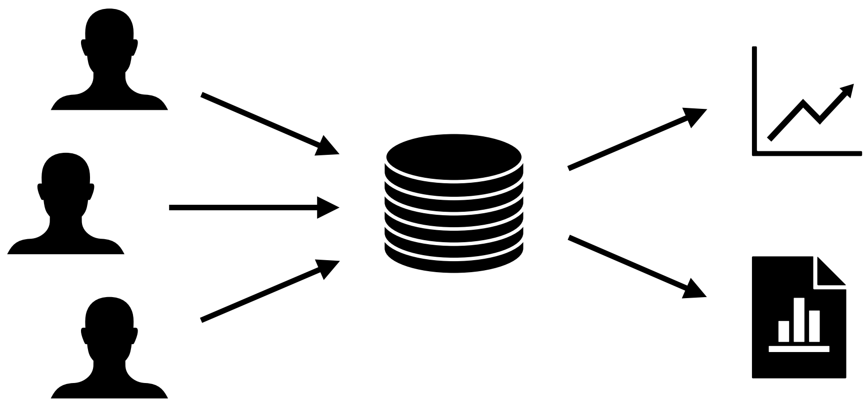
Verifying Probabilistic Programs

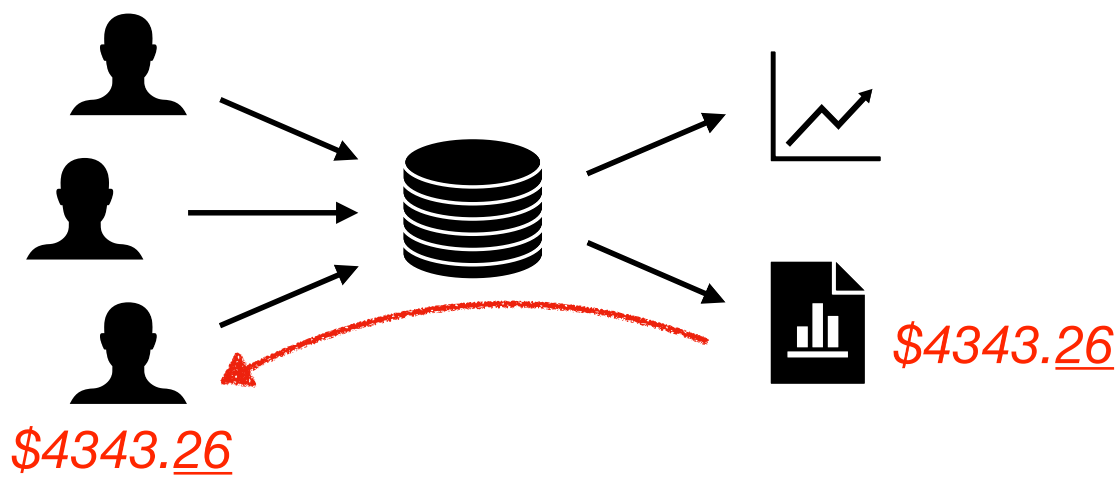
Markus de Medeiros

Probabilistic Programs



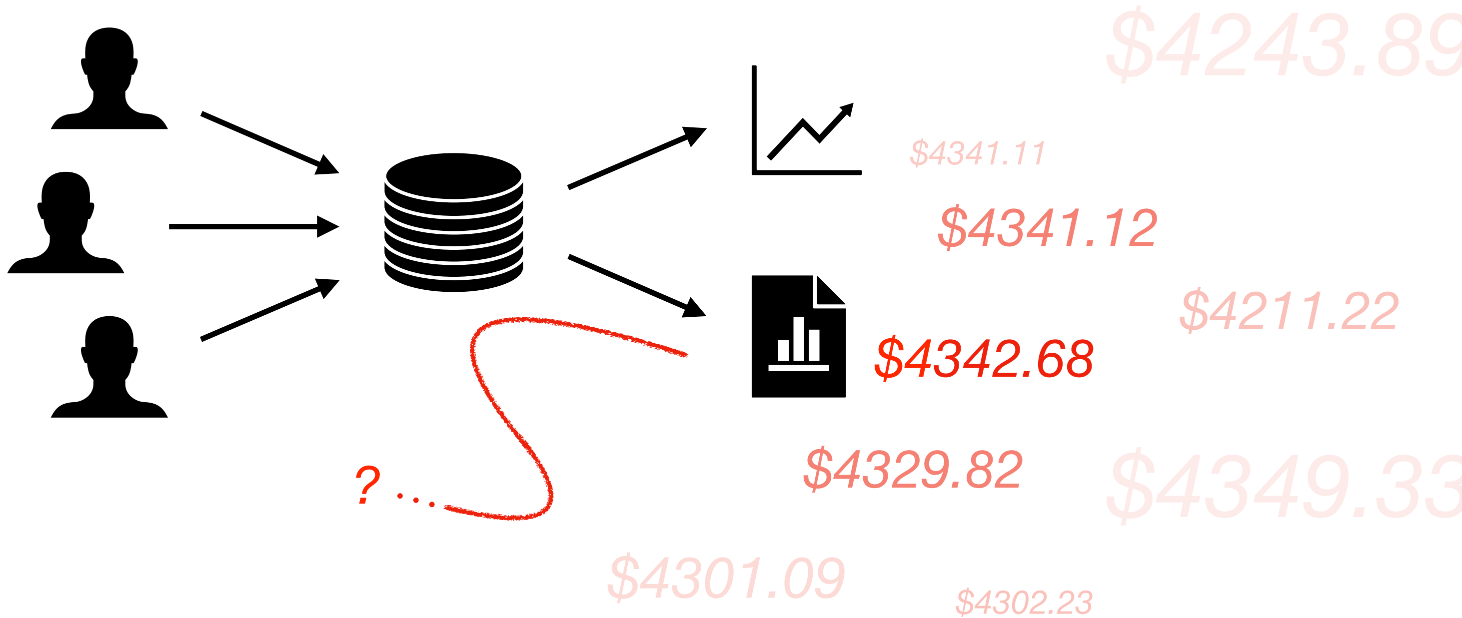
Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Probabilistic Programs



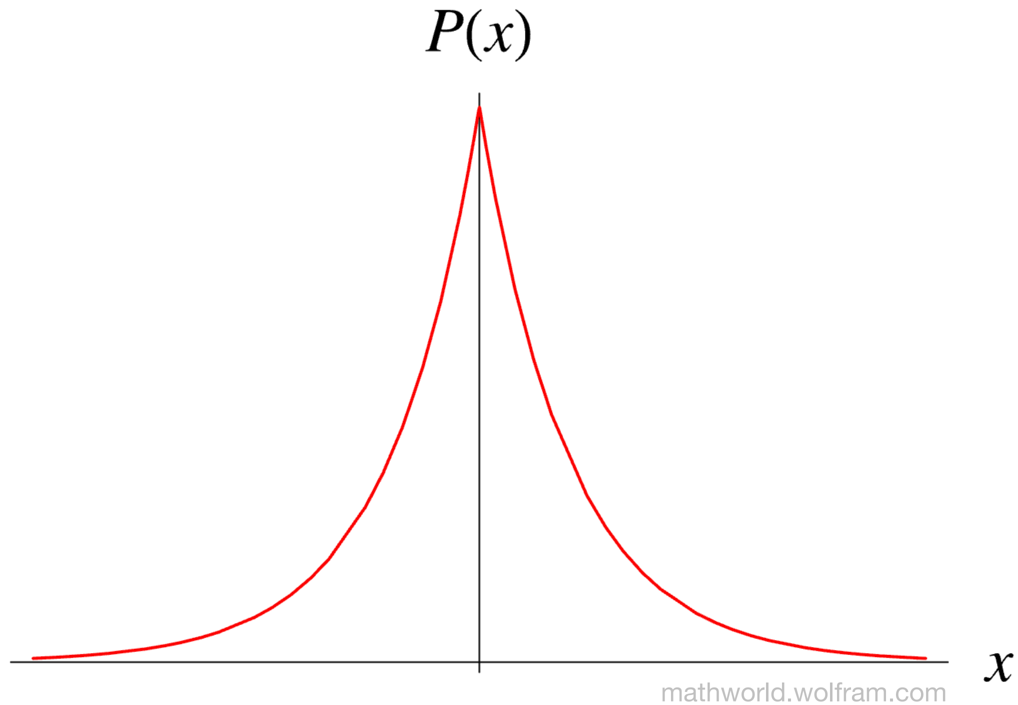
Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Probabilistic Programs



