Almost-Sure Termination by Guarded Refinement

Simon Oddershede Gregersen[1]

joint work with
Alejandro Aguirre[2], Philipp G. Haselwarter[2], Joseph Tassarotti[1], and Lars Birkedal[2]

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

Most of the time, we want our programs to **terminate**…

Most of the time, we want our programs to **terminate**…

How do we prove it?

Most of the time, we want our programs to **terminate**...

How do we prove it? For **probabilistic** programs, the argument can be quite subtle.

Most of the time, we want our programs to **terminate**...

How do we prove it? For **probabilistic** programs, the argument can be quite subtle.

```
let r = ref true in
while !r do
  r ← flip
end
```

Most of the time, we want our programs to **terminate**...

How do we prove it? For **probabilistic** programs, the argument can be quite subtle.

```
let r = ref true in
while !r do
  r ← flip
end
```

The program **almost surely** terminates since $\text{lim}_{n\to\infty} 1 - \frac{1}{2}^n = 1$.

A New Proof Rule for Almost-Sure Termination

ANNABELLE MCIVER, Macquarie University, Australia
CARROLL MORGAN, University of New South Wales, Australia and Data61, CSIRO, Australia
BENJAMIN LUCIEN KAMINSKI, RWTH Aachen University, Germany and UCL, UK
JOOST-PIETER KATOEN, RWTH Aachen University, Germany and IST, Austria

We present a new proof rule for proving almost-sure termination of probabilistic programs, including those that contain demonic non-determinism.

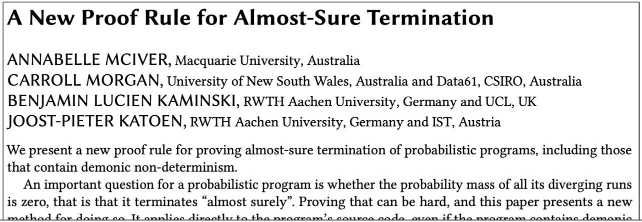
An important question for a probabilistic program is whether the probability mass of all its diverging runs is zero, that is that it terminates "almost surely". Proving that can be hard, and this paper presents a new method for doing so. It applies directly to the program's source code, even if the program contains demonic choice.

Like others, we use variant functions (a.k.a. "super-martingales") that are real-valued and decrease randomly on each loop iteration; but our key innovation is that the amount as well as the probability of the decrease are *parametric*. We prove the soundness of the new rule, indicate where its applicability goes beyond existing rules, and explain its connection to classical results on denumerable (non-demonic) Markov chains.

CCS Concepts: • **Theory of computation → Program verification**; *Probabilistic computation*; *Axiomatic semantics*;

Additional Key Words and Phrases: Almost-sure termination, demonic non-determinism, program logic pGCL.

ACM Reference Format:
Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A New Proof Rule for Almost-Sure Termination. *Proc. ACM Program. Lang.* 2, POPL, Article 33 (January 2018), 28 pages. https://doi.org/10.1145/3158121

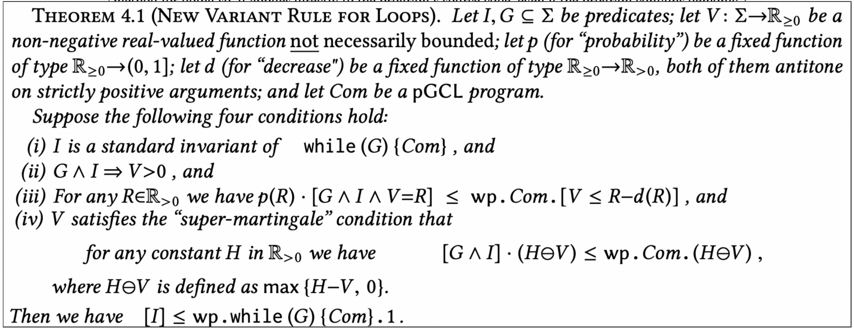


A New Proof Rule for Almost-Sure Termination

ANNABELLE MCIVER, Macquarie University, Australia
CARROLL MORGAN, University of New South Wales, Australia and Data61, CSIRO, Australia
BENJAMIN LUCIEN KAMINSKI, RWTH Aachen University, Germany and UCL, UK
JOOST-PIETER KATOEN, RWTH Aachen University, Germany and IST, Austria

We present a new proof rule for proving almost-sure termination of probabilistic programs, including those that contain demonic non-determinism.

