Iris: Higher-Order Concurrent Separation Logic

Lecture 21: Clutch: Reasoning about randomized programs in Iris

Philipp G. Haselwarter

Aarhus University, Denmark

December 5, 2023

Overview

Earlier:

- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$: e, $(h, e) \rightsquigarrow (h, e')$, and $(h, \mathcal{E}) \to (h', \mathcal{E}')$
- Basic Logic of Resources : $l \hookrightarrow v$, $P * Q$, $P \mathrel{-\!\!*} Q$, $\Gamma \mid P \vdash Q$
- Basic Separation Logic : $\{P\}\, e\, \{v.Q\}$: Prop, isList l xs, ADTs, foldr
- Later ($\triangleright$) and Persistent ($\square$) Modalities.
- Concurrency Intro, Invariants and Ghost State
- CAS, Spin Locks, Concurrent Counter Modules.
- Monotone Resource Algebra
- Case studies: Ticket Lock, Array Based Queuing Lock, and Stack with Helping
- More details of constructions, *e.g.*, weakest preconditions, *etc.*
- Logical Relations for safety & type abstraction in Iris

Today:

- Randomized programming
- Operational Semantics of $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$
- Contextual & logical refinement

Plan

- Part I
  - Why randomization?
  - A probabilistic language: syntax & operational semantics
  - Properties beyond functional correctness
  - Contextual refinement
  - A probabilistic refinement logic

Plan

- Part I
  - Why randomization?
  - A probabilistic language: syntax & operational semantics
  - Properties beyond functional correctness
  - Contextual refinement
  - A probabilistic refinement logic
- Part II
  - Case study: ElGamal encryption
  - Asynchronous coupling rules
  - Vistas

Randomization in CS: Some examples

- Random (number) generation
  - Simulations (physics, modelling), all of the below

Randomization in CS: Some examples

- Random (number) generation
  - Simulations (physics, modelling), all of the below
- Efficient approximate algorithms & data structures (Monte Carlo)
  - primality testing (Miller-Rabin), approximate counters, Bloom filters

Randomization in CS: Some examples

- Random (number) generation
  - Simulations (physics, modelling), all of the below
- Efficient approximate algorithms & data structures (Monte Carlo)
  - primality testing (Miller-Rabin), approximate counters, Bloom filters
- Efficient-on-average exact algorithms (Las Vegas)
  - quicksort, skip lists, treaps

Randomization in CS: Some examples

- Random (number) generation
  - Simulations (physics, modelling), all of the below
- Efficient approximate algorithms & data structures (Monte Carlo)
  - primality testing (Miller-Rabin), approximate counters, Bloom filters
- Efficient-on-average exact algorithms (Las Vegas)
  - quicksort, skip lists, treaps
- Impossible without randomisation
  - Distributed systems
    - symmetry breaking: dining philosophers, consensus
  - Cryptography
    - Public key encryption, (differential) privacy, security games

Goals

Prove functional correctness and probabilistic properties of randomized programs!

Goals

Prove functional correctness and probabilistic properties of randomized programs!

Examples of what has been formalised in Clutch:

1. functional correctness of quicksort,
2. security of the one-time pad,
3. security of the ElGamal public key scheme,
4. correctness of lazy hash functions via refinement,
5. correctness of large random integer generation via refinement.

We will see 1-3 in class; 4 and 5 can be found in the Clutch paper.

Functional correctness

Recall randomised quicksort:

```
rec qs $l$ :=
    let $n$ := length $l$ in
    if $n < 1$ then $l$ else
      let $i_p$ := rand $(n - 1)$ in
      let $(p, r)$ := list_remove_nth $l$ $i_p$ in
      let $(le, gt)$ := partition $r$ $p$ in
      let $(le_s, gt_s)$ := (qs $le$, qs $gt$) in
      $le_s \mathbin{+\!\!+} (p :: gt_s)$
```

Functional correctness

Recall randomised quicksort:

```
rec qs $l$ :=
    let $n$ := length $l$ in
    if $n < 1$ then $l$ else
      let $i_p$ := rand $(n-1)$ in
      let $(p, r)$ := list_remove_nth $l$ $i_p$ in
      let $(le, gt)$ := partition $r$ $p$ in
      let $(le_s, gt_s)$ := (qs $le$, qs $gt$) in
      $le_s \mathbin{+\!\!+} (p :: gt_s)$
```

We want to show:

$$\forall xs. \forall l. \{\text{isList } l\ xs\}\ \text{qs}\ l\ \{v. \exists xs', \text{isList } v\ xs' \wedge \text{isPermutation } xs'\ xs \wedge \text{isSorted } xs'\}$$

Functional correctness

Recall randomised quicksort:

```
rec qs l :=
    let n := length l in
    if n < 1 then l else
     let i_p := rand (n − 1) in
     let (p, r) := list_remove_nth l i_p in
     let (le, gt) := partition r p in
     let (le_s, gt_s) := (qs le, qs gt) in
     le_s ++ (p :: gt_s)
```

We want to show:

$$\forall xs. \forall l. \{\text{isList } l\ xs\}\ \text{qs}\ l\ \{v. \exists xs', \text{isList } v\ xs' \wedge \text{isPermutation } xs'\ xs \wedge \text{isSorted}\ xs'\}$$

What about rand? How do we expect the proof to go?

Probabilistic refinement

Prove that encryption with a one-time pad (OTP) hides information perfectly.

- OTP encryption:

$$
\begin{aligned}
&\mathsf{let}\ \mathsf{keygen} && := \mathsf{flip} \\
&\mathsf{let}\ \mathsf{enc} && := \mathsf{xor} \\
&\mathsf{let}\ \mathsf{xor}\ b_1\ b_2 && := \mathsf{if}\ \ b_1\ \mathsf{then}\ \ \mathsf{not}\ \ b_2\ \mathsf{else}\ \ b_2
\end{aligned}
$$

Probabilistic refinement

Prove that encryption with a one-time pad (OTP) hides information perfectly.