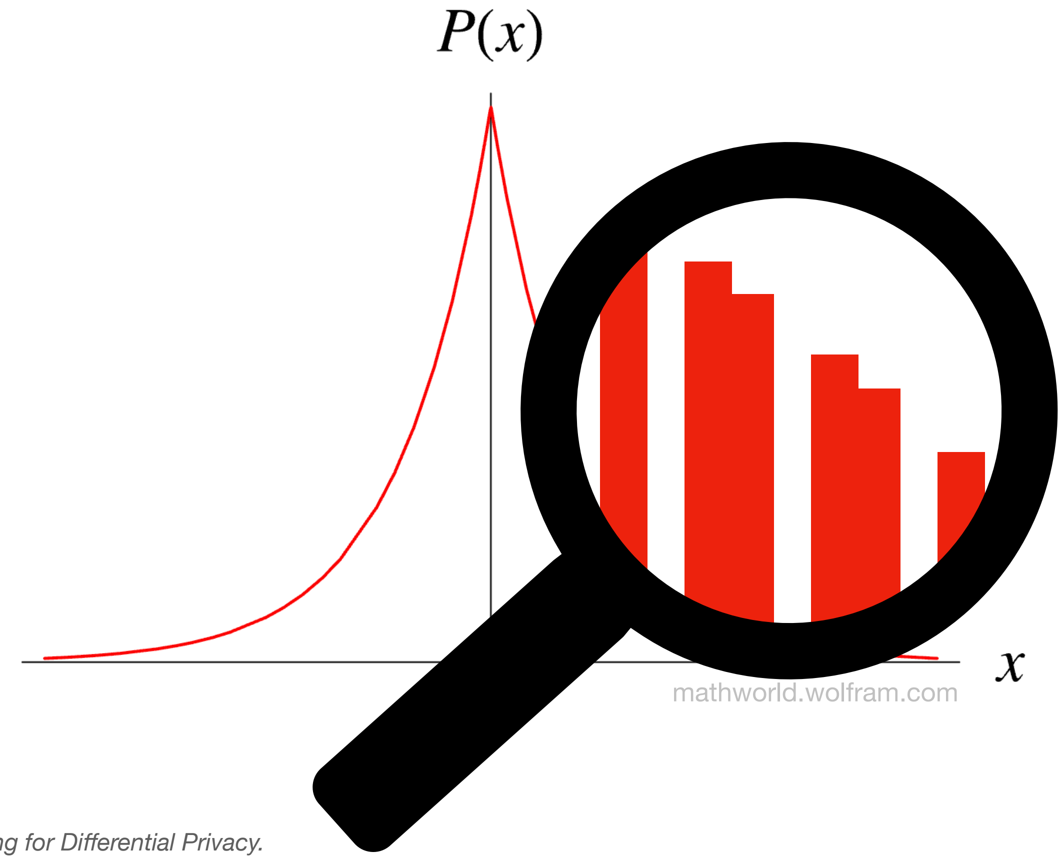
Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Probabilistic Programs



Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

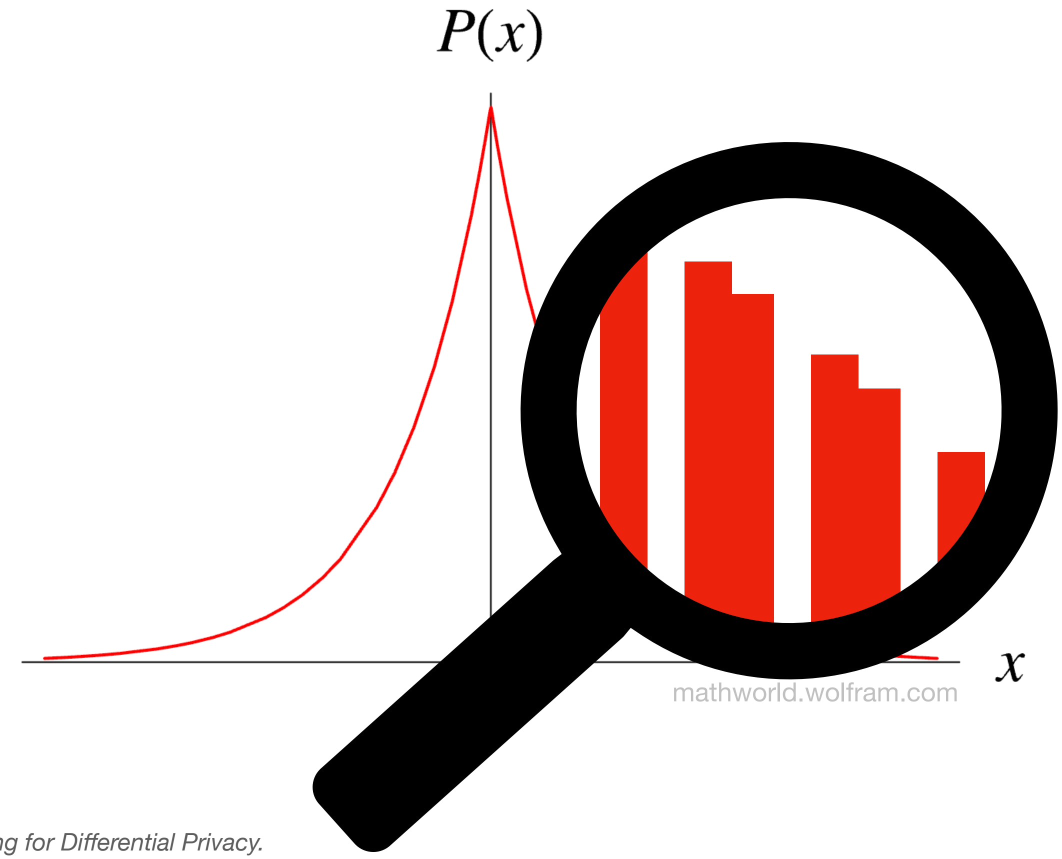
Probabilistic Programs



Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Probabilistic Programs

- Hard to test
- High-Impact
- Quantitative Correctness



Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Verifying Probabilistic Programs

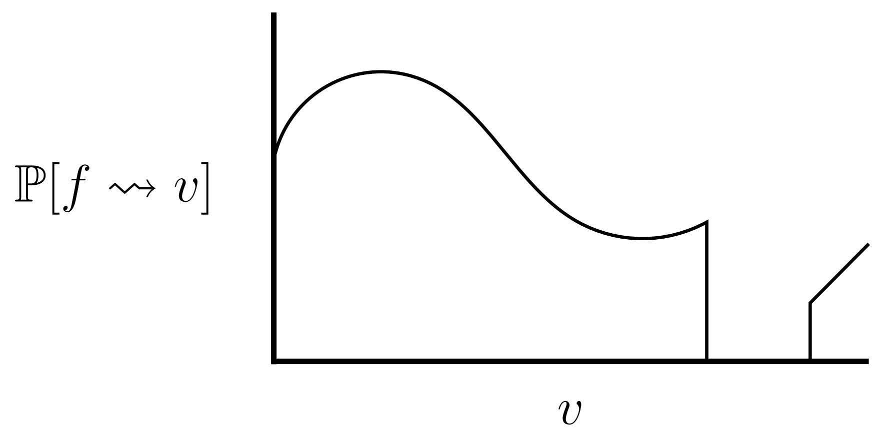
- Eris
- Total Eris
- *Tachis* *(Skipped)*
- SampCert
- Continuous Eris *(Ongoing)*

Section 1.