An important question for a probabilistic program is whether the probability mass of all its diverging runs is zero, that is that it terminates "almost surely". Proving that can be hard, and this paper presents a new method for doing so. It applies directly to the program's source code, even if the program contains demonic



Theorem 4.1 (New Variant Rule for Loops). *Let* $I, G \subseteq \Sigma$ *be predicates; let* $V : \Sigma \rightarrow \mathbb{R}_{\geq 0}$ *be a non-negative real-valued function* <u>not</u> necessarily bounded*; let* p *(for "probability") be a fixed function of type* $\mathbb{R}_{\geq 0} \rightarrow (0, 1]$*; let* d *(for "decrease") be a fixed function of type* $\mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{>0}$*, both of them antitone on strictly positive arguments; and let* Com *be a* pGCL *program.*

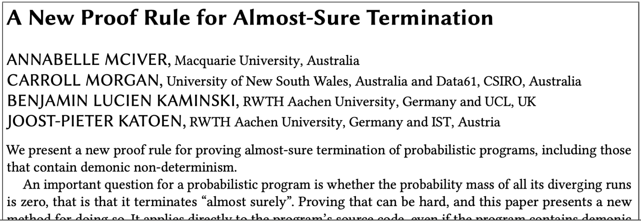
Suppose the following four conditions hold:

(i) *I is a standard invariant of* $\texttt{while}\,(G)\,\{Com\}$ *, and*
(ii) $G \wedge I \Rightarrow V > 0$ *, and*
(iii) *For any* $R \in \mathbb{R}_{>0}$ *we have* $p(R) \cdot [G \wedge I \wedge V = R] \;\leq\; \texttt{wp}.Com.[V \leq R - d(R)]$ *, and*
(iv) *V satisfies the "super-martingale" condition that*

$$\textit{for any constant } H \textit{ in } \mathbb{R}_{>0} \textit{ we have} \qquad [G \wedge I] \cdot (H \ominus V) \leq \texttt{wp}.Com.(H \ominus V)\,,$$

where $H \ominus V$ *is defined as* $\max\,\{H - V,\, 0\}$.

Then we have $[I] \leq \texttt{wp}.\texttt{while}\,(G)\,\{Com\}.1\,.$

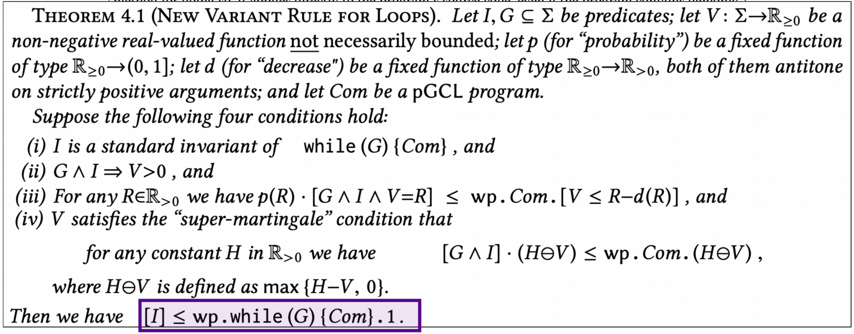


A New Proof Rule for Almost-Sure Termination

ANNABELLE MCIVER, Macquarie University, Australia
CARROLL MORGAN, University of New South Wales, Australia and Data61, CSIRO, Australia
BENJAMIN LUCIEN KAMINSKI, RWTH Aachen University, Germany and UCL, UK
JOOST-PIETER KATOEN, RWTH Aachen University, Germany and IST, Austria

We present a new proof rule for proving almost-sure termination of probabilistic programs, including those that contain demonic non-determinism.

