda m. \text{let } k = \text{flip in } (k \text{ xor } m) \preceq \lambda m. \text{flip : bool} \to \text{bool} \]
A proof in Clutch

\[ f : \mathbb{B} \rightarrow \mathbb{B} \text{ bijection} \]
\[ \forall b : \mathbb{B}. \; \Delta \vdash_{\mathcal{E}} K[b] \preceq K'[f(b)] : \tau \]
\[ \Delta \vdash_{\mathcal{E}} K[\text{flip}] \preceq K'[\text{flip}] : \tau \]  
\[ \text{COUPL} \]

\[ m : \text{bool} \vdash \text{let } k = \text{flip in } (k \text{ xor } m) \preceq \text{flip} : \text{bool} \]
\[ \vdash \lambda m. \text{let } k = \text{flip in } (k \text{ xor } m) \preceq \lambda m. \text{flip} : \text{bool} \rightarrow \text{bool} \]
A proof in Clutch

\[ f : \mathbb{B} \rightarrow \mathbb{B} \text{ bijection} \]
\[
\forall b : \mathbb{B}. \ \Delta \vdash_{E} K[b] \trianglerighteq K'[f(b)] : \tau
\]
\[
\Delta \vdash_{E} K[\text{flip}] \trianglerighteq K'[\text{flip}] : \tau \quad \text{COUPL}
\]

\[ (\cdot \text{xor } m) \text{ bijection} \]

\[
m : \text{bool} \vdash \text{let } k = \text{flip in } (k \\text{xor } m) \trianglerighteq \text{flip} : \text{bool}
\]

\[
\vdash \lambda m. \text{let } k = \text{flip in } (k \\text{xor } m) \trianglerighteq \lambda m. \text{flip} : \text{bool} \rightarrow \text{bool}
\]
A proof in Clutch

\[
f : \mathbb{B} \to \mathbb{B} \text{ bijection} \Rightarrow \\
\forall b : \mathbb{B}. \Delta \vdash_{\mathcal{E}} K[b] \preceq K'[f(b) : \tau] \Rightarrow \\
\Delta \vdash_{\mathcal{E}} K[\text{flip}] \preceq K'[\text{flip}] : \tau \quad \text{COUPL}
\]

(\cdot \text{ xor } m) \text{ bijection} \Rightarrow \\
\begin{align*}
& m, b : \text{bool} \models \text{let } k = b \text{ in } (k \text{ xor } m) \preceq b \text{ xor } m : \text{bool} \\
& m : \text{bool} \models \text{let } k = \text{flip} \text{ in } (k \text{ xor } m) \preceq \text{flip} : \text{bool} \\
& \models \lambda m. \text{let } k = \text{flip} \text{ in } (k \text{ xor } m) \preceq \lambda m. \text{flip} : \text{bool} \to \text{bool}
\end{align*}
A proof in Clutch

\((\cdot \text{xor } m)\) bijection

\[
\begin{align*}
\text{m, b: bool} & \models \text{b xor m} \preceq \text{b xor m : bool} \\
\text{m, b: bool} & \models \text{let } k = \text{b in (k xor m)} \preceq \text{b xor m : bool} \\
\text{m: bool} & \models \text{let } k = \text{flip in (k xor m)} \preceq \text{flip : bool} \\
\models \lambda \text{m. let } k = \text{flip in (k xor m)} \preceq \lambda \text{m. flip : bool} \rightarrow \text{bool}
\end{align*}
\]
Lemma real_ideal_rel :
\[ \forall \text{real} \triangleleft \text{ideal} : \text{lrel_bool} \rightarrow \text{lrel_bool}. \]

Proof.
rel_arrow_val.
iIntros (msg1 msg2) "Hmsg".
rel_pures_l. rel_pures_r.
foldxor.
iDestruct "Hmsg" as "[%b [-> ->]]".
rel_apply (refines_couple_flip_flip (xor_sem b)).
iIntros (k).
rel_pures_l.
foldxor.
iApply xor_xor_sem.
Qed.
Operational semantics of probabilistic languages
A probabilistic sequential language

