Higher-Order Separation Logic for Distributed Systems and Security

Simon Oddershede Gregersen
Programs.
“A computer program is a sequence or set of instructions in a programming language for a computer to execute.”
In a nutshell

This dissertation

... for computer programs, not busses!
This dissertation

Features

‣ Distribution
‣ Information-flow control types
‣ Randomization

Properties

‣ Safety
‣ Simulation
‣ (Liveness)
‣ Noninterference
‣ Contextual equivalence
This dissertation

Aneris: A Mechanized Logic for Modular Reasoning about Distributed Systems
Morten Krogh-Jespersen, Amin Timany, Marit Edna Ohlenbusch, Simon Oddershede Gregersen, Lars Birkedal
@ ESOP ’20

Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic
Léon Gondelman, Simon Oddershede Gregersen, Abel Nieto, Amin Timany, Lars Birkedal
@ POPL ’21

Mechanized Logical Relations for Termination-Insensitive Noninterference
Simon Oddershede Gregersen, Johan Bay, Amin Timany, Lars Birkedal
@ POPL ’21

Trillium: History-Sensitive Refinement in Separation Logic
Amin Timany, Simon Oddershede Gregersen, Léo Stefanesco, Léon Gondelman, Abel Nieto

Asynchronous Probabilistic Couplings in Higher-Order Separation Logic
Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, Lars Birkedal
[Manuscript]
Thesis statement:

Higher-order separation logic is all you need!
This dissertation

Aneris: A Mechanized Logic for Modular Reasoning about Distributed Systems  
Morten Krogh-Jespersen, Amin Timany, Marit Edna Ohlenbusch, Simon Oddershede Gregersen, Lars Birkedal  
@ ESOP '20

Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic  
Léon Gondelman, Simon Oddershede Gregersen, Abel Nieto, Amin Timany, Lars Birkedal  
@ POPL '21

Mechanized Logical Relations for Termination-Insensitive Noninterference  
Simon Oddershede Gregersen, Johan Bay, Amin Timany, Lars Birkedal  
@ POPL '21

Trillium: History-Sensitive Refinement in Separation Logic  
Amin Timany, Simon Oddershede Gregersen, Léo Stefanesco, Léon Gondelman, Abel Nieto, Lars Birkedal  
[Manuscript]

Asynchronous Probabilistic Couplings in Higher-Order Separation Logic  
Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, Lars Birkedal  
[Manuscript]
Mechanized Logical Relations for Termination-Insensitive Noninterference

joint work with Johan Bay, Amin Timany, and Lars Birkedal
The prevailing basic semantic notion of secure information flow is noninterference.
Program $e$ satisfies **termination-insensitive noninterference**, abbrv. $\text{TINI}(e)$, when

$$e[v_1/x] \downarrow o_1 \quad \text{and} \quad e[v_2/x] \downarrow o_2 \quad \text{implies} \quad o_1 \simeq o_2$$

for all secrets $v_1$ and $v_2$.  

The problem

Information-flow control enforcement often comes as a static type system:

\[ \Gamma \vdash e : t^\ell \quad \text{implies} \quad \text{TINI}(e) \]

To really be useful, it must support the same features as modern languages:

- higher types
- reference types
- higher-order state
- ...

The difficulty of proving the system sound increases, however.
This work

- shows that such a rich type system satisfies TINI
- with full mechanization of all results in Coq
- using a semantic model

Compositional integration of syntactically well-typed and ill-typed components:

$$\Gamma, x : \tau_2 \vdash e_1 : \tau_1 \quad \text{and} \quad e_2 \in \llbracket \tau_2 \rrbracket \quad \text{implies} \quad \text{TINI}(e_1[e_2/x])$$
Types

\[ \tau ::= t^\ell \]

\[ t ::= \mathbb{B} \mid \mathbb{N} \mid \tau \times \tau \mid \tau + \tau \]

\[ \tau \rightarrow^\ell \tau \mid \text{ref}(\tau) \mid \alpha \mid \forall_\ell \alpha. \tau \mid \forall_\ell \kappa. \tau \mid \exists_\alpha. \tau \mid \mu_\alpha. \tau \]

\[ \ell ::= \kappa \mid l \in \mathcal{L} \mid \ell \sqcup \ell \]