An important question for a probabilistic program is whether the probability mass of all its diverging runs is zero, that is that it terminates "almost surely". Proving that can be hard, and this paper presents a new method for doing so. It applies directly to the program's source code, even if the program contains demonic



Theorem 4.1 (New Variant Rule for Loops). *Let* $I, G \subseteq \Sigma$ *be predicates; let* $V\colon \Sigma \to \mathbb{R}_{\geq 0}$ *be a non-negative real-valued function* <u>not</u> necessarily bounded*; let* p *(for "probability") be a fixed function of type* $\mathbb{R}_{\geq 0} \to (0,1]$*; let* d *(for "decrease") be a fixed function of type* $\mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$*, both of them antitone on strictly positive arguments; and let* Com *be a* pGCL *program.*

Suppose the following four conditions hold:

(i) I *is a standard invariant of* $\texttt{while}\,(G)\,\{Com\}$ *, and*
(ii) $G \wedge I \Rightarrow V{>}0$ *, and*
(iii) *For any* $R{\in}\mathbb{R}_{>0}$ *we have* $p(R) \cdot [G \wedge I \wedge V{=}R] \;\leq\; \texttt{wp}.Com.[V \leq R{-}d(R)]$ *, and*
(iv) V *satisfies the "super-martingale" condition that*

$$\textit{for any constant } H \textit{ in } \mathbb{R}_{>0} \textit{ we have} \qquad [G \wedge I] \cdot (H \ominus V) \leq \texttt{wp}.Com.(H \ominus V)\,,$$

where $H \ominus V$ *is defined as* $\max\{H{-}V,\, 0\}$.

Then we have $\boxed{[I] \leq \texttt{wp}.\texttt{while}\,(G)\,\{Com\}.1\,.}$

While successful, most existing works consider **first-order languages** and their solutions apply to **syntactic while loops**.

While successful, most existing works consider **first-order languages** and their solutions apply to **syntactic while loops**.

But what if we were to consider a higher-order language?

While successful, most existing works consider **first-order languages** and their solutions apply to **syntactic while loops**.

But what if we were to consider a higher-order language?

Multiple ways for the program to not terminate!

As a (somewhat extreme) example, consider

$$\begin{aligned}
\mathsf{fix} &\triangleq \lambda F.\, \mathsf{let}\; r = \mathsf{ref}\; (\lambda x.\, x)\; \mathsf{in}\; r \leftarrow (\lambda x.\, F\; (!\, r)\; x);\; !\, r \\
\mathsf{F} &\triangleq \lambda f.\, \lambda n.\, \mathsf{if}\; n == 0\; \mathsf{then}\; () \\
&\qquad\qquad\qquad \mathsf{else\;if\;\; flip\;\; then}\; f\; (n - 1)\; \mathsf{else}\; f\; (n + 1) \\
\mathsf{walk} &\triangleq \mathsf{fix}\; \mathsf{F}
\end{aligned}$$

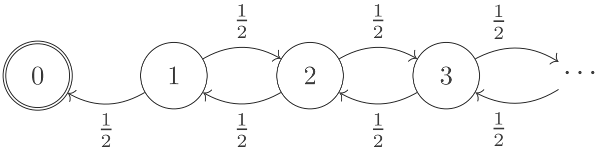
By tying Landin’s knot, we can encode a fixed-point combinator and thus recurse.