- OTP encryption:

$$\begin{aligned}
&\mathsf{let}\ \mathrm{keygen} && := \mathsf{flip} \\
&\mathsf{let}\ \mathrm{enc} && := \mathrm{xor} \\
&\mathsf{let}\ \mathrm{xor}\ b_1\ b_2 && := \mathsf{if}\ \ b_1\ \mathsf{then}\ \ \mathsf{not}\ \ b_2\ \mathsf{else}\ \ b_2
\end{aligned}$$

- Security definition:

$$\begin{aligned}
&\mathsf{let}\ \mathrm{otp} && := \lambda\, msg.\ \mathsf{let}\ key := \mathsf{flip}\ \ \mathsf{in}\ \mathrm{xor}\ \ key\ \ msg \\
&\mathsf{let}\ \mathrm{spec} && := \lambda\, msg.\ \ \mathsf{flip}
\end{aligned}$$

Probabilistic refinement

Prove that encryption with a one-time pad (OTP) hides information perfectly.

- OTP encryption:

$$\begin{aligned}
&\mathsf{let}\ \mathsf{keygen} && := \mathsf{flip} \\
&\mathsf{let}\ \mathsf{enc} && := \mathsf{xor} \\
&\mathsf{let}\ \mathsf{xor}\ b_1\ b_2 && := \mathsf{if}\ \ b_1\ \mathsf{then}\ \ \mathsf{not}\ \ b_2\ \mathsf{else}\ \ b_2
\end{aligned}$$

- Security definition:

$$\begin{aligned}
&\mathsf{let}\ \mathsf{otp} && := \lambda\, \mathit{msg}.\, \mathsf{let}\ \mathit{key} := \mathsf{flip}\ \mathsf{in}\ \mathsf{xor}\ \ \mathit{key}\ \ \mathit{msg} \\
&\mathsf{let}\ \mathsf{spec} && := \lambda\, \mathit{msg}.\ \mathsf{flip}
\end{aligned}$$

We want to show that for all b : bool, “otp b looks like spec b”.
This is not expressible as a Hoare triple!

NB: readily generalises to bitstrings of any size.

The language $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$

- Syntax: modify $\lambda_{\mathsf{ref,conc}}$ as follows

$$\begin{array}{llcl}
 & & & \vdots \\
\mathit{Exp} & e & ::= & \dots \mid \mathsf{rand}\, e \mid \cancel{\mathsf{fork}\,\{e\}} \\
\mathit{ECtx} & E & ::= & \dots \mid \mathsf{rand}\, E \\
\mathit{Heap} & h & \in & \mathit{Loc} \overset{\text{fin}}{\rightharpoonup} \mathit{Val} \\
\cancel{\mathit{TPool}} & \mathcal{E} & \in & \cancel{\mathbb{N} \overset{\text{fin}}{\rightharpoonup} \mathit{Exp}} \\
\mathit{Config} & \rho & ::= & (h, e)
\end{array}$$

- Syntactic sugar: write flip for the term $(0 = \mathsf{rand}\, 1)$

Operational semantics, I

- $\lambda_{\mathrm{ref,conc}}$: stepping *relation* $(h, e) \rightsquigarrow (h', e') \subseteq \mathit{Config} \times \mathit{Config}$
- $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$: stepping *function* $\mathrm{step} : \mathit{Config} \to \mathcal{D}(\mathit{Config})$

Operational semantics, I

- $\lambda_{\mathrm{ref,conc}}$: stepping *relation* $(h, e) \rightsquigarrow (h', e') \subseteq \mathit{Config} \times \mathit{Config}$
- $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$: stepping *function* $\mathsf{step} : \mathit{Config} \to \mathcal{D}(\mathit{Config})$

Definition (Sub-distribution)

A (discrete) *sub-distribution* over a countable set A is a function $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. We write $\mathcal{D}(A)$ for the set of all sub-distributions over A.

Operational semantics, I

- $\lambda_{\mathrm{ref,conc}}$: stepping *relation* $(h, e) \leadsto (h', e') \subseteq \mathit{Config} \times \mathit{Config}$
- $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$: stepping *function* $\mathsf{step} : \mathit{Config} \to \mathcal{D}(\mathit{Config})$

Definition (Sub-distribution)

A (discrete) *sub-distribution* over a countable set A is a function $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. We write $\mathcal{D}(A)$ for the set of all sub-distributions over A.

Write $(h, e) \to^{p} (h', e')$ if $\mathsf{step}(h, e)(h', e') = p$.

- for deterministic reductions $(h, e) \leadsto (h', e')$:
  $(h, e) \to^{1} (h', e')$, i.e. $\mathsf{step}(h, e)(h', e') = 1$, and 0 elsewhere,
- $\mathsf{step}(h, \mathsf{rand}\ N)(h, k) = \frac{1}{N+1}$ for $0 \leq k \leq N$.

Operational semantics, I

- $\lambda_{\mathrm{ref,conc}}$: stepping *relation* $(h, e) \rightsquigarrow (h', e') \subseteq \mathit{Config} \times \mathit{Config}$
- $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$: stepping *function* $\mathsf{step} : \mathit{Config} \to \mathcal{D}(\mathit{Config})$

Definition (Sub-distribution)

A (discrete) *sub-distribution* over a countable set A is a function $\mu : A \to [0, 1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. We write $\mathcal{D}(A)$ for the set of all sub-distributions over A.

Write $(h, e) \to^{p} (h', e')$ if $\mathsf{step}(h, e)(h', e') = p$.

- for deterministic reductions $(h, e) \rightsquigarrow (h', e')$:
  $(h, e) \to^{1} (h', e')$, i.e. $\mathsf{step}(h, e)(h', e') = 1$, and 0 elsewhere,
- $\mathsf{step}(h, \mathsf{rand}\ N)(h, k) = \frac{1}{N+1}$ for $0 \leq k \leq N$.

E.g. $\mathsf{step}(h, \mathsf{rand}\ 0) = \{(h, 0) \mapsto 1\}$,
and $\mathsf{step}(h, \mathsf{rand}\ 1) = \{(h, 0) \mapsto 1/2,\ (h, 1) \mapsto 1/2\}$.