Eris

Probabilistic Programs

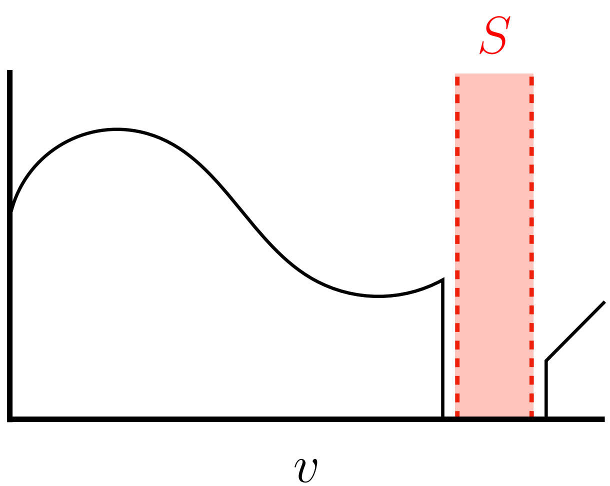
Executing a probabilistic program produces a subdistribution over states



Probabilistic Programs

Executing a probabilistic program produces a subdistribution over states

$\mathbb{P}[f \rightsquigarrow v]$



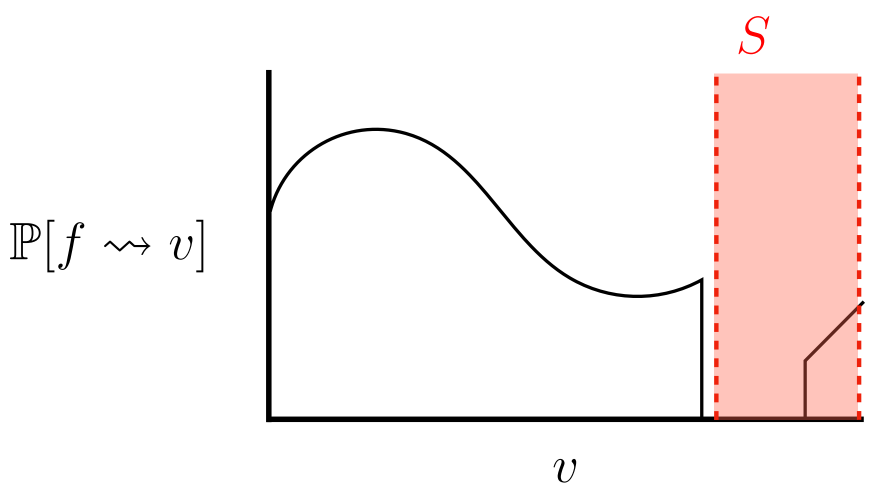
$$C(v) = \begin{cases} v \in S & 1 \\ v \notin S & 0 \end{cases}$$

Safety: $\mathbb{E}[C] = 0$

Quantitative bounds ⇒ properties of the program

Probabilistic Programs

Executing a probabilistic program produces a subdistribution over states



$$C(v) = \begin{cases} v \in S & 1 \\ v \notin S & 0 \end{cases}$$

Safety: $\mathbb{E}[C] = 0$

Quantitative bounds ⇒ properties of the program

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Barthe, Gaboardi, Grégoire, Hsu, and Strub. *A program logic for union bounds.*

Approximate Specifications

aHL

$$\{x \neq y\}\ \textsf{collide}\ x\ y\ \{b.\, b = \textsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\textsf{True}\}\ \textsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Barthe, Gaboardi, Grégoire, Hsu, and Strub. *A program logic for union bounds.*