For this presentation we consider \( \mathcal{L} = \{ \bot, \top \} \) where \( \bot \subseteq \top, \top \nsubseteq \bot \)

Consider \textbf{if secret then } f \textbf{ ()} --- if \( f \) has public side effects, \textit{secret} is leaked
Typing judgment

\[ \Xi | \Psi | \Gamma \vdash_{pc} e : \tau \]

Term-level context

Type-level context

Label context

“Program counter” label
Type system

\[
\text{T-IF} \quad \Xi \mid \Psi \mid \Gamma \vdash_{pc} e : \mathbb{B}^l \quad \forall i \in \{1, 2\}. \Xi \mid \Psi \mid \Gamma \vdash_{pc \cup \ell} e_i : \tau \quad \Psi \vdash \tau \downarrow \ell \\
\Xi \mid \Psi \mid \Gamma \vdash_{pc} \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau
\]

\[
\text{T-STORE} \quad \Xi \mid \Psi \mid \Gamma \vdash_{pc} e_1 : \text{ref}(\tau)^l \quad \Xi \mid \Psi \mid \Gamma \vdash_{pc} e_2 : \tau \quad \Psi \vdash \tau \downarrow pc \cup \ell \\
\Xi \mid \Psi \mid \Gamma \vdash_{pc} e_1 := e_2 : \perp^l
\]
\textbf{Theorem} (Termination-Insensitive Noninterference)

If
\[
x : \mathbb{B}^\top \vdash \bot \quad e : \mathbb{B}^\bot, \quad \vdash \bot \quad v_1 : \mathbb{B}^\top, \quad \text{and} \quad \vdash \bot \quad v_2 : \mathbb{B}^\top
\]

then
\[
(\emptyset, e[v_1/x]) \rightarrow^* (h_1, v_1') \quad \text{and} \quad (\emptyset, e[v_2/x]) \rightarrow^* (h_2, v_2') \quad \text{implies} \quad v_1' = v_2'.
\]
Our approach

We set up a binary logical relation

\[ \Xi \mid \Psi \mid \Gamma \vdash e_1 \approx e_2 : \tau \]

such that

\[ \Xi \mid \Psi \mid \Gamma \vdash_{pc} e : \tau \quad \Rightarrow \quad \Xi \mid \Psi \mid \Gamma \vdash e \approx e : \tau \]

\[ \Xi \mid \Psi \mid \Gamma \vdash e \approx e : \tau \quad \Rightarrow \quad \text{TINI}(e) \]

However, this requires manipulating and defining a complex semantic model.
Our approach cont’d

We combat the complexity by defining the relation in Iris:

- Convenient logical connectives for expressing the relation,
- High-level logic to reason within, and
- Coq formalization and the Iris Proof Mode to mechanize our proofs

Not a novel approach, but some novel challenges!
Value relation

\[[a]_\theta^e \triangleq \tau_\theta(\theta(a))\]
\[[\mathbb{B}]_\theta^e (v, v') \triangleq v = v' (\mathbf{1})\]
\[[\mathbb{B}]_\theta^e (v, v') \triangleq v = v' \in \{\mathbf{true}, \mathbf{false}\}\]

\[[[r_1 \times r_2]_\theta^e (v, v') \triangleq \exists v_1, v_2, v'_1, v'_2. v = (v_1, v_2) \times v' = (v'_1, v'_2) * [r_1]_\theta^e (v_1, v'_1) * [r_2]_\theta^e (v_2, v'_2)\]

\[[[r_1 + r_2]_\theta^e (v, v') \triangleq \bigvee_{v' \in \{v, v'\}} \exists w, w'. v = 1w1, w * v' = 1w1, w' * [r_1 + r_2]_\theta^e (v, v')\]

\[[\sqrt{r_1} \supseteq r_2]_\theta^e (v, v') \triangleq \square (\exists w, w'. [\sqrt{r_1}]_\theta^e (v, w) \Leftrightarrow [r_2]_\theta^e (w, v')) * [\sqrt{r_1}]_\theta^e (v, w) * [r_1]_\theta^e (v, v')\]