Pure reduction: same as for $\lambda_{\mathrm{ref,conc}}$

$$v \circledcirc v' \overset{\text{pure}}{\rightsquigarrow} v'' \qquad \text{if } v'' = v \circledcirc v'$$

$$\mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2 \overset{\text{pure}}{\rightsquigarrow} e_1$$

$$\mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2 \overset{\text{pure}}{\rightsquigarrow} e_2$$

$$\pi_i\,(v_1, v_2) \overset{\text{pure}}{\rightsquigarrow} v_i$$

$$\begin{pmatrix} \mathsf{match\ inj}_i\ v\ \mathsf{with} \\ \quad \mathsf{inj}_1\ x_1 \Rightarrow e_1 \\ \mid \mathsf{inj}_2\ x_2 \Rightarrow e_2 \\ \mathsf{end} \end{pmatrix} \overset{\text{pure}}{\rightsquigarrow} e_i[v/x_i]$$

$$(\mathsf{rec}\ f\ \ x := e)\ v \overset{\text{pure}}{\rightsquigarrow} e[(\mathsf{rec}\ f\ \ x := e)/f, v/x]$$

Configuration reduction

$$
\begin{aligned}
(h, e) &\rightarrow^1 (h, e') && \text{if } e \overset{\text{pure}}{\rightsquigarrow} e' \\
(h, \mathsf{ref}\,(v)) &\rightarrow^1 (h[\ell \mapsto v], \ell) && \text{where } \ell = \mathsf{freshLoc}(\mathrm{dom}(h)) \\
(h, !\,\ell) &\rightarrow^1 (h, h(\ell)) && \text{if } \ell \in \mathrm{dom}(h) \\
(h, \ell \leftarrow v) &\rightarrow^1 (h[\ell \mapsto v], ()) && \text{if } \ell \in \mathrm{dom}(h) \\
(h, \mathsf{rand}\ N) &\rightarrow^p (h, k) && p = \frac{1}{N+1} \text{ and } 0 \leq k \leq N \\
(h, E[e]) &\rightarrow^p (h', E[e']) && \text{if } \ (h, e) \rightarrow^p (h', e')
\end{aligned}
$$

Aside: Probabilities and concurrency

- the $\rightsquigarrow$ relation of $\lambda_{\mathrm{ref,conc}}$ can relate a reducible expression to more than one reduct: $([\ell \mapsto 7][0 \mapsto (\ell \leftarrow 42),\ 1 \mapsto !\,\ell]) \rightsquigarrow$
$$\begin{cases} ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto !\,\ell]) \rightsquigarrow ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto 42]) \\ ([\ell \mapsto 7][0 \mapsto (\ell \leftarrow 42),\ 1 \mapsto 7]) \rightsquigarrow ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto 7]) \end{cases}$$
- could we have used a relation on $\mathit{Config} \times \mathcal{D}(\mathit{Config})$?
- maybe: we can find a monadic structure to define program evaluation
- but: no canonical solution (equational theory “not as expected”) — the “right answer” depends on the application.

Aside: Probabilities and concurrency

- the $\rightsquigarrow$ relation of $\lambda_{\text{ref,conc}}$ can relate a reducible expression to more than one reduct: $([\ell \mapsto 7][0 \mapsto (\ell \leftarrow 42),\ 1 \mapsto {!}\,\ell]) \rightsquigarrow$
$$\begin{cases} ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto {!}\,\ell]) \rightsquigarrow ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto 42]) \\ ([\ell \mapsto 7][0 \mapsto (\ell \leftarrow 42),\ 1 \mapsto 7]) \rightsquigarrow ([\ell \mapsto 42][0 \mapsto (),\ 1 \mapsto 7]) \end{cases}$$
- could we have used a relation on $\mathit{Config} \times \mathcal{D}(\mathit{Config})$?
- maybe: we can find a monadic structure to define program evaluation
- but: no canonical solution (equational theory "not as expected") — the "right answer" depends on the application.

Is banishing fork $\{e\}$ enough?

Operational semantics, II

How do we iterate the step : $\mathit{Config} \to \mathcal{D}(\mathit{Config})$ function to *evaluate* programs?

Operational semantics, II

How do we iterate the step : $\mathit{Config} \to \mathcal{D}(\mathit{Config})$ function to *evaluate* programs?

Lemma (Probability Monad)

Let $\mu \in \mathcal{D}(A)$, $a \in A$, and $f : A \to \mathcal{D}(B)$. Then

1. $\mathrm{bind}(f, \mu)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$
2. $\mathrm{ret}(a)(a') \triangleq 1$ *if* $a = a'$, 0 *otherwise*

gives monadic structure to $\mathcal{D}$. We write $\mu \gg\!= f$ for $\mathrm{bind}(f, \mu)$*.*

We can chain steps together with bind.

Operational semantics, III

Definition (n-step execution)

$$\mathrm{exec}_n(e, \sigma) \triangleq \begin{cases} \mathbf{0} & \text{if } e \notin \mathit{Val} \text{ and } n = 0 \\ \mathrm{ret}(e) & \text{if } e \in \mathit{Val} \\ \mathrm{step}(e, \sigma) \ggg= \mathrm{exec}_{(n-1)} & \text{otherwise} \end{cases}$$

Operational semantics, III

Definition (*n*-step execution)

$$\mathrm{exec}_n(e, \sigma) \triangleq \begin{cases} \mathbf{0} & \text{if } e \notin \mathit{Val} \text{ and } n = 0 \\ \mathrm{ret}(e) & \text{if } e \in \mathit{Val} \\ \mathrm{step}(e, \sigma) \gg\!= \mathrm{exec}_{(n-1)} & \text{otherwise} \end{cases}$$

We can take the limit since exec is monotone and bounded.

$$\mathrm{exec}(\rho)(v) \triangleq \lim_{n \to \infty} \mathrm{exec}_n(\rho)(v)$$

A program thus induces a distribution on values.

Example: flip, I

Let $\rho = ([], \mathit{flip}) \triangleq ([], 0 = \mathsf{rand}\, 1)$. Execution of ρ yields the following:

$$\begin{aligned} \mathsf{exec}(\rho) &= \lim_{n\to\infty} \mathsf{exec}_n((\rho)) = \mathsf{exec}_2(\rho) \\ &= \mathsf{step}(\rho) \ggg= (\lambda\rho' \,.\, \mathsf{step}\,\rho' \ggg= (\lambda(h'', e'').\ \mathsf{ret}\, e'' \text{ if } e'' \in \mathit{Val} \text{ else } \mathbf{0})) \quad (*) \end{aligned}$$

Example: flip, I

Let $\rho = ([], \mathit{flip}) \triangleq ([], 0 = \mathsf{rand}\,1)$. Execution of ρ yields the following:

$$\begin{aligned}\mathsf{exec}(\rho) &= \lim_{n\to\infty} \mathsf{exec}_n((\rho)) = \mathsf{exec}_2(\rho) \\ &= \mathsf{step}(\rho) \gg\!= (\lambda\rho' \,.\, \mathsf{step}\,\rho' \gg\!= (\lambda(h'', e'').\ \mathsf{ret}\, e'' \text{ if } e'' \in \mathit{Val} \text{ else } \mathbf{0})) \quad (*)\end{aligned}$$