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Barthe, Gaboardi, Grégoire, Hsu, and Strub. *A program logic for union bounds.*

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Barthe, Gaboardi, Grégoire, Hsu, and Strub. *A program logic for union bounds.*

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 1

$$\frac{\forall a.\, \{\ldots\}\ f\ a\ \{\ldots\}_{\epsilon(a)}}{\{\ldots\}\ \mathsf{map}\ f\ L\ \{\ldots\}_{\sum_{a \in L} \epsilon(a)}}$$

error specifications propagate

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_{0}$$
$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\mathsf{test}\ d = \ \mathsf{if\ decide}\ d\ \mathsf{then}\ (\mathsf{true}, G\ d)\ \mathsf{else}\ (\mathsf{false}, F\ d)$$

$$\{\top\}\ \mathsf{test}\ d\ \{(v, d).\, P\}_{?}$$

error depends on return value

Error Credits

Eris

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

$$\{\lightning(2^{-64}) * x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}$$

Expected Error Bounds as a Resource

Error Credits

Eris

Expected Error Bounds as a Resource

$$\boxed{\vdash \{ \lightning(\epsilon) \} \; f \; \{ v.\, P \}}$$

If f terminates with value v,
$P\,v$ holds with probability $1 - \epsilon$.

Step-indexed & higher-order
Mechanized in Rocq

Error Credits

Eris

Expected Error Bounds as a Resource

$$\boxed{\vdash \{\lightning(\epsilon)\}\ f\ \{v.\, P\}}$$

$$\frac{\{P\}\ f\ \{Q\}}{\{P * \lightning(\epsilon)\}\ f\ \{Q * \lightning(\epsilon)\}} \qquad (\triangleright P \Rightarrow P) \vdash P$$

$$\Big\{\ \{P * \lightning(\epsilon)\}\ f\ \{Q\}\ \Big\}\ g\ \{R\}$$

Iris

Step-indexed & higher-order

Mechanized in Rocq

The Eris Logic

Limitation 1

The Eris Logic

Limitation 1

Standard higher-order specification:

$$\frac{\forall a,\ \{P\ a\}\ f\ a\ \{Q\ a\}}{\left\{ \mathop{\Huge *}_{a \in L} (P\ a) \right\} \textsf{map}\ f\ L \left\{ L'.\ \mathop{\Huge *}_{a \in L'} (Q\ a) \right\}}$$

The Eris Logic

Limitation 1

Standard higher-order specification:

$$\frac{\forall a,\ \{P\ a\}\ f\ a\ \{Q\ a\}}{\left\{ \mathop{\Huge\ast}_{a \in L} (P\ a) \right\} \textsf{map}\ f\ L \left\{ L'.\ \mathop{\Huge\ast}_{a \in L'} (Q\ a) \right\}}$$

Derived error-aware specification:

$$\frac{\forall y,\ \{↯(2^{-64})\}\ \textsf{hash}\ y\ \{v.\, v \neq v'\}}{\left\{ \mathop{\Huge\ast}_{a \in L} ↯(2^{-64}) \right\} \textsf{map hash}\ L \left\{ L'.\ \mathop{\Huge\ast}_{a \in L'} a \neq v' \right\}}$$

The Eris Logic

Limitation 2

$$\{\top\}\, G\ d\, \{d.\, P\}_{0}$$
$$\{\top\}\, F\ d\, \{d.\, P\}_{1/100}$$

$$\begin{aligned}\mathsf{test}\ d = \ & \mathsf{if}\ \mathsf{decide}\ d \\ & \mathsf{then}\ (\mathsf{true}, G\ d) \\ & \mathsf{else}\ (\mathsf{false}, F\ d)\end{aligned}$$

$$\{\top\}\, \mathsf{test}\ d\, \{(v, d).\, P\}_{?}$$

The Eris Logic

Limitation 2

$$\{\top\}\; G\ d\; \{d.\, P\}$$
$$\{\lightning(1/100)\}\; F\ d\; \{d.\, P\}$$

$$\mathsf{test}\ d = \ \mathsf{if\ decide}\ d \ \mathsf{then}\ (\mathsf{true}, G\ d) \ \mathsf{else}\ (\mathsf{false}, F\ d)$$

