ASYNCHRONOUS PROBABILISTIC COUPLINGS

in Higher-Order Separation Logic

Simon Gregersen Alejandro Aguirre Philipp Haselwarter
Joseph Tassarotti Lars Birkedal

alejandro@cs.au.dk

AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

May 23, 2023

SOLIDUM PETIT IN PROFUNDIS · UNIVERSITAS ARHUSIENSIS

One-time pad

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

$$\mathsf{real} \triangleq \lambda(m \colon \mathsf{bool}).\,\mathsf{let}\; k = \mathsf{flip}\;\; \mathsf{in}\; (k \;\mathsf{xor}\; m)$$

$$\mathsf{ideal} \triangleq \lambda(m \colon \mathsf{bool}).\; \mathsf{flip}$$

One-time pad

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

$$\text{real} \triangleq \lambda(m\colon \text{bool}).\,\text{let}\; k = \text{flip}\; \text{in}\; (k\; \text{xor}\; m)$$
$$\text{ideal} \triangleq \lambda(m\colon \text{bool}).\,\text{flip}$$

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

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 $\mathcal{A}$ 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.

$$\vDash e \precsim e' : \tau$$

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.

$$\vDash e \precsim e' : \tau$$

Multiple examples of this in Iris, e.g. ReLoC

Clutch in a nutshell

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
ReLoC logical refinement
ReLoC type interpretation
HeapLang WP rules
HeapLang
Iris WP
Iris base language
Iris base logic

Clutch
ReLoC + Clutch logical refinement
ReLoC + Clutch type interpretation
Iris + Clutch WP rules
$\mathbf{F}^{\text{rand}}_{\mu,\text{ref}}$
Clutch WP
Clutch base language
Iris base logic

A proof in Clutch

$$\dfrac{}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\ \mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}$$

A proof in Clutch

$$\frac{m : \mathsf{bool} \vDash \mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \mathsf{flip} : \mathsf{bool}}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\,\mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}$$

A proof in Clutch

$$\frac{f : \mathbb{B} \to \mathbb{B} \text{ bijection} \qquad \forall b\colon \mathbb{B}.\ \Delta \vDash_{\mathcal{E}} K[b] \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} K[\,\mathsf{flip}\,] \precsim K'[\,\mathsf{flip}\,] : \tau}\ \text{COUPL}$$

$$\frac{\overline{m\colon \mathsf{bool} \vDash \mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \mathsf{flip} : \mathsf{bool}}}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\,\mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}$$

A proof in Clutch

$$\frac{f : \mathbb{B} \to \mathbb{B} \text{ bijection} \qquad \forall b : \mathbb{B}.\ \Delta \vDash_{\mathcal{E}} K[b] \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} K[\,\mathsf{flip}\,] \precsim K'[\,\mathsf{flip}\,] : \tau} \text{ COUPL}$$

$$\frac{\dfrac{(\cdot \ \mathsf{xor}\ m) \text{ bijection}}{m : \mathsf{bool} \vDash \mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \mathsf{flip} : \mathsf{bool}}}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\,\mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}$$

A proof in Clutch

$$\frac{f : \mathbb{B} \to \mathbb{B} \text{ bijection} \qquad \forall b \colon \mathbb{B}.\ \Delta \vDash_{\mathcal{E}} K[b] \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} K[\,\mathsf{flip}\,] \precsim K'[\,\mathsf{flip}\,] : \tau} \text{ COUPL}$$

$$\cfrac{(\cdot\ \mathsf{xor}\ m) \text{ bijection} \qquad \cfrac{}{m, b \colon \mathsf{bool} \vDash \mathsf{let}\ k = b\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim b\ \mathsf{xor}\ m : \mathsf{bool}}}{\cfrac{m \colon \mathsf{bool} \vDash \mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \mathsf{flip} : \mathsf{bool}}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\,\mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}}$$