We compute: $\mathsf{step}([], 0 = \mathsf{rand}\,1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$,

$\mathsf{step}([], 0 = 0) = \{([], \mathsf{true}) \mapsto 1,\ \_ \mapsto 0\}$, $\quad \mathsf{step}([], 0 = 1) = \{([], \mathsf{false}) \mapsto 1,\ \_ \mapsto 0\}$

Example: flip, I

Let $\rho = ([], \mathit{flip}) \triangleq ([], 0 = \mathsf{rand}\, 1)$. Execution of ρ yields the following:

$$\begin{aligned}\mathrm{exec}(\rho) &= \lim_{n\to\infty} \mathrm{exec}_n((\rho)) = \mathrm{exec}_2(\rho) \\ &= \mathrm{step}(\rho) \gg\!= (\lambda\rho' \,.\, \mathrm{step}\,\rho' \gg\!= (\lambda(h'', e'').\ \mathrm{ret}\, e'' \text{ if } e'' \in \mathit{Val} \text{ else } \mathbf{0})) \quad (*)\end{aligned}$$

We compute: $\mathrm{step}([], 0 = \mathsf{rand}\, 1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$,

$\mathrm{step}([], 0 = 0) = \{([], \mathsf{true}) \mapsto 1,\ \_ \mapsto 0\}$, $\quad \mathrm{step}([], 0 = 1) = \{([], \mathsf{false}) \mapsto 1,\ \_ \mapsto 0\}$

$$\begin{aligned}(*) &= \mathrm{step}(\rho) \gg\!= \lambda\rho' \,.\, \lambda v \,.\, \sum_{(h'', e'') \in \mathit{Config}} \mathrm{step}(\rho')((h'', e'')) \cdot ((\mathrm{ret}\, e'' \text{ if } e'' \in \mathit{Val} \text{ else } \mathbf{0})\ v) \\ &= \mathrm{step}(\rho) \gg\!= \lambda\rho' \,.\, \lambda v \,.\, \sum_{(h'', e'') \in \mathit{Config}} \mathrm{step}(\rho')((h'', e'')) \cdot (1 \text{ if } e'' = v \text{ else } 0) \\ &\ldots \\ &= \{\mathsf{true} \mapsto 0.5,\ \mathsf{false} \mapsto 0.5,\ \_ \mapsto 0\}\end{aligned}$$

Example: flip, I

Let $\rho = ([], \mathit{flip}) \triangleq ([], 0 = \mathsf{rand}\,1)$. Execution of ρ yields the following:

$$\begin{aligned}\mathrm{exec}(\rho) &= \lim_{n\to\infty} \mathrm{exec}_n((\rho)) = \mathrm{exec}_2(\rho)\\ &= \mathrm{step}(\rho) \gg\!= (\lambda\rho' \,.\, \mathrm{step}\,\rho' \gg\!= (\lambda(h'', e'').\ \mathrm{ret}\, e''\ \mathrm{if}\ e'' \in \mathit{Val}\ \mathrm{else}\ \mathbf{0})) \quad (*)\end{aligned}$$

We compute: $\mathrm{step}([], 0 = \mathsf{rand}\,1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$,

$\mathrm{step}([], 0 = 0) = \{([], \mathsf{true}) \mapsto 1,\ \_ \mapsto 0\}$, $\quad \mathrm{step}([], 0 = 1) = \{([], \mathsf{false}) \mapsto 1,\ \_ \mapsto 0\}$

$$\begin{aligned}(*) &= \mathrm{step}(\rho) \gg\!= \lambda\rho' \,.\, \lambda v \,.\, \sum_{(h'',e'')\in \mathit{Config}} \mathrm{step}(\rho')((h'', e'')) \cdot ((\mathrm{ret}\, e''\ \mathrm{if}\ e'' \in \mathit{Val}\ \mathrm{else}\ \mathbf{0})\ v)\\ &= \mathrm{step}(\rho) \gg\!= \lambda\rho' \,.\, \lambda v \,.\, \sum_{(h'',e'')\in \mathit{Config}} \mathrm{step}(\rho')((h'', e'')) \cdot (1\ \mathrm{if}\ e'' = v\ \mathrm{else}\ 0)\\ &\dots\\ &= \{\mathsf{true} \mapsto 0.5,\ \mathsf{false} \mapsto 0.5,\ \_ \mapsto 0\}\end{aligned}$$

What would have happened if we'd only run exec for one step?

Example: flip, II

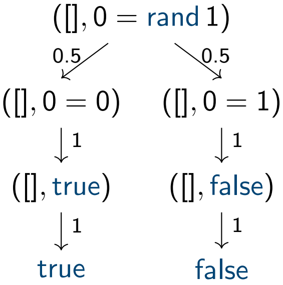
$$\mathrm{step}([], 0 = \mathsf{rand}\, 1) =$$

Example: flip, II

$$\mathsf{step}([], 0 = \mathsf{rand}\,1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$$

Example: flip, II

$$\mathsf{step}([], 0 = \mathsf{rand}\,1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$$



$([], 0 = \mathsf{rand}\,1)$

0.5

0.5

$([], 0 = 0)$

$([], 0 = 1)$

1

1

$([], \mathsf{true})$

$([], \mathsf{false})$

1

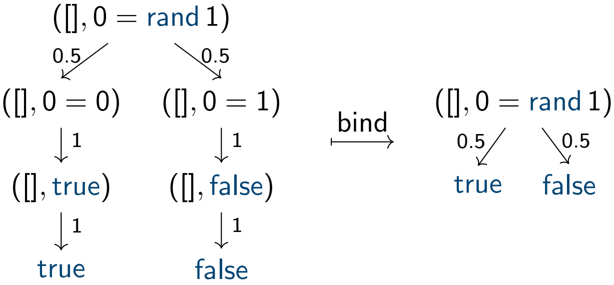
1

true

false

Example: flip, II

$$\text{step}([], 0 = \text{rand}\,1) = \{([], 0 = 0) \mapsto 0.5,\ ([], 0 = 1) \mapsto 0.5,\ \_ \mapsto 0\}$$


$([], 0 = \text{rand}\,1)$

0.5

0.5

$([], 0 = 0)$

$([], 0 = 1)$

1

1

$([], \text{true})$

$([], \text{false})$

1

1

true

false

bind

$([], 0 = \text{rand}\,1)$

0.5

0.5

true

false

What would have happened if we'd only run exec for one step?