State-dependent specification:

$$\left\{\ \lightning(1/100)\ \right\} \mathsf{test}\ d \left\{ (v, d).\, P * \begin{pmatrix} \mathsf{if}\ v \\ \ \mathsf{then}\ \lightning(1/100) \\ \ \mathsf{else}\ \top \end{pmatrix} \right\}$$

Error Credits

Core Rules

Error Credits

Core Rules

Spending $\lightning(1) \vdash \bot$

Error Credits

Core Rules

Spending $\lightning(1) \vdash \bot$

Splitting $\lightning(\epsilon_1 + \epsilon_2) \dashv\vdash \lightning(\epsilon_1) * \lightning(\epsilon_2)$

Error Credits

Core Rules

Spending $\lightning(1) \vdash \bot$

Splitting $\lightning(\epsilon_1 + \epsilon_2) \dashv\vdash \lightning(\epsilon_1) * \lightning(\epsilon_2)$

Averaging

$$\frac{\mathbb{E}_{x \sim D}[\epsilon_x] = \overline{\epsilon}}{\{\lightning(\overline{\epsilon})\}\ \mathsf{sample}(D)\ \{x.\, \lightning(\epsilon_x)\}}$$

$\lightning(\overline{\epsilon})$

$f(\mathsf{sample}(5))$

$\lightning(\epsilon_0)$ $\lightning(\epsilon_1)$ $\lightning(\epsilon_2)$ $\lightning(\epsilon_3)$ $\lightning(\epsilon_4)$

$f(0)$ $f(1)$ $f(2)$ $f(3)$ $f(4)$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Error Credits

Derived Rules

$$\{\lightning(\epsilon_1) * P\}\ e_1\ \{Q\}$$

$$\{\lightning(\epsilon_2) * Q\}\ e_2\ \{R\}$$

aHL Union Bound

$$\frac{\{P\}\ e_1\ \{Q\}_{\epsilon_1} \qquad \{Q\}\ e_2\ \{R\}_{\epsilon_2}}{\{P\}\ e_1; e_2\ \{R\}_{\epsilon_1 + \epsilon_2}}$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$\lightning(\epsilon_1 + \epsilon_2) * P$$

$$e_1;\ e_2$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$\lightning(\epsilon_1) * \lightning(\epsilon_2) * P$$

$$e_1;\ e_2$$

Splitting

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$↯(\epsilon_1) * ↯(\epsilon_2) * P$$

Splitting

$$\{↯(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$e_1;\ e_2$$

Frame Rule

$$\{↯(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$↯(\epsilon_2) * Q$$

$$e_2$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\lightning(\epsilon_1) * \lightning(\epsilon_2) * P$$

Splitting

$$e_1;\ e_2$$

Frame Rule

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\} \qquad \downarrow$$

$$\lightning(\epsilon_2) * Q$$

$$e_2$$

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\} \qquad \downarrow$$

$$R$$

Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

$$\lightning(1/5)$$

$$f(\mathsf{sample}(5))$$

Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Averaging

$$\begin{tikzcd}[column sep=small]
 & & \overset{\text{↯}(1/5)}{f(\mathsf{sample}(5))} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\overset{\text{↯}(0)}{f(0)} & \overset{\text{↯}(0)}{f(1)} & \overset{\text{↯}(0)}{f(2)} & \overset{\text{↯}(1)}{f(3)} & \overset{\text{↯}(0)}{f(4)}
\end{tikzcd}$$