Expressions relation

\[E[r]_\theta^e (v, v') \triangleq \text{mwp} \ v \sim v' \{[r]_\theta^e\}\]

Environment relation

\[G[x : \tau]_\theta^e (v, v') \triangleq \text{G}[G[x : \tau]_\theta^e (v, v') * [r]_\theta^e (w, w')\]

Semantic typing relation

\[\text{Coh}(\theta) \triangleq \bigwedge_{(\mathbf{1}, \Phi, \theta_1, \theta_2) \in \mathbb{C}} \square (\forall v, v', \Phi(v, v') \rightarrow \Phi_1(v) \Leftrightarrow \Phi_2(v'))\]

\[\mathbb{E}[\Phi] \vdash \psi \equiv \psi' : \tau \triangleq \bigwedge_{\forall \theta, \rho : \mathbb{E}[\Phi] \subseteq \mathbb{E}[\psi] \subseteq \mathbb{E}[\psi']} \text{Coh}(\theta) \triangleq \text{G}[G[x : \tau]_\theta^e (v, v') \rightarrow [r]_\theta^e (e[v/\theta], e[v'/\theta])\]
Challenge #1

Existing encodings of “logical” logical relations are termination sensitive:

\[ e_1 \rightarrow^* v_1 \Rightarrow e_2 \rightarrow^* v_2 \quad \land \quad v_1 \approx v_2. \]

However, we need a termination insensitive notion:

\[ e_1 \rightarrow^* v_1 \quad \land \quad e_2 \rightarrow^* v_2 \quad \Rightarrow \quad v_1 \approx v_2. \]

Solution: a new modal weakest precondition theory \( \text{mwp} \ e \sim e' \{Q\} \)
Challenge #2

As part of our proofs, we have to show, e.g.,

\[ \vdash \text{if } v \text{ then } e_1 \text{ else } e_2 \quad \approx \quad \text{if } v' \text{ then } e_1 \text{ else } e_2 \quad : \quad t^\top \]

where \( \vdash v \approx v' : \mathbb{B}^\top \), meaning \( v, v' \in \{ \text{true, false} \} \). This means we have to prove, e.g.,

\[ \vdash e_1 \quad \approx \quad e_2 \quad : \quad t^\top \]

Luckily, we don’t really need to care about return values, only side-effects!
Challenge #2

Solution:

- A **binary relation** for relating terms that are “publicly equivalent”
- A **unary relation** for characterising terms that do not have public side-effects

\[
[t^e]_\Theta^\rho(v, v') \triangleq \begin{cases} 
[t]_\Theta^\rho(v, v') & \text{if } \llbracket \ell \rrbracket_\rho = \bot \\
[t]_\Theta^\rho_L(v) \ast [t]_\Theta^\rho_R(v') & \text{othw.}
\end{cases}
\]

Needs two instantiation of the MWP theory: \( \text{mwp } e \{ Q \} \) and \( \text{mwp } e \sim e' \{ Q \} \) and a logical way of encoding a “subsumption” property.
Asynchronous Probabilistic Couplings in Higher-Order Separation Logic

joint work with Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal
Setting the stage

- Distributed applications often communicate over an untrusted network.
- Randomization is a crucial ingredient in cryptographic protocols.
- Security is often phrased as an indistinguishability of two probabilistic programs.

**Goal:** a relational program logic for an expressive language with coin flips for proving contextual equivalences.
higher-order state

impredicative polymorphism

existential types

CTX EQV.

recursive types

probabilistic choice
Complications

Programs evaluate to distributions over values, not just values.

What do we do about those?