Example: flip, III

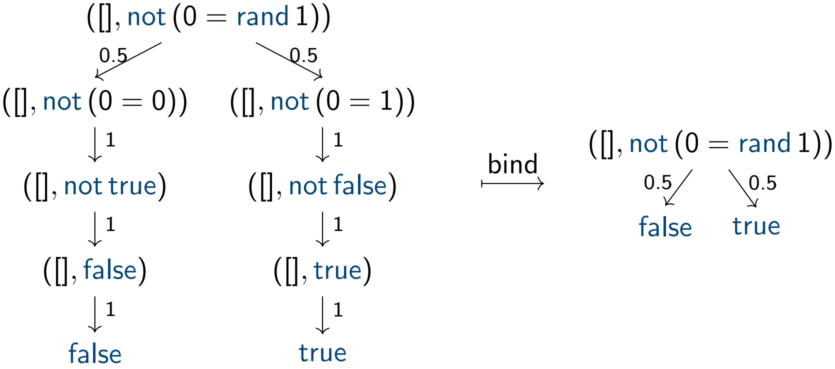
How does flip relate to not flip?

$$\begin{tikzcd}
 & ([], \mathsf{not}\,(0 = \mathsf{rand}\,1)) \arrow[dl, "0.5"'] \arrow[dr, "0.5"] & \\
([], \mathsf{not}\,(0 = 0)) \arrow[d, "1"] & & ([], \mathsf{not}\,(0 = 1)) \arrow[d, "1"] \\
([], \mathsf{not}\,\mathsf{true}) \arrow[d, "1"] & & ([], \mathsf{not}\,\mathsf{false}) \arrow[d, "1"] \\
([], \mathsf{false}) \arrow[d, "1"] & & ([], \mathsf{true}) \arrow[d, "1"] \\
\mathsf{false} & & \mathsf{true}
\end{tikzcd}$$

$$\xmapsto{\text{bind}}$$

Example: flip, III

How does flip relate to not flip?



$([], \mathsf{not}\,(0 = \mathsf{rand}\,1))$

0.5

0.5

$([], \mathsf{not}\,(0 = 0))$

$([], \mathsf{not}\,(0 = 1))$

1

1

$([], \mathsf{not}\,\mathsf{true})$

$([], \mathsf{not}\,\mathsf{false})$

1

1

$([], \mathsf{false})$

$([], \mathsf{true})$

1

1

false

true

$\xmapsto{\text{bind}}$

$([], \mathsf{not}\,(0 = \mathsf{rand}\,1))$

0.5

0.5

false

true

Example: adding two random numbers

$$([], \mathsf{rand}\ N + \mathsf{rand}\ K)$$

Example: adding two random numbers

$$([], \textsf{rand}\ N + \textsf{rand}\ K) \rightarrow_{\frac{1}{N+1}} ([], i + \textsf{rand}\ K)$$

Example: adding two random numbers

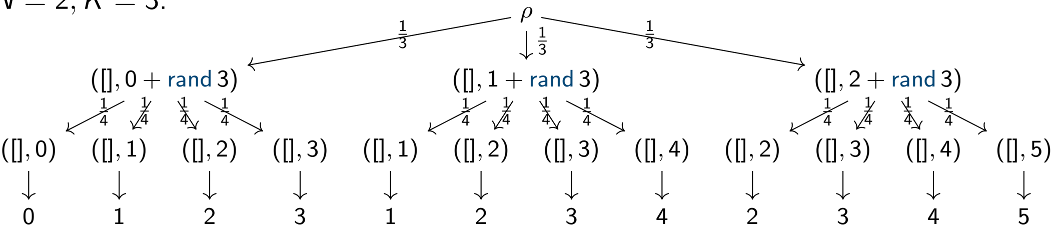
$$([], \mathsf{rand}\ N + \mathsf{rand}\ K) \rightarrow_{\frac{1}{N+1}} ([], i + \mathsf{rand}\ K) \rightarrow_{\frac{1}{K+1}} ([], i + j)$$

For $N = 2, K = 3$:

Example: adding two random numbers

$$([], \mathsf{rand}\ N + \mathsf{rand}\ K) \rightarrow_{\frac{1}{N+1}} ([], i + \mathsf{rand}\ K) \rightarrow_{\frac{1}{K+1}} ([], i + j)$$

For $N = 2, K = 3$:



ρ

$\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{3}$

$([], 0 + \mathsf{rand}\ 3)$ $([], 1 + \mathsf{rand}\ 3)$ $([], 2 + \mathsf{rand}\ 3)$

$\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$

$([], 0)$ $([], 1)$ $([], 2)$ $([], 3)$ $([], 1)$ $([], 2)$ $([], 3)$ $([], 4)$ $([], 2)$ $([], 3)$ $([], 4)$ $([], 5)$

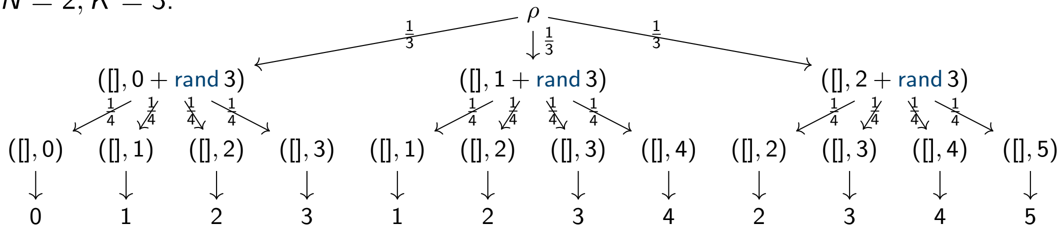
0 1 2 3 1 2 3 4 2 3 4 5

$\xmapsto{\text{bind}}$

Example: adding two random numbers

$$([], \mathsf{rand}\ N + \mathsf{rand}\ K) \rightarrow_{\frac{1}{N+1}} ([], i + \mathsf{rand}\ K) \rightarrow_{\frac{1}{K+1}} ([], i + j)$$

For $N = 2, K = 3$:



ρ

$\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{3}$

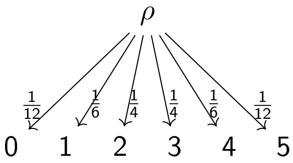
$([], 0 + \mathsf{rand}\ 3)$ $([], 1 + \mathsf{rand}\ 3)$ $([], 2 + \mathsf{rand}\ 3)$

$\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$

$([], 0)$ $([], 1)$ $([], 2)$ $([], 3)$ $([], 1)$ $([], 2)$ $([], 3)$ $([], 4)$ $([], 2)$ $([], 3)$ $([], 4)$ $([], 5)$

0 1 2 3 1 2 3 4 2 3 4 5

$\xmapsto{\text{bind}}$



ρ

$\frac{1}{12}$ $\frac{1}{6}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{6}$ $\frac{1}{12}$

0 1 2 3 4 5

Example: recursion

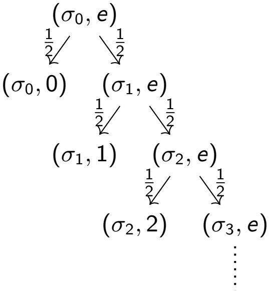
Let ℓ be a location and write σ_i for the heap $[\ell \mapsto i]$.

Define $e \triangleq (\mathsf{rec}\ f\ \_ := \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ !\,\ell\ \mathsf{else}\ (\ell \leftarrow !\,\ell + 1; f\,()))\,()$.

Example: recursion

Let ℓ be a location and write σ_i for the heap $[\ell \mapsto i]$.

Define $e \triangleq (\text{rec } f \;\_ := \text{if flip then } !\,\ell \text{ else } (\ell \leftarrow !\,\ell + 1; f\,()))\,()$.

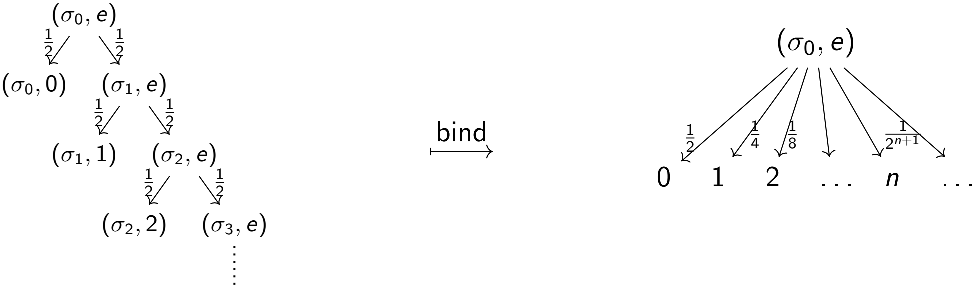


$$\xmapsto{\text{bind}}$$

Example: recursion

Let ℓ be a location and write σ_i for the heap $[\ell \mapsto i]$.

Define $e \triangleq (\mathsf{rec}\ f\ \_ := \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ !\,\ell\ \mathsf{else}\ (\ell \leftarrow !\,\ell + 1; f\,()))\,()$.



The result is an infinite tree (distribution on values).

Reasoning about $\mathbf{F}_{\mu,\mathrm{ref}}^{\mathrm{rand}}$ programs: correctness

```
rec qs $l$ :=
    let $n$ := length $l$ in
    if $n < 1$ then $l$ else
      let $i_p$ := rand $(n - 1)$ in
      let $(p, r)$ := list_remove_nth $l$ $i_p$ in
      let $(le, gt)$ := partition $r$ $p$ in
      let $(le_s, gt_s)$ := (qs $le$, qs $gt$) in
      $le_s \mathbin{+\!\!+} (p :: gt_s)$
```

For quicksort, the following is enough:

$$\frac{S \vdash \forall k, 0 \leq k \leq N \Rightarrow \{P\}\, k\, \{Q\}}{S \vdash \{P\}\ \mathsf{rand}\, N\, \{Q\}}$$

Note: Q is a predicate on *values*, not distributions on values!

$$\forall xs. \forall l. \{\mathrm{isList}\ l\ xs\}\ \mathrm{qs}\ l\ \{v. \exists xs', \mathrm{isList}\ v\ xs' \wedge \mathrm{isPermutation}\ xs'\ xs \wedge \mathrm{isSorted}\ xs'\}$$

Reasoning about $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ programs: contextual refinement

$$\begin{array}{llll}
\mathsf{let}\ \mathsf{keygen} & := \mathsf{flip} & \mathsf{let}\ \mathsf{otp} & := \lambda\, \mathit{msg}.\, \mathsf{let}\ \mathit{key} := \mathsf{flip}\ \mathsf{in}\ \mathsf{xor}\ \mathit{key}\ \mathit{msg} \\
\mathsf{let}\ \mathsf{enc} & := \mathsf{xor} & \mathsf{let}\ \mathsf{spec} & := \lambda\, \mathit{msg}.\ \mathsf{flip} \\
\mathsf{let}\ \mathsf{xor}\ b_1\ b_2 & := \mathsf{if}\ b_1\ \mathsf{then}\ \mathsf{not}\ b_2\ \mathsf{else}\ b_2 & &
\end{array}$$

We want to show that otp exhibits the same *observable behaviour* as spec.

Reasoning about $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ programs: contextual refinement

$$\begin{array}{lll@{\qquad}lll}
\mathsf{let}\ \mathsf{keygen} & := & \mathsf{flip} & \mathsf{let}\ \mathsf{otp} & := & \lambda\, \mathit{msg}\,.\ \mathsf{let}\ \mathit{key} := \mathsf{flip}\ \mathsf{in}\ \mathsf{xor}\ \mathit{key}\ \mathit{msg} \\
\mathsf{let}\ \mathsf{enc} & := & \mathsf{xor} & \mathsf{let}\ \mathsf{spec} & := & \lambda\, \mathit{msg}\,.\ \mathsf{flip} \\
\mathsf{let}\ \mathsf{xor}\ b_1\ b_2 & := & \mathsf{if}\ \ b_1\ \mathsf{then}\ \ \mathsf{not}\ \ b_2\ \mathsf{else}\ \ b_2 & & &
\end{array}$$

We want to show that otp exhibits the same *observable behaviour* as spec.