As a (somewhat extreme) example, consider

$$\begin{aligned}
\mathsf{fix} &\triangleq \lambda F.\, \mathsf{let}\; r = \mathsf{ref}\; (\lambda x.\, x)\; \mathsf{in}\; r \leftarrow (\lambda x.\, F\; (!\, r)\; x);\; !\, r \\
\mathsf{F} &\triangleq \lambda f.\, \lambda n.\, \mathsf{if}\; n == 0\; \mathsf{then}\; () \\
&\qquad\qquad\quad \mathsf{else\; if}\;\; \mathsf{flip}\;\; \mathsf{then}\; f\; (n-1)\; \mathsf{else}\; f\; (n+1) \\
\mathsf{walk} &\triangleq \mathsf{fix}\; \mathsf{F}
\end{aligned}$$

By tying Landin’s knot, we can encode a fixed-point combinator and thus recurse.

In essence, however, the termination argument is well known.

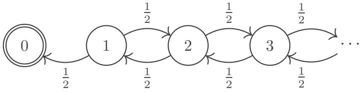


This work

A higher-order separation logic, Caliper, for **termination-preserving refinement** between probabilistic programs and probabilistic transition systems.

For example, to show that walk(n) terminates we show the refinement

$$\text{walk}(n) \quad \preceq$$



As a consequence, by showing that the model terminates, so does the program.

Caliper

Two key components:

- A **refinement weakest precondition** $\mathrm{rwp}\ e\ \{\Phi\}$ for reasoning about programs,
- A **separation logic resource** $\mathrm{spec}(m)$ for tracking the current model state.

Theorem (Soundness)

If $\mathrm{spec}(m) \vdash \mathrm{rwp}\ e\ \{\Phi\}$ *then* $\mathit{exec}_{\Downarrow}(m) \leq \mathit{exec}_{\Downarrow}(e)$.

Caliper cont'd

The program logic satisfies the typical separation logic rules, *e.g.*,

$$
\begin{aligned}
\forall \ell.\, \ell \mapsto v \mathrel{-\!\!*} \Phi(\ell) &\vdash \mathsf{rwp}\ \mathsf{ref}\, v\ \{\Phi\} && \text{(wp-alloc)} \\
(\ell \mapsto v \mathrel{-\!\!*} \Phi(v)) * \ell \mapsto v &\vdash \mathsf{rwp}\ !\,\ell\ \{\Phi\} && \text{(wp-load)} \\
(\ell \mapsto w \mathrel{-\!\!*} \Phi(())) * \ell \mapsto v &\vdash \mathsf{rwp}\ \ell \leftarrow w\ \{\Phi\} && \text{(wp-store)} \\
\mathsf{rwp}\ e\ \Big\{v.\, \mathsf{rwp}\ K[v]\ \{\Phi\}\Big\} &\vdash \mathsf{rwp}\ K[e]\ \{\Phi\} && \text{(wp-bind)} \\
&\vdots && \vdots
\end{aligned}
$$

...but there is **no rule for reasoning about recursion or loops!**

Caliper cont’d

Instead, Caliper makes use of **guarded recursion** with the **later modality** and, in particular, the Löb induction principle.

$$\frac{\triangleright P \vdash P}{\vdash P}$$

Caliper cont’d

Instead, Caliper makes use of **guarded recursion** with the **later modality** and, in particular, the Löb induction principle.

$$\frac{\triangleright P \vdash P}{\vdash P}$$

Key idea

By only allowing later modalities to be eliminated when the **model** makes a transition, we **preserve termination** across the refinement relation.

Later elimination

The simplest case is when the model makes a deterministic transition:

$$\frac{m_1 \rightarrow^1 m_2 \qquad \mathsf{spec}(m_2) * P \vdash \mathsf{rwp}\; e\; \{\Phi\}}{\mathsf{spec}(m_1) * \triangleright P \vdash \mathsf{rwp}\; e\; \{\Phi\}}$$

Later elimination

The simplest case is when the model makes a deterministic transition:

$$\frac{m_1 \rightarrow^{1} m_2 \qquad \mathsf{spec}(m_2) * P \vdash \mathsf{rwp}\ e\ \{\Phi\}}{\mathsf{spec}(m_1) * \triangleright P \vdash \mathsf{rwp}\ e\ \{\Phi\}}$$