We introduce $F^{\text{rand}}_{\mu, \text{ref}}$: sequential fragment of HeapLang plus sampling

$$\tau \in \text{Type} ::= \alpha | \text{unit} | \text{bool} | \text{nat} | \text{int} | \tau \times \tau | \tau + \tau | \tau \rightarrow \tau | \forall \alpha. \tau | \exists \alpha. \tau | \mu \alpha. \tau | \text{ref} \tau$$

$$e \in \text{Expr} ::= v | x | e_1(e_2) | \text{if } e \text{ then } e_1 \text{ else } e_2 | \text{fst}(e) | \text{snd}(e) | \text{ref}(e) | ! e | e_1 \leftarrow e_2 | \text{fold } e | \text{unfold } e | \cdots | \text{flip}$$

flip chooses uniformly between true and false.

Our implementation supports discrete uniform sampling as well.
A distribution over a countable type $A$ is a non-negative map $\mu : A \to \mathbb{R}$ such that $\sum_{a \in A} \mu(a) \leq 1$.

Probability distributions have a monadic structure given by:

\[
\text{ret} : A \to \mathcal{D}(A) \\
\text{ret}(a) \triangleq \lambda a'.\text{if } (a = a') \text{ then } 1 \text{ else } 0 \\
\gg : \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) \\
(\mu \gg f)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)
\]
We start from a probabilistic head step reduction $\text{hdStep} : \text{Cfg} \rightarrow D(\text{Cfg})$:

\[
(\lambda x.e) \, v, \sigma \xrightarrow{\text{hd}} e[\!\! v/\!\! x ] \!, \, \sigma \\
\ldots
\]

\[
\text{flip, } \sigma \xrightarrow{\text{hd}} b, \sigma \quad b \in \{\text{true, false}\}
\]
We start from a probabilistic head step reduction $\text{hdStep}: \text{Cfg} \rightarrow \mathcal{D}(\text{Cfg})$:

\[
(\lambda x.e) \, v, \sigma \rightarrow_h^1 e[v/x], \sigma \\
\ldots \\
\text{flip}, \sigma \rightarrow_h^{1/2} b, \sigma \quad b \in \{\text{true, false}\}
\]

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

\[
e, \sigma \rightarrow_p e', \sigma \\
\quad \frac{(K[e], \sigma) \rightarrow_p (K[e'], \sigma')}{(K[e'], \sigma')}
\]
Probabilistic evaluation

We define a “stratified” evaluation and full evaluation as the limit:

\[ \lim_{n \to \infty} \text{exec}_n(e, \sigma) \]

\[ \text{exec}_n(e, \sigma) \triangleq \begin{cases} 
\text{ret } e & \text{if } e \in \text{Val} \\
0 & \text{if } e \in \text{Val} \land n = 0 \\
\text{step}(e, \sigma) \gg \text{exec}_m & \text{if } e \in \text{Val} \land n = m + 1 
\end{cases} \]

By summing over all values, we obtain the probability of termination:

\[ \text{Pterm}(e, \sigma) \triangleq \sum_{v \in \text{Val}} \text{exec}_n(e, \sigma)(v) \]
We define an abstract notion of probabilistic Language, in which `prim_step` is a function.

```
Structure language := Language { 
  expr : Type;
  state : Type;
  (* ... *)
  prim_step : expr → state → distr (expr * state);
  (* ... *)
}.
```
We then lift it into an EctxLanguage

```coq
Structure ectxLanguage := EctxLanguage { (* ... *)
  fill : ectx → expr → expr;
  decomp : expr → ectx * expr;
  head_step : expr → state → distr (expr * state);
  (* ... *)
}. and decompose expressions explicitly

Definition prim_step (e1 : expr Λ) (σ1 : state Λ) : distr (expr Λ * state Λ) :=
  let '(K, e1') := decomp e1 in
  '(e2, σ2) ← head_step e1' σ1; dret (fill K e2, σ2)
```
A coupling-based WP
Couplings 101

Couplings are a construction that allows us to reason relationally about probabilistic programs.
Couplings 101

Couplings are a construction that allows us to reason relationally about probabilistic programs. To construct a coupling between $\mu_1 : \mathcal{D}(A)$, $\mu_2 : \mathcal{D}(B)$:
Couplings 101