A proof in Clutch

$$\dfrac{(\cdot\ \mathsf{xor}\ m)\ \text{bijection} \qquad \dfrac{m, b\colon \mathsf{bool} \vDash b\ \mathsf{xor}\ m \precsim b\ \mathsf{xor}\ m : \mathsf{bool}}{m, b\colon \mathsf{bool} \vDash \mathsf{let}\ k = b\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim b\ \mathsf{xor}\ m : \mathsf{bool}}}{\dfrac{m\colon \mathsf{bool} \vDash \mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \mathsf{flip} : \mathsf{bool}}{\vDash \lambda m.\mathsf{let}\ k = \mathsf{flip}\ \mathsf{in}\ (k\ \mathsf{xor}\ m) \precsim \lambda m.\,\mathsf{flip} : \mathsf{bool} \to \mathsf{bool}}}$$

```
Lemma real_ideal_rel :
  ⊢ REL real << ideal : lrel_bool → 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 $\mathbf{F}^{\text{rand}}_{\mu,\text{ref}}$: sequential fragment of HeapLang plus sampling

$$\begin{aligned}\tau \in \text{Type} ::= {}& \alpha \mid \text{unit} \mid \text{bool} \mid \text{nat} \mid \text{int} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \\ & \forall \alpha.\, \tau \mid \exists \alpha.\, \tau \mid \mu\, \alpha.\, \tau \mid \text{ref}\, \tau\end{aligned}$$

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

flip chooses uniformly between true and false.

Our implementation supports discrete uniform sampling as well

Probability distributions

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:

$$\mathrm{ret} : A \to \mathcal{D}(A)$$
$$\mathrm{ret}(a) \triangleq \lambda a'.\text{if } (a = a') \text{ then } 1 \text{ else } 0$$

$$\gg\!= : \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B)$$
$$(\mu \gg\!= f)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$$

Operational semantics

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

$$(\lambda x.e)\ v, \sigma \to_{\mathsf{h}}^{1} e[v/x], \sigma$$

$$\dots$$

$$\mathsf{flip}, \sigma \to_{\mathsf{h}}^{1/2} b, \sigma \qquad b \in \{\mathsf{true}, \mathsf{false}\}$$

Operational semantics

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

$$(\lambda x.e)\ v, \sigma \to_{\mathsf{h}}^{1} e[v/x], \sigma$$

$$\ldots$$

$$\mathsf{flip}, \sigma \to_{\mathsf{h}}^{1/2} b, \sigma \qquad b \in \{\mathsf{true}, \mathsf{false}\}$$

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

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

Probabilistic evaluation

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

$$\mathrm{exec}_n^{\Downarrow}(e,\sigma) \triangleq \begin{cases} \mathrm{ret}\, e & \text{if } e \in \mathsf{Val} \\ \mathbf{0} & \text{if } e \in \mathsf{Val} \wedge n = 0 \\ \mathrm{step}(e,\sigma) \gg\!= \mathrm{exec}_m^{\Downarrow} & \text{if } e \in \mathsf{Val} \wedge n = m+1 \end{cases}$$

$$\mathrm{exec}^{\Downarrow}(e,\sigma) \triangleq \mathsf{lim}_{n\to\infty}\, \mathrm{exec}_n^{\Downarrow}$$

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

$$\mathsf{Pterm}(e,\sigma) \triangleq \sum_{v \in \mathsf{Val}} \mathrm{exec}^{\Downarrow}(e,\sigma)(v)$$

Probabilistic languages in Iris

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

```
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 $R : A \to B \to \text{Prop}$