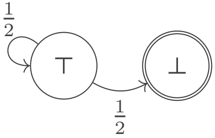
For probabilistic transitions, Caliper satisfies a range of **coupling rules** in the style of probabilistic relational Hoare logic (pRHL), *e.g.*,

$$\frac{\begin{array}{c} m_\bot \neq m_\top \\ m \rightarrow^{\frac{1}{2}} m_\bot \qquad \mathsf{spec}(m_\bot) * P \vdash \mathsf{rwp}\ K[\mathsf{false}]\ \{\Phi\} \\ m \rightarrow^{\frac{1}{2}} m_\top \qquad \mathsf{spec}(m_\top) * P \vdash \mathsf{rwp}\ K[\mathsf{true}]\ \{\Phi\} \end{array}}{\mathsf{spec}(m) * \triangleright P \vdash \mathsf{rwp}\ K[\,\mathsf{flip}\,]\ \{\Phi\}}$$

Example

```
let r = ref true in
 while !r do
   r ← flip
 end
```

$\preceq$



Example

Goal

$$\text{rwp}\left(\begin{array}{l}\text{let } r = \text{ref true in}\\ \text{while } !\, r \text{ do}\\ \quad r \leftarrow \text{flip}\\ \text{end}\end{array}\right)\{\_.\, \text{spec}(\bot)\}$$

Assumptions

$\text{spec}(\top)$

Example

Goal

$$\mathsf{rwp}\begin{pmatrix}\mathsf{while}\,!\,\ell\;\mathsf{do}\\ \ell \leftarrow \mathsf{flip}\\ \mathsf{end}\end{pmatrix}\{\_.\,\mathsf{spec}(\bot)\}$$

Assumptions

$\mathsf{spec}(\top)$

$\ell \mapsto \mathsf{true}$

Example

Goal

$$\mathsf{rwp}\left(\begin{array}{l} \mathsf{while}\ !\ell\ \mathsf{do} \\ \quad \ell \leftarrow \mathsf{flip} \\ \mathsf{end} \end{array}\right)\{\_.\ \mathsf{spec}(\bot)\}$$

Assumptions

$\mathsf{spec}(\top)$

$\ell \mapsto \mathsf{true}$

$$\triangleright\left(\begin{array}{c} \mathsf{spec}(\top) * \ell \mapsto \mathsf{true} \mathrel{-\!\!*} \\ \mathsf{rwp}\ \ldots\ \{\ldots\} \end{array}\right)$$

Example

Goal

$$\mathsf{rwp}\left(\begin{array}{l} \mathsf{if}\ !\ell\ \mathsf{then} \\ \quad \ell \leftarrow \mathsf{flip}; \\ \quad \mathsf{while}\ !\ell\ \mathsf{do} \\ \qquad \ell \leftarrow \mathsf{flip} \\ \quad \mathsf{end} \end{array}\right)\{\_.\ \mathsf{spec}(\bot)\}$$

Assumptions

$\mathsf{spec}(\top)$

$\ell \mapsto \mathsf{true}$

$$\triangleright\left(\begin{array}{l} \mathsf{spec}(\top) * \ell \mapsto \mathsf{true} \mathrel{-\!\!*} \\ \qquad \mathsf{rwp}\ \ldots\ \{\ldots\} \end{array}\right)$$

Example

Goal

$$\mathsf{rwp}\begin{pmatrix}\ell \leftarrow \mathsf{flip}; \\ \mathsf{while}\,!\,\ell\ \mathsf{do} \\ \quad \ell \leftarrow \mathsf{flip} \\ \mathsf{end}\end{pmatrix}\{\_.\ \mathsf{spec}(\bot)\}$$

Assumptions

$\mathsf{spec}(\top)$

$\ell \mapsto \mathsf{true}$

$$\triangleright\begin{pmatrix}\mathsf{spec}(\top) * \ell \mapsto \mathsf{true} \mathrel{-\!\!*} \\ \mathsf{rwp}\ \ldots\ \{\ldots\}\end{pmatrix}$$