Many probabilistic relational Hoare logics (pRHLs) make use of probabilistic couplings:

\[ \mu_1 \sim \mu_2 : R \]

If \( R \triangleq (\cong) \) then \( \mu_1 = \mu_2 \).
Couplings in pRHLs

In pRHLs, couplings manifest as coupling rules:

\[
\begin{align*}
\text{PRHL-COUPLE} & \quad f \text{ bijection} \\
\{\text{True}\} \quad \text{flip} & \sim \text{flip} \quad \{v_1, v_2. \exists b : \mathbb{B}. v_1 = b \land v_2 = f(b)\}
\end{align*}
\]

E.g., for One-Time Pad:

\[
\begin{align*}
\text{let } k = \text{flip in} & \quad k \otimes m \\
& \sim \quad \text{flip}
\end{align*}
\]

Pick \( f(b) = \text{if } m \text{ then } \neg b \text{ else } b \)
However, the approach requires you to **synchronize** the probabilistic choices.

This is not always possible.

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

\[
\text{let } r = \text{ref}(\text{None}) \text{ in }
\lambda_. \text{match } !r \text{ with }
\quad \text{Some } (b) \Rightarrow b
\quad \text{| None} \Rightarrow \text{let } b = \text{flip in } r := \text{Some } (b); b
\text{end}
\]
This work

- A higher-order probabilistic relational separation logic, “Clutch”, for proving contextual equivalence of probabilistic programs with higher-order references, impredicative polymorphism, and recursive types.

- A proof method for asynchronous couplings that allows us to reason about sampling as if it was state.

- Full mechanization of all results in Coq.
**Key ideas of Clutch**

A (separation logic) refinement judgment

\[ \Delta \models e_1 \preceq e_2 : \tau \]

"\( e_1 \) refines \( e_2 \) at type \( \tau \)"

---

**REL-PURE-L**

\( e_1 \pure \Rightarrow e'_1 \quad \Delta \models K[e'_1] \preceq e_2 : \tau \)

\[ \Delta \models K[e_1] \preceq e_2 : \tau \]

---

**REL-LOAD-L**

\( \ell \mapsto v \quad \Delta \models K[v] \preceq e_2 : \tau \)

\[ \Delta \models K[!\ell] \preceq e_2 : \tau \]

---

**REL-STORE-R**

\( \ell \mapsto_s v \quad \ell \mapsto_s w \quad \Delta \models e_1 \preceq K[()] : \tau \)

\[ \Delta \models e_1 \preceq K[\ell := w] : \tau \]

---

**REL-COUPLE-FLIPS**

\[ f \text{ bijection} \quad \forall b. \Delta \models K[b] \preceq K'[f(b)] : \tau \]

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

---

**REL-STORE**

\[ \ell \mapsto_s v \quad \ell \mapsto_s w \quad \Delta \models e_1 \preceq K[()] : \tau \]

\[ \Delta \models e_1 \preceq K[\ell := w] : \tau \]
Asynchronous couplings

To support asynchronous couplings, we introduce presampling tapes.
Asynchronous couplings

To support asynchronous couplings, we introduce \textit{presampling tapes}.

Operationally, we extend the state of program execution with a "heap of tapes" onto which we can presample bits.

\begin{center}
\begin{tabular}{c|c|c}
\(t_0\) & \(t_1\) & \(t_2\) \\
\hline
0 & 0 & 1 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
1 & 1 & 0
\end{tabular}
\end{center}
Asynchronous couplings

To support asynchronous couplings, we introduce presampling tapes.

Operationally, we extend the state of program execution with a "heap of tapes" onto which we can presample bits.

\[
\begin{array}{c|c|c|c}
  \tau_0 & \tau_1 & \tau_2 & \tau_3 \\
  \hline
  0 & 0 & 1 & \\
  1 & 1 & 0 & \\
  0 & & 0 & \\
  1 & & & \\
\end{array}
\]
Asynchronous couplings

To support asynchronous couplings, we introduce presampling tapes.