Let $\mathcal{C}$ be a well-typed program context, i.e. a program of type τ' with a hole of type τ. Let's fix $\tau' = \mathsf{bool}$.

$$\Theta \mid \Gamma \vdash e_1 \precsim_{\mathsf{ctx}} e_2 : \tau \quad \triangleq \quad \forall (\mathcal{C} : (\Theta \mid \Gamma \vdash \tau) \Rightarrow (\emptyset \mid \emptyset \vdash \mathsf{bool})),\ \sigma : \mathit{State},\ b : \mathsf{bool}\,.$$
$$\mathsf{exec}(\mathcal{C}[e_1], \sigma)\ b \leq \mathsf{exec}(\mathcal{C}[e_2], \sigma)\ b$$

Reasoning about $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ programs: contextual refinement

$$
\begin{array}{lll}
\mathsf{let}\ \mathsf{keygen} & := \mathsf{flip} & \mathsf{let}\ \mathsf{otp} & := \lambda\, \mathit{msg}.\ \mathsf{let}\ \mathit{key} := \mathsf{flip}\ \mathsf{in}\ \mathsf{xor}\ \mathit{key}\ \mathit{msg} \\
\mathsf{let}\ \mathsf{enc} & := \mathsf{xor} & \mathsf{let}\ \mathsf{spec} & := \lambda\, \mathit{msg}.\ \mathsf{flip} \\
\mathsf{let}\ \mathsf{xor}\ b_1\ b_2 & := \mathsf{if}\ b_1\ \mathsf{then}\ \mathsf{not}\ b_2\ \mathsf{else}\ b_2 & &
\end{array}
$$

We want to show that otp exhibits the same *observable behaviour* as spec.

Let $\mathcal{C}$ be a well-typed program context, i.e. a program of type τ' with a hole of type τ. Let's fix $\tau' = \mathsf{bool}$.

$$
\Theta \mid \Gamma \vdash e_1 \precsim_{\mathsf{ctx}} e_2 : \tau \quad \triangleq \quad \forall (\mathcal{C} : (\Theta \mid \Gamma \vdash \tau) \Rightarrow (\emptyset \mid \emptyset \vdash \mathsf{bool})),\ \sigma : \mathit{State},\ b : \mathsf{bool}\,. \\
\mathsf{exec}(\mathcal{C}[e_1], \sigma)\ b \leq \mathsf{exec}(\mathcal{C}[e_2], \sigma)\ b
$$

How can we possibly reason about *all contexts* $\mathcal{C}$?

Reasoning about $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ programs: logical refinement

We define a binary logical relation in terms of a notion of weakest precondition for $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$, which supports relational reasoning:

$$\Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau$$

"In environment Δ, the expression e_1 refines the expression e_2 at type τ under the invariants in $\mathcal{E}$."

Reasoning about $\mathbf{F}_{\mu,\mathsf{ref}}^{\mathsf{rand}}$ programs: logical refinement

We define a binary logical relation in terms of a notion of weakest precondition for $\mathbf{F}_{\mu,\mathsf{ref}}^{\mathsf{rand}}$, which supports relational reasoning:

$$\Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau$$

"In environment Δ, the expression e_1 refines the expression e_2 at type τ under the invariants in $\mathcal{E}$."

Theorem (Soundness)

Let Ξ be a type variable context, and assume that, for all Δ assigning a relational interpretation to all type variables in Ξ, we can derive $\Delta \mid \Gamma \vDash e_1 \precsim e_2 : \tau$. Then $\Xi \mid \Gamma \vdash e_1 \precsim_{ctx} e_2 : \tau$

Logical refinement: Structural & deterministic rules

REL-PURE-L

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

REL-PURE-R

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

REL-ALLOC-L

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

REL-ALLOC-R

$$\frac{\forall \ell.\, \ell \mapsto_{\mathsf{s}} v \twoheadrightarrow \Delta \vDash_{\mathcal{E}} e_1 \precsim E[\ell] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\mathsf{ref}(v)] : \tau}$$

REL-LOAD-L

$$\frac{\ell \mapsto v \qquad \ell \mapsto v \twoheadrightarrow \Delta \vDash_{\mathcal{E}} E[v] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} E[!\ell] \precsim e_2 : \tau}$$

REL-LOAD-R

$$\frac{\ell \mapsto_{\mathsf{s}} v \qquad \ell \mapsto_{\mathsf{s}} v \twoheadrightarrow \Delta \vDash_{\mathcal{E}} e_1 \precsim E[v] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[!\ell] : \tau}$$

REL-STORE-L

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

REL-STORE-R

$$\frac{\ell \mapsto_{\mathsf{s}} v \qquad \ell \mapsto_{\mathsf{s}} w \twoheadrightarrow \Delta \vDash_{\mathcal{E}} e_1 \precsim E[()] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\ell \leftarrow w] : \tau}$$

REL-REC

$$\frac{\Box\,(\forall v_1, v_2.\, [\![\tau]\!]_{\Delta}(v_1, v_2) \twoheadrightarrow \Delta \vDash_{\top} (\mathsf{rec}\ f_1\ x_1 := e_1)\, v_1 \precsim (\mathsf{rec}\ f_2\ x_2 := e_2)\, v_2 : \sigma)}{\Delta \vDash_{\top} \mathsf{rec}\ f_1\ x_1 := e_1 \precsim \mathsf{rec}\ f_2\ x_2 := e_2 : \tau \to \sigma}$$

REL-RETURN

$$\frac{[\![\tau]\!]_{\Delta}(v_1, v_2)}{\Delta \vDash_{\top} v_1 \precsim v_2 : \tau}$$

REL-BIND

$$\frac{\Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau \qquad \forall v_1, v_2.\, [\![\tau]\!]_{\Delta}(v_1, v_2) \twoheadrightarrow \Delta \vDash_{\top} E[v_1] \precsim E'[v_2] : \sigma}{\Delta \vDash_{\mathcal{E}} E[e_1] \precsim E'[e_2] : \sigma}$$

Logical refinement: Probabilistic rules

REL-RAND-L

$$\frac{\forall n \leq N.\, \Delta \vDash_{\mathcal{E}} E[n] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} E[\,\mathsf{rand}(N)] \precsim e_2 : \tau}$$

REL-RAND-R

$$\frac{e_1 \notin Val \qquad \forall n \leq N.\, \Delta \vDash_{\mathcal{E}} e_1 \precsim E[b] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\,\mathsf{rand}(N)] : \tau}$$

REL-COUPLE-RANDS

$$\frac{f \text{ bijection} \qquad \forall n \leq N.\, \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[f(n)] : \tau}{\Delta \vDash_{\mathcal{E}} E[\,\mathsf{rand}(N)] \precsim E'[\,\mathsf{rand}(N)] : \tau}$$