Example

Goal

$$\mathsf{rwp}\left(\begin{array}{l}\ell \leftarrow \mathsf{flip};\\ \mathsf{while}\ !\,\ell\ \mathsf{do}\\ \quad \ell \leftarrow \mathsf{flip}\\ \mathsf{end}\end{array}\right)\ \{\_.\ \mathsf{spec}(\bot)\}$$

$$\boxed{\dfrac{\begin{array}{cc} & m_{\bot} \neq m_{\top} \\ m \to^{\frac{1}{2}} m_{\bot} & \mathsf{spec}(m_{\bot}) * P \vdash \mathsf{rwp}\ K[\mathsf{false}]\ \{\Phi\} \\ m \to^{\frac{1}{2}} m_{\top} & \mathsf{spec}(m_{\top}) * P \vdash \mathsf{rwp}\ K[\mathsf{true}]\ \{\Phi\}\end{array}}{\mathsf{spec}(m) * \triangleright P \vdash \mathsf{rwp}\ K[\ \mathsf{flip}\ ]\ \{\Phi\}}}$$

Assumptions

$\mathsf{spec}(\top)$

$\ell \mapsto \mathsf{true}$

$$\triangleright\left(\begin{array}{c}\mathsf{spec}(\top) * \ell \mapsto \mathsf{true} \mathrel{-\!\!*} \\ \mathsf{rwp}\ \ldots\ \{\ldots\}\end{array}\right)$$

Example

Goal

$$\text{rwp}\begin{pmatrix}\ell \leftarrow b; \\ \text{while}\ !\,\ell\ \text{do} \\ \quad \ell \leftarrow \text{flip} \\ \text{end}\end{pmatrix} \{\_.\ \text{spec}(\bot)\}$$

$$\boxed{\dfrac{\begin{array}{cc} \multicolumn{2}{c}{m_\bot \neq m_\top} \\ m \xrightarrow{}^{\frac{1}{2}} m_\bot & \text{spec}(m_\bot) * P \vdash \text{rwp}\ K[\text{false}]\ \{\Phi\} \\ m \xrightarrow{}^{\frac{1}{2}} m_\top & \text{spec}(m_\top) * P \vdash \text{rwp}\ K[\text{true}]\ \{\Phi\} \end{array}}{\text{spec}(m) * \triangleright P \vdash \text{rwp}\ K[\ \text{flip}\ ]\ \{\Phi\}}}$$

Assumptions

$\text{spec}(\text{if}\ b\ \text{then}\ \top\ \text{else}\ \bot)$

$\ell \mapsto \text{true}$

$$\begin{pmatrix}\text{spec}(\top) * \ell \mapsto \text{true} \mathrel{-\!\!*} \\ \quad \text{rwp}\ \ldots\ \{\ldots\}\end{pmatrix}$$

Example

Goal

$$\mathsf{rwp}\begin{pmatrix}\mathsf{while}\,!\,\ell\ \mathsf{do} \\ \quad \ell \leftarrow \mathsf{flip} \\ \mathsf{end}\end{pmatrix}\{\_.\ \mathsf{spec}(\bot)\}$$

Assumptions

$$\mathsf{spec}(\mathsf{if}\ b\ \mathsf{then}\ \top\ \mathsf{else}\ \bot)$$

$$\ell \mapsto b$$

$$\begin{pmatrix}\mathsf{spec}(\top) * \ell \mapsto \mathsf{true} \mathrel{-\!\!*} \\ \mathsf{rwp}\ \ldots\ \{\ldots\}\end{pmatrix}$$

Takeaways

The approach taken in Caliper exploits three key ingredients:

- **Higher-order separation logic** for powerful modular reasoning
- **Guarded recursion** for termination-preserving refinement reasoning
- **Probabilistic couplings** for “aligning” probabilistic transitions

Well-tested abstractions that scale to reasoning about complex programs!