Couplings 101

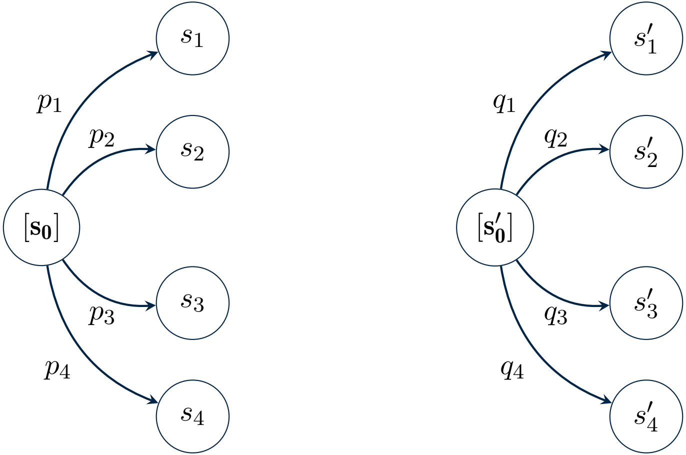
Couplings are a construction that allows us to reason relationally about probabilistic programs.
To construct a coupling between $\mu_1 : \mathcal{D}(A), \mu_2 : \mathcal{D}(B)$:

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

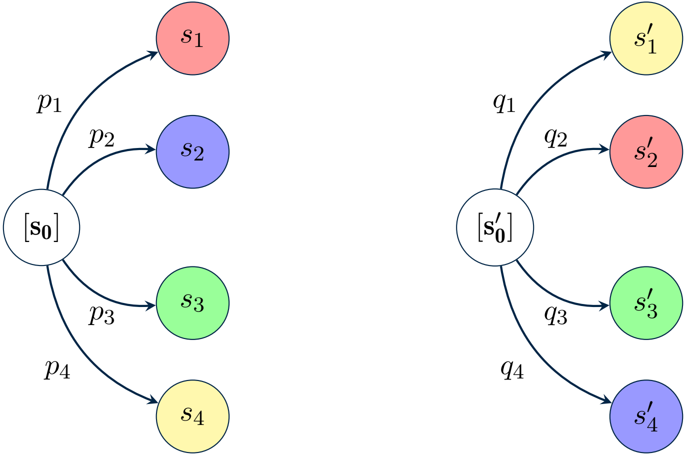
We then say that "μ_1 and μ_2 are coupled by R".

notation: $\mu_1 \sim \mu_2 : R$

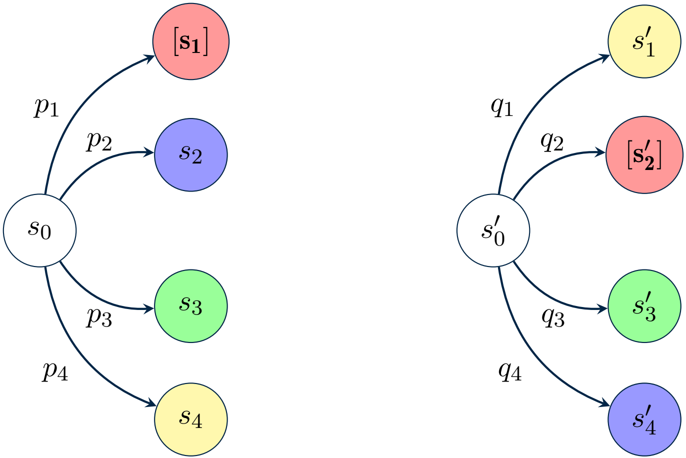
Couplings 101



Probabilistic Couplings



Probabilistic Couplings



Reasoning with couplings

Reasoning with couplings

Introduction: $$\frac{(a,b) \in R}{\mathrm{ret}(a) \sim \mathrm{ret}(b) : R}$$

Reasoning with couplings

Introduction:

$$\frac{(a, b) \in R}{\mathrm{ret}(a) \sim \mathrm{ret}(b) : R} \qquad \frac{f : \mathbb{B} \to \mathbb{B} \text{ bij.} \qquad \forall b, (b, f(b)) \in R}{\textsf{flip} \sim \textsf{flip} : R}$$

Reasoning with couplings

Introduction:

$$\frac{(a, b) \in R}{\mathrm{ret}(a) \sim \mathrm{ret}(b) : R} \qquad \frac{f : \mathbb{B} \to \mathbb{B} \text{ bij.} \qquad \forall b, (b, f(b)) \in R}{\mathsf{flip} \sim \mathsf{flip} : R}$$