Operationally, we extend the state of program execution with a "heap of tapes" onto which we can presample bits.

\[ \text{flip}(\nu_3) \rightarrow \frac{1}{2} \quad b \]

<table>
<thead>
<tr>
<th>( \nu_0 )</th>
<th>( \nu_1 )</th>
<th>( \nu_2 )</th>
<th>( \nu_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Asynchronous couplings

To support asynchronous couplings, we introduce presampling tapes.

Operationally, we extend the state of program execution with a "heap of tapes" onto which we can presample bits.

flip(ν₂)
Asynchronous couplings

To support asynchronous couplings, we introduce presampling tapes.

Operationally, we extend the state of program execution with a “heap of tapes” onto which we can presample bits.

\[
\text{flip}(\nu_2) \rightarrow^1 1
\]
Asynchronous couplings

But no language primitives add values to the tapes!

Instead, presampling steps will be ghost operations purely used in the logic.

— in fact, they can be entirely erased!
Asynchronous couplings cont’d

Tapes are "just" state so we introduce a separation logic connective

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

that denotes ownership of a tape and its contents.

\[
\begin{align*}
\text{REL-ALLOC-TAPE-L} & \quad \forall \nu. \nu \leftrightarrow \epsilon \rightarrow \Delta \vdash K[\nu] \preceq e : \tau \\
\Delta \vdash K[\text{tape}] & \preceq e : \tau
\end{align*}
\]

\[
\begin{align*}
\text{REL-FLIP-TAPE-L} & \quad \nu \leftrightarrow b \cdot \bar{b} \quad \nu \leftrightarrow \bar{b} \rightarrow \Delta \vdash K[b] \preceq e_2 : \tau \\
\Delta & \vdash K[\text{flip}(\nu)] \preceq e_2 : \tau
\end{align*}
\]
Asynchronous couplings cont’d

\[
\begin{align*}
\text{rel-couple-tape-l} & : \quad f \text{ bijection} \quad \nu \leftrightarrow \bar{b} \quad \forall b. \nu \leftrightarrow \bar{b} \cdot b \quad \Delta \vdash e \preceq K'[f(b)] : \tau \\
\Delta \vdash e \preceq K'[\text{flip()}] : \tau
\end{align*}
\]

\[
\begin{array}{ccc}
\vdash & \vdash \\
e, \sigma & K[\text{flip()}], \sigma' \\
\downarrow & \downarrow \\
e, \sigma[\nu \mapsto b] & K[f(b)], \sigma' \\
\vdots & \vdots
\end{array}
\]
Motivating example

let $r = \text{ref}(\text{None})$ in
\[
\lambda_. \ \text{match} \ !r \ \text{with} \\
\text{Some} \ (b) \Rightarrow b \\
| \text{None} \Rightarrow \text{let} \ b = \text{flip} \ \text{in} \\
\text{r} := \text{Some} \ (b) ; \\
b
\text{end}
\]

\[\leadsto_{\text{ctx}}\]

let $b = \text{flip}$ in
\[
\lambda_. \ b
\]
Motivating example

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

\[
\begin{align*}
\lambda_. \text{ match } !r \text{ with } \\
\phantom{\lambda_.} &\text{let } b = \text{flip in } \\
\phantom{\lambda_.} &\text{let } r = \text{Some } (b) \text{; } \\
\phantom{\lambda_.} &\text{let } b = \text{flip in } \\
\phantom{\lambda_.} &\text{let } r = \text{Some } (b) \text{; } \\
\end{align*}
\]

\[
\begin{align*}
\lambda_. \text{ match } !r \text{ with } \\
\phantom{\lambda_.} &\text{let } b = \text{flip in } \\
\phantom{\lambda_.} &\text{let } r = \text{Some } (b) \text{; } \\
\phantom{\lambda_.} &\text{let } b = \text{flip in } \\
\phantom{\lambda_.} &\text{let } r = \text{Some } (b) \text{; } \\
\phantom{\lambda_.} &\text{end}
\end{align*}
\]
Motivating example