Couplings are a construction that allows us to reason relationally about probabilistic programs.
To construct a coupling between $\mu_1 : \mathcal{D}(A), \mu_2 : \mathcal{D}(B)$:

- We pick a way of synchronizing the randomness of the two distributions
Couplings 101

Couplings are a construction that allows us to reason relationally about probabilistic programs.
To construct a coupling between $\mu_1 : \mathcal{D}(A), \mu_2 : \mathcal{D}(B)$:

- We pick a way of synchronizing the randomness of the two distributions
- We ensure that every possible outcome satisfies a particular relation $R : A \rightarrow B \rightarrow \text{Prop}$
Couplings are a construction that allows us to reason relationally about probabilistic programs.

To construct a coupling between $\mu_1 : \mathcal{D}(A)$, $\mu_2 : \mathcal{D}(B)$:

- We pick a way of synchronizing the randomness of the two distributions
- We ensure that every possible outcome satisfies a particular
  
  $R : A \rightarrow B \rightarrow \text{Prop}$

We then say that “$\mu_1$ and $\mu_2$ are coupled by $R$”.

**notation:** $\mu_1 \sim \mu_2 : R$
Couplings 101
Probabilistic Couplings
Probabilistic Couplings
Reasoning with couplings
Reasoning with couplings

Introduction:

\[
\begin{align*}
(a, b) & \in R \\
\text{ret}(a) & \sim \text{ret}(b) : R
\end{align*}
\]
Reasoning with couplings

Introduction:

\[
\begin{align*}
(a, b) \in R & \quad \Rightarrow \quad \text{ret}(a) \sim \text{ret}(b) : R \\
\forall b, (b, f(b)) \in R & \quad \Rightarrow \quad \text{flip} \sim \text{flip} : R
\end{align*}
\]
Reasoning with couplings

Introduction:

\[
\frac{(a, b) \in R}{\text{ret}(a) \sim \text{ret}(b) : R}
\]

\[
\frac{f : \mathbb{B} \to \mathbb{B} \text{ bij.}}{\forall b, (b, f(b)) \in R}
\]

\[
\frac{\text{flip} \sim \text{flip} : R}{\text{flip} \sim \text{flip} : R}
\]

Sequencing:

\[
\frac{\mu_1 \sim \mu_2 : R}{\forall (a, b) \in R. f_1(a) \sim f_2(b) : S}
\]

\[
\frac{\mu_1 \gg f_1 \sim \mu_2 \gg f_2 : S}{\mu_1 \gg f_1 \sim \mu_2 \gg f_2 : S}
\]
Reasoning with couplings

Introduction:

\[
\begin{align*}
(a, b) & \in R \\
\text{ret}(a) & \sim \text{ret}(b) : R \\
\end{align*}
\]

\[
\begin{align*}
f : \mathbb{B} & \rightarrow \mathbb{B} \text{ bij.} \\
\forall b, (b, f(b)) & \in R \\
\text{flip} & \sim \text{flip} : R
\end{align*}
\]

Sequencing:

\[
\begin{align*}
\mu_1 & \sim \mu_2 : R \\
\forall (a, b) & \in R. f_1(a) \sim f_2(b) : S \\
\mu_1 & \gg f_1 \sim \mu_2 \gg f_2 : S
\end{align*}
\]

Elimination:

\[
\begin{align*}
\mu_1 & \sim \mu_2 : (=) \\
\forall x. \mu_1(x) & = \mu_2(x)
\end{align*}
\]
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$\text{wp}_E e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_E \Phi(e_1)) \lor$$

$$(e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow \models_E \emptyset)$$

execCoupl($e_1, \sigma_1, \rho_2$)

$$(\lambda e'_1, \sigma'_1, \rho'_2. \triangleright_{\emptyset} \models_E S(\sigma'_1) \ast G(\rho'_2) \ast \text{wp}_E e'_1 \{ \Phi \}))$$
A coupling-based WP