Sequencing:

$$\frac{\mu_1 \sim \mu_2 : R \qquad \forall (a, b) \in R.\ f_1(a) \sim f_2(b) : S}{\mu_1 \gg\!= f_1 \sim \mu_2 \gg\!= f_2 : S}$$

Reasoning with couplings

Introduction:

$$\frac{(a, b) \in R}{\mathrm{ret}(a) \sim \mathrm{ret}(b) : R} \qquad \frac{f : \mathbb{B} \to \mathbb{B} \text{ bij.} \qquad \forall b, (b, f(b)) \in R}{\text{flip} \sim \text{flip} : R}$$

Sequencing:

$$\frac{\mu_1 \sim \mu_2 : R \qquad \forall (a, b) \in R.\ f_1(a) \sim f_2(b) : S}{\mu_1 \gg\!= f_1 \sim \mu_2 \gg\!= f_2 : S}$$

Elimination:

$$\frac{\mu_1 \sim \mu_2 : (=)}{\forall x. \mu_1(x) = \mu_2(x)}$$

A coupling-based WP

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

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\, e_1\, \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \vee \\ & (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*}\ {}_{\mathcal{E}}\Rrightarrow_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \qquad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright\ {}_{\emptyset}\Rrightarrow_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\, e_1'\, \{\Phi\}))\end{aligned}$$

A coupling-based WP

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

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\; e_1 \;\{\Phi\} \triangleq\; & (e_1 \in \mathsf{Val} \wedge \mathrel{\Rrightarrow}_{\mathcal{E}} \Phi(e_1)) \vee \\ & (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\; S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*} {}_{\mathcal{E}}\!\Rrightarrow_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \qquad (\lambda e_1', \sigma_1', \rho_2'.\; \triangleright\, {}_{\emptyset}\!\Rrightarrow_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\; e_1' \;\{\Phi\}))\end{aligned}$$

A coupling-based WP

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

$$
\begin{aligned}
\mathsf{wp}_{\mathcal{E}}\, e_1\, \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \land \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}} \Phi(e_1)) \lor \\
& (e_1 \notin \mathsf{Val} \land \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*}\ {}_{\mathcal{E}}\!\mathrel{|\!\!\Rrightarrow}_{\emptyset} \\
& \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\
& \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright\, {}_{\emptyset}\!\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\, e_1'\, \{\Phi\}))
\end{aligned}
$$

A coupling-based WP

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

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\, e_1\, \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \vee \\ & (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\, S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*} {}_{\mathcal{E}}\Rrightarrow_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\, \triangleright {}_{\emptyset}\Rrightarrow_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\, e_1'\, \{\Phi\}))\end{aligned}$$

A coupling-based WP

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

$$
\begin{aligned}
\mathsf{wp}_{\mathcal{E}}\ e_1\ \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \vee \\
& (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \twoheadrightarrow {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \\
& \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\
& \qquad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright\, {}_{\emptyset}{\Rrightarrow}_{\mathcal{E}}\, S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\ e_1'\ \{\Phi\}))
\end{aligned}
$$

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 ρ is tracked by a specification predicate $G(\rho)$:

$$
\begin{aligned}
\mathsf{wp}_{\mathcal{E}}\ e_1\ \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \wedge {\Rrightarrow_{\mathcal{E}}}\, \Phi(e_1)) \vee \\
& (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*}\ {}_{\mathcal{E}}{\Rrightarrow_{\emptyset}} \\
& \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\
& \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright\ {}_{\emptyset}{\Rrightarrow_{\mathcal{E}}}\, S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\ e_1'\ \{\Phi\}))
\end{aligned}
$$

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 ρ is tracked by a specification predicate $G(\rho)$:

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\; e_1\; \{\Phi\} \triangleq\; & (e_1 \in \mathsf{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \vee \\ & (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\; S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*}\; {}_{\mathcal{E}}\!\Rrightarrow_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\; \triangleright\, {}_{\emptyset}\!\Rrightarrow_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\; e_1'\; \{\Phi\}))\end{aligned}$$