More in the paper

- Semantic model and soundness of the logic.
- More general and expressive coupling rules (uniform sampling), asynchronous couplings for flexible coupling-based reasoning.
- A series of case studies showcasing the approach and how it supports compositional separation-logic reasoning.
    - A higher-order list generator
    - Lazily-sampled reals
    - Treaps
    - A sampler for Galton-Watson trees

Summary

- **Caliper**, a separation logic for **termination-preserving refinement** between probabilistic programs and probabilistic transition systems.
- To preserve termination, Caliper exploits **guarded recursion** which seamlessly integrate with existing separation-logic reasoning principles.
- **Probabilistic couplings** for relational reasoning about probabilistic systems.
- Full mechanization in the Coq proof assistant using the Iris framework.

Thank you!

E-mail `s.gregersen@nyu.edu`

Presampling tapes

In our POPL'24 paper, we introduced **presampling tapes** to alleviate the asynchronous nature of relational reasoning about higher-order programs.

Key idea: a resource $\iota \hookrightarrow \vec{b}$ that "prophesizes" the outcome of future samplings.

Presampling tapes

In our POPL'24 paper, we introduced **presampling tapes** to alleviate the asynchronous nature of relational reasoning about higher-order programs.

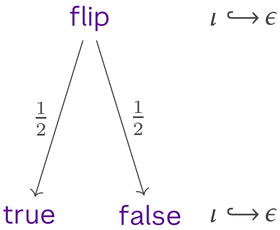
Key idea: a resource $\iota \hookrightarrow \vec{b}$ that "prophesizes" the outcome of future samplings.

flip $\qquad$ $\iota \hookrightarrow \epsilon$

Presampling tapes

In our POPL'24 paper, we introduced **presampling tapes** to alleviate the asynchronous nature of relational reasoning about higher-order programs.

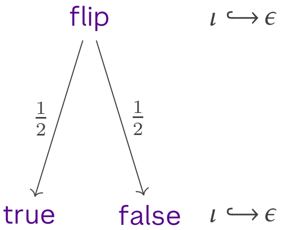
Key idea: a resource $\iota \hookrightarrow \vec{b}$ that "prophesizes" the outcome of future samplings.



Presampling tapes

In our POPL'24 paper, we introduced **presampling tapes** to alleviate the asynchronous nature of relational reasoning about higher-order programs.

Key idea: a resource $\iota \hookrightarrow \vec{b}$ that "prophesizes" the outcome of future samplings.



flip $\iota \hookrightarrow \epsilon$

$\frac{1}{2}$ $\frac{1}{2}$

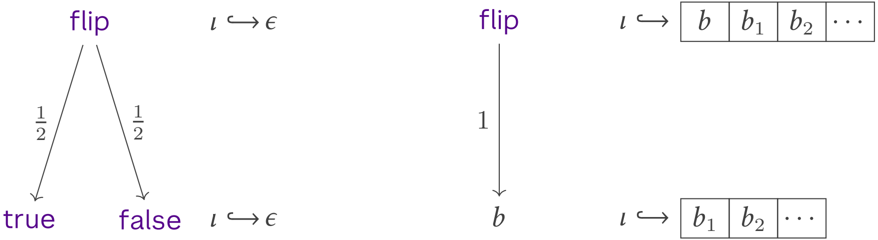
true false $\iota \hookrightarrow \epsilon$

flip $\iota \hookrightarrow \boxed{b}\boxed{b_1}\boxed{b_2}\boxed{\cdots}$

Presampling tapes

In our POPL'24 paper, we introduced **presampling tapes** to alleviate the asynchronous nature of relational reasoning about higher-order programs.