Our WP couples the execution of the implementation program \( e_1 \) with another specification program whose configuration \( \rho \) is tracked by a specification predicate \( G(\rho) \):

\[
wp_e e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_e \Phi(e_1)) \lor \\
(e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow_e \models_e \emptyset) \\
\text{execCoupl}(e_1, \sigma_1, \rho_2) \\
(\lambda e_1', \sigma_1', \rho_2'. \models_e S(\sigma_1') \ast G(\rho_1') \ast wp_e e_1' \{ \Phi \}))
\]
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$wp_\mathcal{E} e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_\mathcal{E} \Phi(e_1)) \lor$$

$$(e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) * G(\rho_2) \rightarrow \mathcal{E} \models \emptyset)$$

execCoupl($e_1, \sigma_1, \rho_2$)

$$(\lambda e_1', \sigma_1', \rho_2'. \triangleright \emptyset \models_\mathcal{E} S(\sigma_1') * G(\rho_1') * wp_\mathcal{E} e_1' \{ \Phi \}))$$
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$
wp_\mathcal{E} e_1 \{ \Phi \} \equiv (e_1 \in \text{Val} \land \models_\mathcal{E} \Phi(e_1)) \lor \\
(e_1 \not\in \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow \models_\mathcal{E} \emptyset) \\
\text{execCoupl}(e_1, \sigma_1, \rho_2) \\
(\lambda e_1', \sigma_1', \rho_2'. \triangleright \models_\mathcal{E} S(\sigma_1') \ast G(\rho_1') \ast wp_\mathcal{E} e_1' \{ \Phi \}))
$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$wp_\mathcal{E} e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_\mathcal{E} \Phi(e_1)) \lor$$

$$\bigl( e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow_\mathcal{E} \emptyset \bigr)$$

$$\text{execCoupl}(e_1, \sigma_1, \rho_2)$$

$$\bigl( \lambda e'_1, \sigma'_1, \rho'_2. \triangleright_\emptyset \models_\mathcal{E} S(\sigma'_1) \ast G(\rho'_1) \ast wp_\mathcal{E} e'_1 \{ \Phi \} \bigr)$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$\text{wp}_\mathcal{E} e_1 \{\Phi\} \triangleq (e_1 \in \text{Val} \land \mathcal{E} \vdash \Phi(e_1)) \lor (e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow \mathcal{E} \not\vdash \emptyset)$$

execCoupl($e_1, \sigma_1, \rho_2$)

$$(\lambda e_1', \sigma_1', \rho_2'. \not\vdash \mathcal{E} S(\sigma_1') \ast G(\rho_1') \ast \text{wp}_\mathcal{E} e_1' \{\Phi\})$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$
wp_\mathcal{E} e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_\mathcal{E} \Phi(e_1)) \lor \\
(e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) * G(\rho_2) \rightarrow_\mathcal{E} \models \emptyset) \\
\text{execCoupl}(e_1, \sigma_1, \rho_2) \\
(\lambda e'_1, \sigma'_1, \rho'_2. \triangleright_\emptyset \models_\mathcal{E} S(\sigma'_1) * G(\rho'_2) * wp_\mathcal{E} e'_1 \{ \Phi \}))
$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$wp_\mathcal{E} e_1 \{\Phi\} \triangleq (e_1 \in \text{Val} \land \mathcal{E} \Phi(e_1)) \lor (e_1 \notin \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow \mathcal{E} \Rightarrow \emptyset)$$

execCoupl($e_1, \sigma_1, \rho_2$)

$$(\lambda e_1', \sigma_1', \rho_1'. \Rightarrow \mathcal{E} S(\sigma_1') \ast G(\rho_1') \ast wp_\mathcal{E} e_1' \{\Phi\}))$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
A coupling-based WP

Our WP couples the execution of the implementation program $e_1$ with another specification program whose configuration $\rho$ is tracked by a specification predicate $G(\rho)$:

$$\text{wp}_\mathcal{E} e_1 \{ \Phi \} \triangleq (e_1 \in \text{Val} \land \models_\mathcal{E} \Phi(e_1)) \lor (e_1 \not\in \text{Val} \land \forall \sigma_1, \rho_2. S(\sigma_1) \ast G(\rho_2) \rightarrow \mathcal{E} \not\models_\emptyset)$$

$$\text{execCoupl}(e_1, \sigma_1, \rho_2)$$

$$\lambda e_1', \sigma_1', \rho_2. \models_\mathcal{E} S(\sigma_1') \ast G(\rho_1') \ast \text{wp}_\mathcal{E} e_1' \{ \Phi \})$$

execCoupl couples every step on the LHS with 0 or more steps on the RHS.
Abstraction through couplings