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 ρ is tracked by a specification predicate $G(\rho)$:

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\ e_1\ \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \land \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \lor \\ & (e_1 \notin \mathsf{Val} \land \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \twoheadrightarrow {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright {}_{\emptyset}{\Rrightarrow}_{\mathcal{E}} S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\ e_1'\ \{\Phi\}))\end{aligned}$$

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 ρ is tracked by a specification predicate $G(\rho)$:

$$\begin{aligned}\mathsf{wp}_{\mathcal{E}}\ e_1\ \{\Phi\} \triangleq\ & (e_1 \in \mathsf{Val} \wedge \Rrightarrow_{\mathcal{E}} \Phi(e_1)) \vee \\ & (e_1 \notin \mathsf{Val} \wedge \forall \sigma_1, \rho_2.\ S(\sigma_1) * G(\rho_2) \mathrel{-\!\!*}\ {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \\ & \quad \mathsf{execCoupl}(e_1, \sigma_1, \rho_2) \\ & \quad\quad (\lambda e_1', \sigma_1', \rho_2'.\ \triangleright\ {}_{\emptyset}{\Rrightarrow}_{\mathcal{E}}\ S(\sigma_1') * G(\rho_1') * \mathsf{wp}_{\mathcal{E}}\ e_1'\ \{\Phi\}))\end{aligned}$$

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 $\Phi : \mathsf{Val} \to \mathsf{iProp}$ (and not $\mathcal{D}(\mathsf{Val}) \to \mathsf{iProp}$)
- 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

Adequacy

Assume:

$$\mathsf{specCtx} * \mathsf{spec}(e') \vdash \mathsf{wp}\ e\ \left\{v. \exists v'.\ \mathsf{spec}(v') * \varphi(v, v')\right\}$$

Adequacy

Assume:

$$\text{specCtx} * \text{spec}(e') \vdash \text{wp } e \left\{v.\exists v'.\, \text{spec}(v') * \varphi(v, v')\right\}$$

- ReLoC: If e terminates, so does e', and the result values are related by φ

Adequacy

Assume:

$$\mathsf{specCtx} * \mathsf{spec}(e') \vdash \mathsf{wp}\ e\ \{v. \exists v'. \mathsf{spec}(v') * \varphi(v, v')\}$$

- ReLoC: If e terminates, so does e', and the result values are related by φ
- Clutch: e terminates with lower or equal probability than e' and the result distributions are coupled by φ

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:

$$\Theta \mid \Gamma \vdash e_1 \precsim_{\mathsf{ctx}} e_2 : \tau \triangleq \forall \tau', (\mathcal{C} : (\Theta \mid \Gamma \vdash \tau) \Rightarrow (\emptyset \mid \emptyset \vdash \tau')), \sigma.$$
$$\mathsf{Pterm}(\mathcal{C}[e_1], \sigma) \leq \mathsf{Pterm}(\mathcal{C}[e_2], \sigma)$$

Logical refinement

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

$$\Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau \triangleq \forall K.\, \mathsf{specCtx} \mathrel{-\!\!*} G(K[e_2]) \mathrel{-\!\!*} \mathsf{naTok}(\mathcal{E}) \mathrel{-\!\!*} \\ \mathsf{wp}\; e_1 \,\{v_1.\exists v_2.\, G(K[v_2]) * \mathsf{naTok}(\top) * [\![\tau]\!]_\Delta(v_1, v_2)\}$$

The value interpretation $[\![\tau]\!]_\Delta(v_1, v_2)$ is essentially the same as in ReLoC.