\[
\begin{align*}
\text{let } r & = \text{ref(\text{None}) in} \\
\lambda_\cdot \text{match } \! r \text{ with} \\
& \quad \text{Some } (b) \Rightarrow b \\
& \quad | \text{None} \Rightarrow \text{let } b = \text{flip in} \\
& \quad \quad r := \text{Some } (b); \\
& \quad b \\
\end{align*}
\]

\[
\begin{align*}
\text{let } \iota & = \text{tape in} \\
\text{let } r & = \text{ref(\text{None}) in} \\
\lambda_\cdot \text{match } \! r \text{ with} \\
& \quad \text{Some } (b) \Rightarrow b \\
& \quad | \text{None} \Rightarrow \text{let } b = \text{flip(\iota) in} \\
& \quad \quad r := \text{Some } (b); \\
& \quad b \\
\end{align*}
\]

\[
\begin{align*}
\text{let } \iota & = \text{tape in} \\
\text{let } r & = \text{ref(\text{None}) in} \\
\lambda_\cdot \text{match } \! r \text{ with} \\
& \quad \text{Some } (b) \Rightarrow b \\
& \quad | \text{None} \Rightarrow \text{let } b = \text{flip(\iota) in} \\
& \quad \quad r := \text{Some } (b); \\
& \quad b \\
\end{align*}
\]

\[
\begin{align*}
\text{let } r & = \text{ref(\text{None}) in} \\
\lambda_\cdot \text{match } \! r \text{ with} \\
& \quad \text{Some } (b) \Rightarrow b \\
& \quad | \text{None} \Rightarrow \text{let } b = \text{flip(\iota) in} \\
& \quad \quad r := \text{Some } (b); \\
& \quad b \\
\end{align*}
\]
Conclusion
higher-order
impredicative polymorphism
label polymorphism
higher-order
recursive types
IFC types
existential types
probabilistic choice
higher-order state
impreclicative polymorphism
existential types
recursive types
CTX EQV.
Higher-order separation logic is all you need!
Thank you!

Simon Oddershede Gregersen
PhD dissertation defence
17 March, 2023
(Very) simple examples

- **Multiplying by zero**

  \[ \lambda v. v \times 0 \]
  
  cannot be syntactically typed at \( \mathbb{N}^T \rightarrow \mathbb{N}^\perp \).

- **Temporary explicit “leaks”**

  ```
  let x = !l in l := !h; ...; l := x
  ```

  is not syntactically well-typed if \( h \) contains sensitive information.
\[
\begin{align*}
\tau &= \text{fresh}(\sigma) \\
\text{tape, } \sigma \rightarrow^1 \tau, \sigma[\tau \mapsto \epsilon] \\
\sigma(\tau) &= \epsilon \\
&\quad b \in \{\text{true, false}\} \\
\text{flip}(\tau), \sigma \rightarrow^{1/2} b, \sigma \\
\text{flip(} \tau, \sigma[\tau \mapsto b \cdot \bar{b}] \rightarrow^1 b, \sigma[\tau \mapsto \bar{b}] 
\end{align*}
\]
**Definition (Coupling).** Let $\mu_1 \in \mathcal{D}(A), \mu_2 \in \mathcal{D}(B)$. A sub-distribution $\mu \in \mathcal{D}(A \times B)$ is a coupling of $\mu_1$ and $\mu_2$ if

1. $\forall a. \sum_{b \in B} \mu(a, b) = \mu_1(a)$
2. $\forall b. \sum_{a \in A} \mu(a, b) = \mu_2(b)$

Given relation $R : A \times B$ we say $\mu$ is an $R$-coupling if furthermore $\text{supp}(\mu) \subseteq R$. We write $\mu_1 \sim \mu_2 : R$ if there exists an $R$-coupling of $\mu_1$ and $\mu_2$.

**Lemma (Composition of couplings).** Let $R : A \times B$, $S : A' \times B'$, $\mu_1 \in \mathcal{D}(A)$, $\mu_2 \in \mathcal{D}(B)$, $f_1 : A \to \mathcal{D}(A')$, and $f_2 : B \to \mathcal{D}(B')$.