The coupling-based WP acts as an abstraction layer:

- The postcondition has type $\Phi: \text{Val} \rightarrow \text{iProp}$ (and not $D(\text{Val}) \rightarrow \text{iProp}$)
- No explicit reasoning about probabilities, everything is hidden by `execCoupl`
- In fact, WP obeys the standard rules for the deterministic sequential fragment of `HeapLang`
- We only add new rules for probabilistic constructs
Assume:

\[
\text{specCtxt} \ast \text{spec}(e') \vdash \text{wp } e \{ v. \exists v'. \text{spec}(v') \ast \varphi(v, v') \}
\]
Assume:

\[ \text{specCtx} \ast \text{spec}(e') \vdash \text{wp} \ e \ \{ \forall v. \exists v'. \ \text{spec}(v') \ast \varphi(v, v') \} \]

- ReLoC: If \( e \) terminates, so does \( e' \), and the result values are related by \( \varphi \)
Assume:

\[
\text{specCtx} \times \text{spec}(e') \vdash \text{wp } e \{ v.\exists v'. \text{spec}(v') \star \varphi(v, v') \}
\]

- **ReLoC**: If \( e \) terminates, so does \( e' \), and the result values are related by \( \varphi \).
- **Clutch**: \( e \) terminates with lower or equal probability than \( e' \) and the result distributions are coupled by \( \varphi \).
Reasoning about contextual refinement
Contextual refinement

Two programs are contextually equivalent if they have the same observable behavior under any context. We define contextual refinement through the termination probability:

$$\Theta \mid \Gamma \vdash e_1 \preceq_{\text{ctx}} e_2 : \tau \triangleq \forall \tau', (\mathcal{C} : (\Theta \mid \Gamma \vdash \tau) \Rightarrow (\emptyset \mid \emptyset \vdash \tau')), \sigma.$$ 

$$\text{Pterm}(\mathcal{C}[e_1], \sigma) \leq \text{Pterm}(\mathcal{C}[e_2], \sigma)$$
Logical refinement

We define a ReLoC-style logic refinement and prove it sound wrt. contextual refinement:

$$\Delta \models e_1 \preceq e_2 : \tau \triangleq \forall K. \text{specCtx} \rightarrow G(K[e_2]) \rightarrow \text{naTok}(\mathcal{E}) \rightarrow \wp e_1 \{v_1.\exists v_2. G(K[v_2]) \ast \text{naTok}(\top) \ast [\tau]_{\Delta}(v_1, v_2)\}$$

The value interpretation $[\tau]_{\Delta}(v_1, v_2)$ is essentially the same as in ReLoC.
We recover the standard ReLoC rules:

\[
\begin{align*}
& e_1 \xrightarrow{\text{pure}} e'_1 \quad \triangleright (\Delta \models e \ K[e'_1] \preceq e_2 : \tau) \quad e_2 \xrightarrow{\text{pure}} e'_2 \quad \Delta \models e_1 \preceq K[e'_2] : \tau \\
& \forall \ell. \ell \leftrightarrow v \longrightarrow \Delta \models e \ K[\ell] \preceq e_2 : \tau \quad \ell \leftrightarrow v \quad \ell \leftrightarrow w \longrightarrow \Delta \models e \ K[] \preceq e_2 : \tau \\
& \Delta \models e \ K[\text{ref}(v)] \preceq e_2 : \tau \quad \Delta \models e \ K[\ell \leftarrow w] \preceq e_2 : \tau
\end{align*}
\]
A coupling rule

To reason relationally about probabilistic choices, the judgment satisfies

\[
f : \mathbb{B} \rightarrow \mathbb{B} \text{ bijection} \quad \forall b. \Delta \vdash_{\varepsilon} K[b] \preceq K'[f(b)] : \tau
\]

\[
\Delta \vdash_{\varepsilon} K[\text{flip}] \preceq K'[\text{flip}] : \tau
\]