Some relational rules

We recover the standard ReLoC rules:

$$\frac{e_1 \overset{\text{pure}}{\rightsquigarrow} e_1' \qquad \triangleright(\Delta \vDash_{\mathcal{E}} K[e_1'] \precsim e_2 : \tau)}{\Delta \vDash_{\mathcal{E}} K[e_1] \precsim e_2 : \tau} \qquad \frac{e_2 \overset{\text{pure}}{\rightsquigarrow} e_2' \qquad \Delta \vDash_{\mathcal{E}} e_1 \precsim K[e_2'] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim K[e_2] : \tau}$$

$$\frac{\forall \ell.\, \ell \mapsto v \twoheadrightarrow \Delta \vDash_{\mathcal{E}} K[\ell] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} K[\mathsf{ref}(v)] \precsim e_2 : \tau} \qquad \frac{\ell \mapsto v \qquad \ell \mapsto w \twoheadrightarrow \Delta \vDash_{\mathcal{E}} K[()] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} K[\ell \leftarrow w] \precsim e_2 : \tau}$$

A coupling rule

To reason relationally about probabilistic choices, the judgment satisfies

$$\frac{f : \mathbb{B} \to \mathbb{B} \text{ bijection} \qquad \forall b.\, \Delta \vDash_{\mathcal{E}} K[b] \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} K[\,\mathsf{flip}\,] \precsim K'[\,\mathsf{flip}\,] : \tau}$$

This rule builds a coupling for flip and sequences it through the contexts

Soundness

Theorem (Soundness)

Logical refinement implies contextual refinement

Soundness

Theorem (Soundness)

Logical refinement implies contextual refinement

In particular

$$\vDash e_1 \precsim e_2 : \mathsf{bool}$$

implies

$$\mathrm{exec}^{\Downarrow}(e_1, \sigma)(\mathsf{true}) \leq \mathrm{exec}^{\Downarrow}(e_2, \sigma)(\mathsf{true})$$
$$\mathrm{exec}^{\Downarrow}(e_1, \sigma)(\mathsf{false}) \leq \mathrm{exec}^{\Downarrow}(e_2, \sigma)(\mathsf{false})$$

But...

The approach fundamentally relies on being able to “synchronize” the probabilistic samplings.

$$\frac{f \text{ bijection} \qquad \forall b.\, \Delta \vDash_{\mathcal{E}} K[b] \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} K[\,\mathsf{flip}\,] \precsim K'[\,\mathsf{flip}\,] : \tau}$$

This is not always possible.

Eager vs Lazy sampling

$$\begin{aligned}\textsf{eager} &\triangleq \textsf{let } b = \textsf{flip in}\\ &\quad \lambda\_.\, b\end{aligned}$$

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

Eager vs Lazy sampling

$$\text{eager} \triangleq \text{let } b = \text{flip in} \\ \lambda\_.\, b$$

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

How can we show $\vdash \text{eager} \simeq_{\text{ctx}} \text{lazy} : \text{unit} \to \text{nat}$?

Asynchronous probabilistic couplings

Asynchronous couplings

We extend the operational semantics with presampling tapes and labelled flips

$$
\begin{array}{ll}
\textsf{tape}, \sigma \to^{1} \iota, \sigma[\iota \mapsto \epsilon] & \text{if } \iota = \mathrm{fresh}(\sigma) \\
\textsf{flip}(\iota), \sigma \to^{1/2} n, \sigma & \text{if } \sigma(\iota) = \epsilon \\
\textsf{flip}(\iota), \sigma \to^{1} b, \sigma[\iota \mapsto \vec{b}] & \text{if } \sigma(\iota) = b \cdot \vec{b}
\end{array}
$$

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:

$$\iota \hookrightarrow \vec{b}$$

The asynchronous coupling rule looks like:

$$\frac{f \text{ bijection} \qquad e \notin \mathsf{Val} \qquad \iota \hookrightarrow \vec{b} \qquad \forall b.\, \iota \hookrightarrow \vec{b} \cdot b \;-\!\!*\; \Delta \vDash_{\mathcal{E}} e \precsim K'[f(b)] : \tau}{\Delta \vDash_{\mathcal{E}} e \precsim K'[\,\mathsf{flip}\,] : \tau}$$

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`