1. If $(a, b) \in R$ then $\text{ret}(a) \sim \text{ret}(b) : R$.
2. If $\forall (a, b) \in R. f_1(a) \sim f_2(b)$ and $\mu_1 \sim \mu_2 : R$ then $\mu_1 \gg f_1 \sim \mu_2 \gg f_2 : S$

**Definition (Refinement Coupling).** Let $\mu_1 \in \mathcal{D}(A), \mu_2 \in \mathcal{D}(B)$. A sub-distribution $\mu \in \mathcal{D}(A \times B)$ is a refinement coupling of $\mu_1$ and $\mu_2$ if

1. $\forall a. \sum_{b \in B} \mu(a, b) = \mu_1(a)$
2. $\forall b. \sum_{a \in A} \mu(a, b) \leq \mu_2(b)$

Given relation $R : A \times B$ we say $\mu$ is an $R$-refinement-coupling if furthermore $\text{supp}(\mu) \subseteq R$. We write $\mu_1 \preceq \mu_2 : R$ if there exists an $R$-refinement-coupling of $\mu_1$ and $\mu_2$. 
\[
\begin{align*}
\text{exec}_n(e, \sigma) & \triangleq \\
& \begin{cases} 
\text{ret}(e) & \text{if } e \in \text{Val} \\
0 & \text{if } e \notin \text{Val}, n = 0 \\
\text{step}(e, \sigma) \gg \text{exec}_{n-1}(e, \sigma) & \text{otherwise}
\end{cases} \\
\text{exec}(\rho)(v) & \triangleq \lim_{n \to \infty} \text{exec}_n(\rho)(v) \\
\text{exec}_\beta(\rho) & \triangleq \sum_v \text{exec}(\rho)(v)
\end{align*}
\]

**Lemma (Erasure).** If \( \sigma_1(i) \in \text{dom}(\sigma_1) \) then
\[
\text{exec}_n(e_1, \sigma_1) \sim (\text{step}_i(\sigma_1) \gg \lambda \sigma_2. \text{exec}_n(e_1, \sigma_2)) : (=)
\]

**Theorem (Adequacy).** Let \( \varphi : \text{Val} \times \text{Val} \to \text{Prop} \) be a predicate in the meta-logic. If
\[
\text{specCtx} \ast \text{spec}(e') \vdash \text{wp } e \{ v. \exists v'. \text{spec}(v') \ast \varphi(v, v') \}
\]
is provable in Clutch then \( \forall n. \text{exec}_n(e, \sigma) \preceq \text{exec}(e', \sigma') : \varphi \).
\[ \Delta \models e_1 \preceq e_2 : \tau \triangleq \forall K. \text{specCtx} \rightarrow \text{spec}(K[e_2]) \rightarrow \text{naTok}(\Delta) \rightarrow \wp e_1 \{v_1. \exists v_2. \text{spec}(K[v_2]) \ast \text{naTok}(\top) \ast \box{\[\tau\]} \Delta(v_1, v_2) \}
\]

\[ G(\rho) \triangleq \text{specInterp}_\bullet(\rho) \]

\[ \text{specInv} \triangleq \exists \rho, e, \sigma. \text{specInterp}_\circ(\rho) \ast \text{spec}_\bullet(e) \ast \text{heaps}(\sigma) \ast \text{execConf}_n(\rho)(e, \sigma) = 1 \]

\[ \text{specCtx} \triangleq \text{specInv} \land \text{spec} \]

\[ \wp_\Delta e_1 \{\Phi\} \triangleq (e_1 \in \text{Val} \land \box{\[\Phi\]} e_1) \lor (e_1 \notin \text{Val} \land \forall \sigma_1, \rho_1.
\]

\[ S(\sigma_1) \ast G(\rho_1) \rightarrow \Delta \models_0 \]

\[ \text{execCoup}(e_1, \sigma_1, \rho_1)(\lambda e_2, \sigma_2, \rho_2).
\]

\[ \Delta \models_0 S(\sigma_2) \ast G(\rho_2) \ast \wp_\Delta e_2 \{\Phi\}) \]