One-time pad is secure

$$\begin{array}{lll@{\qquad}lll}
\text{let keygen} & := & \text{flip} & \text{let otp} & := & \lambda\, msg.\ \text{let}\ key := \text{flip}\ \text{in}\ \text{xor}\ key\ \ msg \\
\text{let enc} & := & \text{xor} & \text{let spec} & := & \lambda\, msg.\ \text{flip} \\
\text{let xor}\ b_1\ b_2 & := & \text{if}\ \ b_1\ \text{then}\ \ \text{not}\ \ b_2\ \text{else}\ \ b_2 & & &
\end{array}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{f\ \text{bij.} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\overline{[\![\text{bool}]\!]\,(\text{false}, \text{false})}}{\vDash\ \text{false} \precsim \text{false} : \text{bool}}}{\vDash\ (\text{xor true true}) \precsim (0 = 1) : \text{bool}} \quad \vdots}{\vDash\ (\text{xor true}\ b) \precsim (0 = f(0)) : \text{bool}} \qquad \dfrac{\vdots}{\vDash\ \text{xor false}\ b \precsim 0 = f(1) : \text{bool}}}{\forall\, 0 \le n \le 1\,.\ \vDash\ (\text{let}\ key := (0 = n)\ \text{in xor}\ key\ b) \precsim (0 = f(n)) : \text{bool}}}{\vDash\ (\text{let}\ key := (0 = \text{rand}\ 1)\ \text{in xor}\ key\ b) \precsim (0 = \text{rand}\ 1) : \text{bool}}}{\vDash\ \text{otp}\ b \precsim \text{spec}\ b : \text{bool}}}{\dfrac{\Box\, (\forall v_1, v_2.\ [\![\text{bool}]\!]\,(v_1, v_2) \twoheadrightarrow\!\!\ast\ \vDash_{\top}\ \text{otp}\ v_1 \precsim \text{spec}\ v_2 : \text{bool})}{\vDash_{\top}\ \text{otp} \precsim \text{spec} : \text{bool} \to \text{bool}}}$$

$$f(n) = \begin{cases} 0 & \text{if } (n = 0) \oplus b \\ 1 & \text{else} \end{cases}$$

Relational interpretation of types

$$
\begin{aligned}
\llbracket \alpha \rrbracket_\Delta(v_1, v_2) &\triangleq \Delta(\alpha)(v_1, v_2) \\
\llbracket \mathsf{unit} \rrbracket_\Delta(v_1, v_2) &\triangleq v_1 = v_2 = () \\
\llbracket \mathsf{int} \rrbracket_\Delta(v_1, v_2) &\triangleq \exists z \in \mathbb{Z}.\ v_1 = v_2 = z \\
\llbracket \mathsf{nat} \rrbracket_\Delta(v_1, v_2) &\triangleq \exists n \in \mathbb{N}.\ v_1 = v_2 = n \\
\llbracket \mathsf{bool} \rrbracket_\Delta(v_1, v_2) &\triangleq \exists b \in \mathbb{B}.\ v_1 = v_2 = b \\
\llbracket \tau \to \sigma \rrbracket_\Delta(v_1, v_2) &\triangleq \Box\,(\forall w_1, w_2.\ \llbracket \tau \rrbracket_\Delta(w_1, w_2) \mathrel{-\!\!*} \Delta \vDash v_1\ w_1 \precsim v_2\ w_2 : \sigma) \\
\llbracket \tau \times \sigma \rrbracket_\Delta(v_1, v_2) &\triangleq \exists w_1, w_1', w_2, w_2'.\ (v_1 = (w_1, w_1')) * (v_2 = (w_2, w_2')) * \llbracket \tau \rrbracket_\Delta(w_1, w_2) * \llbracket \sigma \rrbracket_\Delta(w_1', w_2') \\
\llbracket \tau + \sigma \rrbracket_\Delta(v_1, v_2) &\triangleq \exists w_1, w_2.\ (v_1 = \mathsf{inl}(w_1) * v_2 = \mathsf{inl}(w_2) * \llbracket \tau \rrbracket_\Delta(w_1, w_2)) \lor \\
&\qquad\qquad (v_1 = \mathsf{inr}(w_1) * v_2 = \mathsf{inr}(w_2) * \llbracket \sigma \rrbracket_\Delta(w_1, w_2)) \\
\llbracket \mu\,\alpha.\ \tau \rrbracket_\Delta(v_1, v_2) &\triangleq (\mu R.\ \lambda(v_1, v_2).\ \exists w_1, w_2.\ (v_1 = \mathsf{fold}\ w_1) * (v' = \mathsf{fold}\ w_2) * \triangleright \llbracket \tau \rrbracket_{\Delta, \alpha \mapsto R}(w_1, w_2))\ (v_1, v_2) \\
\llbracket \forall \alpha.\ \tau \rrbracket_\Delta(v_1, v_2) &\triangleq \Box\ (\forall R.\ (\Delta, \alpha \mapsto R \vDash v_1\ \_ \precsim v_2\ \_ : \tau) \\
\llbracket \exists \alpha.\ \tau \rrbracket_\Delta(v_1, v_2) &\triangleq \exists R, w_1, w_2.\ (v_1 = \mathsf{pack}\ w_2) * (v_2 = \mathsf{pack}\ w_2) * \llbracket \tau \rrbracket_{\Delta, \alpha \mapsto R}(w_1, w_2) \\
\llbracket \mathsf{ref}\ \tau \rrbracket_\Delta(v_1, v_2) &\triangleq \exists \ell_1, \ell_2.\ (v_1 = \ell_1) * (v_2 = \ell_2) * \boxed{\exists w_1, w_2.\ \ell_1 \mapsto w_1 * \ell_2 \mapsto_{\mathsf{s}} w_2 * \llbracket \tau \rrbracket_\Delta(w_1, w_2)}^{\mathcal{N}.\ell_1.\ell_2} \\
\llbracket \mathsf{tape} \rrbracket_\Delta(v_1, v_2) &\triangleq \exists \iota_1, \iota_2, N.\ (v_1 = \iota_1) * (v_2 = \iota_2) * \boxed{\iota_1 \hookrightarrow (N, \varepsilon) * \iota_2 \hookrightarrow_{\mathsf{s}} (N, \varepsilon)}^{\mathcal{N}.\iota_1.\iota_2}
\end{aligned}
$$