This rule builds a coupling for flip and sequences it through the contexts
<table>
<thead>
<tr>
<th>Theorem (Soundness)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical refinement implies contextual refinement</td>
</tr>
</tbody>
</table>
Soundness

Theorem (Soundness)

Logical refinement implies contextual refinement

In particular

\[ \vdash e_1 \preceq e_2 : \text{bool} \]

implies

\[ \text{exec}^{\downarrow}(e_1, \sigma)(\text{true}) \leq \text{exec}^{\downarrow}(e_2, \sigma)(\text{true}) \]
\[ \text{exec}^{\downarrow}(e_1, \sigma)(\text{false}) \leq \text{exec}^{\downarrow}(e_2, \sigma)(\text{false}) \]
The approach fundamentally relies on being able to “synchronize” the probabilistic samplings.

This is not always possible.
Eager vs Lazy sampling

eager \triangleq \text{let } b = \text{flip in } \lambda_.\ b

lazy \triangleq \text{let } r = \text{ref}(\text{None}) \text{ in }
\lambda_.\ \text{match } !r \text{ with }
\quad \text{Some}(b) \Rightarrow b
\quad | \text{None} \Rightarrow \text{let } b = \text{flip in }\ r \leftarrow \text{Some}(b); 
b\end

How can we show `$eager$' vs Lazy?
Eager vs Lazy sampling

\[ \text{eager} \triangleq \text{let } b = \text{flip} \text{ in } \lambda_. \ b \]

\[ \text{lazy} \triangleq \text{let } r = \text{ref}(\text{None}) \text{ in } \]

\[ \lambda_. \text{ match } !r \text{ with } \]

\[ \text{Some}(b) \Rightarrow b \]

\[ | \text{None} \Rightarrow \text{let } b = \text{flip} \text{ in } r \leftarrow \text{Some}(b); \]

\[ b \]

end

How can we show \( \vdash \text{eager} \sim_{\text{ctx}} \text{lazy} : \text{unit} \rightarrow \text{nat} \) ?
Asynchronous probabilistic couplings
Asynchronous couplings

We extend the operational semantics with presampling tapes and labelled flips

\[
\text{tape}, \sigma \to^1 \nu, \sigma[\nu \mapsto \epsilon] \quad \text{if } \nu = \text{fresh}(\sigma)
\]

\[
\text{flip}(\nu), \sigma \to^{1/2} \eta, \sigma \quad \text{if } \sigma(\nu) = \epsilon
\]

\[
\text{flip}(\nu), \sigma \to^1 b, \sigma[\nu \mapsto \bar{b}] \quad \text{if } \sigma(\nu) = b \cdot \bar{b}
\]

Note: The tapes can only be populated at the logical level, no language operation writes to them.
This is modelled by a “points-to-like” connective:

\[ \nu \leftrightarrow \vec{b} \]

The asynchronous coupling rule looks like:

\[
\begin{align*}
& f \text{ bijection} \quad e \notin \text{Val} \\
& \nu \leftrightarrow \vec{b} \quad \forall b. \nu \leftrightarrow \vec{b} \cdot b \quad \Rightarrow \quad \Delta \models e \preceq K'[f(b)] : \tau \\
& \Delta \models e \preceq K'[\text{flip}] : \tau
\end{align*}
\]

Soundness relies on the fact that presampling does not change the result distribution.
Future work

- Reasoning about approximate contextual equivalence.
- Supporting more general notions of probabilistic refinement, e.g. Markov Decision Processes.
- Supporting quantitative reasoning about expected costs, runtime, etc.
- Constructing couplings across recursive calls.
Clutch in a nutshell

- A **probabilistic operational semantics** for sequential probabilistic languages
- A **unary coupling-based WP** to prove relations between probabilistic programs
- A **ReLoC-style logical relation** to prove contextual refinement of probabilistic programs
- A **ghost resource** to reason about samples that happen in the future

Try Clutch!  [https://github.com/logsem/clutch](https://github.com/logsem/clutch)