Key idea: a resource $\iota \hookrightarrow \vec{b}$ that "prophesizes" the outcome of future samplings.



flip $\iota \hookrightarrow \epsilon$

$\frac{1}{2}$ $\frac{1}{2}$

true false $\iota \hookrightarrow \epsilon$

flip $\iota \hookrightarrow \boxed{b}\boxed{b_1}\boxed{b_2}\boxed{\cdots}$

1

b $\iota \hookrightarrow \boxed{b_1}\boxed{b_2}\boxed{\cdots}$

Presampling tapes cont’d

Presampling, however, is just a **ghost operation**!

$$\frac{\begin{array}{cc} & m_{\bot} \neq m_{\top} \\ m \longrightarrow^{\frac{1}{2}} m_{\bot} & P * \mathsf{spec}(m_f) * \iota \hookrightarrow \vec{b} \cdot \mathsf{false} \vdash \mathsf{rwp}\ e\ \{\Phi\} \\ m \longrightarrow^{\frac{1}{2}} m_{\top} & P * \mathsf{spec}(m_t) * \iota \hookrightarrow \vec{b} \cdot \mathsf{true} \vdash \mathsf{rwp}\ e\ \{\Phi\} \end{array}}{\triangleright P * \iota \hookrightarrow \vec{b} * \mathsf{spec}(m) \vdash \mathsf{rwp}\ e\ \{\Phi\}}$$

Presampling tapes cont’d

Presampling, however, is just a **ghost operation**!

$$\frac{\begin{array}{cc} & m_\bot \neq m_\top \\ m \longrightarrow^{\frac{1}{2}} m_\bot & P * \mathsf{spec}(m_f) * \iota \hookrightarrow \vec{b} \cdot \mathsf{false} \vdash \mathsf{rwp}\ e\ \{\Phi\} \\ m \longrightarrow^{\frac{1}{2}} m_\top & P * \mathsf{spec}(m_t) * \iota \hookrightarrow \vec{b} \cdot \mathsf{true} \vdash \mathsf{rwp}\ e\ \{\Phi\} \end{array}}{\triangleright P * \iota \hookrightarrow \vec{b} * \mathsf{spec}(m) \vdash \mathsf{rwp}\ e\ \{\Phi\}}$$

Two immediate benefits that we exploit:

- Eliminating later modalities “asynchronously”
- Relating *one* model step to *multiple* (non-adjacent) samplings

$$\dfrac{\mathrm{step}(m_1) \lesssim \mathrm{unif}(N) : R \qquad \vdash \forall (m_2, n) \in R.\, (\mathsf{spec}(m_2) * P) \mathrel{-\!\!*} \mathsf{rwp}\ n\ \{\Phi\}}{\mathsf{spec}(m_1) * \triangleright P \vdash \mathsf{rwp}\ \mathsf{rand}\, N\ \{\Phi\}}$$

Lemma

If $\sum_{m' \in M} \mathit{exec}_n(m)(m') \leq r$ *for all* n *then* $\mathit{exec}_{\Downarrow}(m) \leq r$.

Definition (Left-partial coupling)

Let $\mu_1 \in \mathcal{D}(A)$ and $\mu_2 \in \mathcal{D}(B)$. A sub-distribution $\mu \in \mathcal{D}(A \times B)$ is a *left-partial coupling* of μ_1 and μ_2 if

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

We write $\mu_1 \lesssim \mu_2$ if there exists a left-partial coupling of μ_1 and μ_2. Given a relation $R \subseteq A \times B$ we say μ is a left-partial R-coupling if furthermore $\mathrm{supp}(\mu) \subseteq R$. We write $\mu_1 \lesssim \mu_2 : R$ if there exists a left-partial R-coupling of μ_1 and μ_2.

Lemma

If $\mu_1 \lesssim \mu_2$ *then* $\sum_{a \in A} \mu_1(a) \leq \sum_{b \in B} \mu_2(b)$.

$$\left(\forall f, v'.\ \left\{\forall v''.\ \triangleright\left(\{\mathit{\Phi}(v'')\}\ f\ v''\ \{\Psi\}\right)\right\}\ F\ f\ v'\ \{\Psi\} * \mathit{\Phi}(v')\right) \vdash \{\mathit{\Phi}(v)\}\ \mathsf{fix}\ F\ v\ \{\Psi\}$$