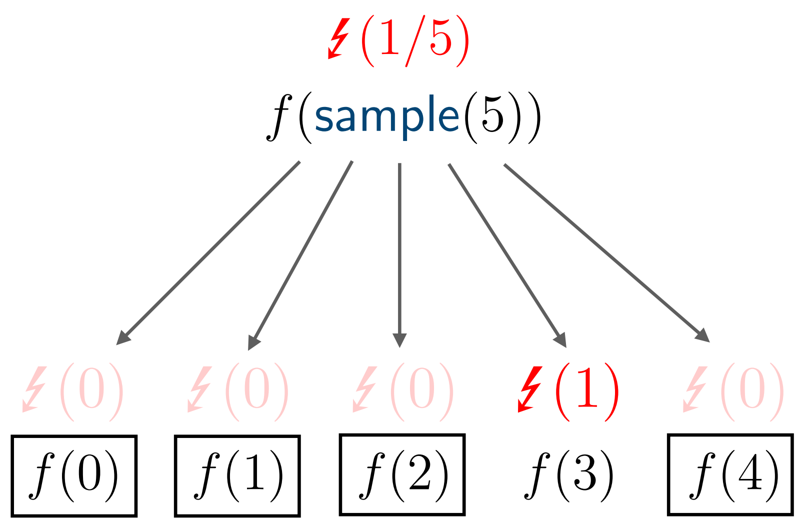
Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Averaging

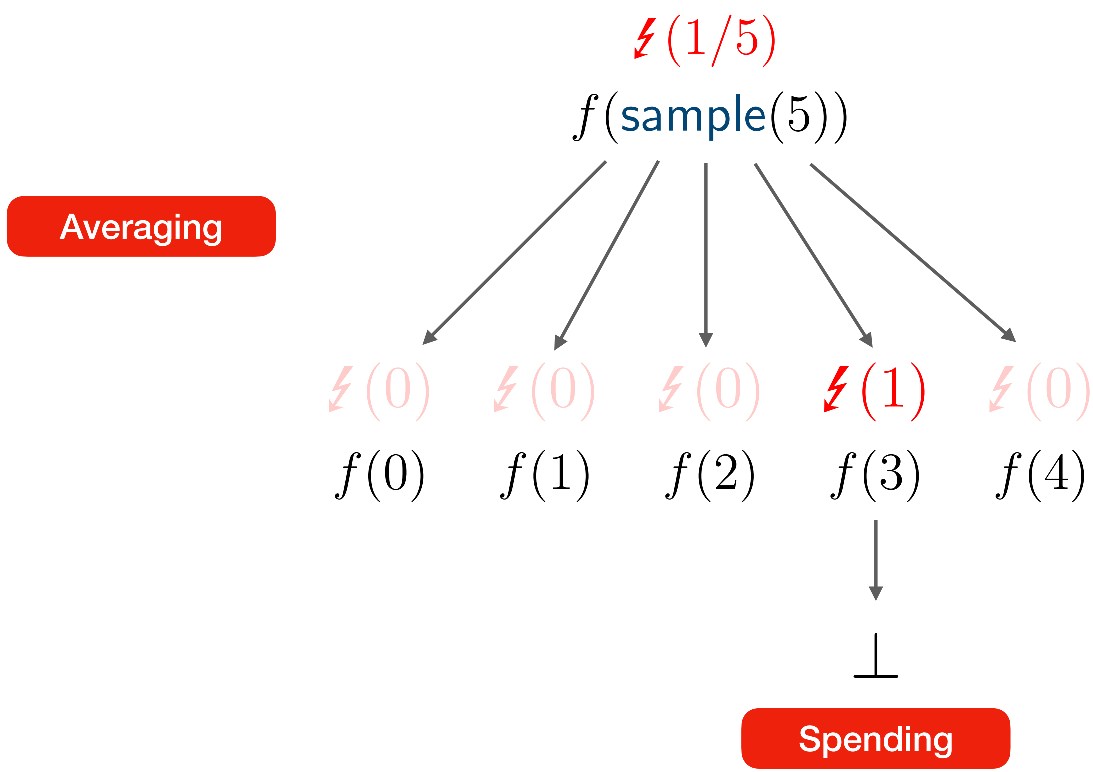


Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x\sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$



Eris

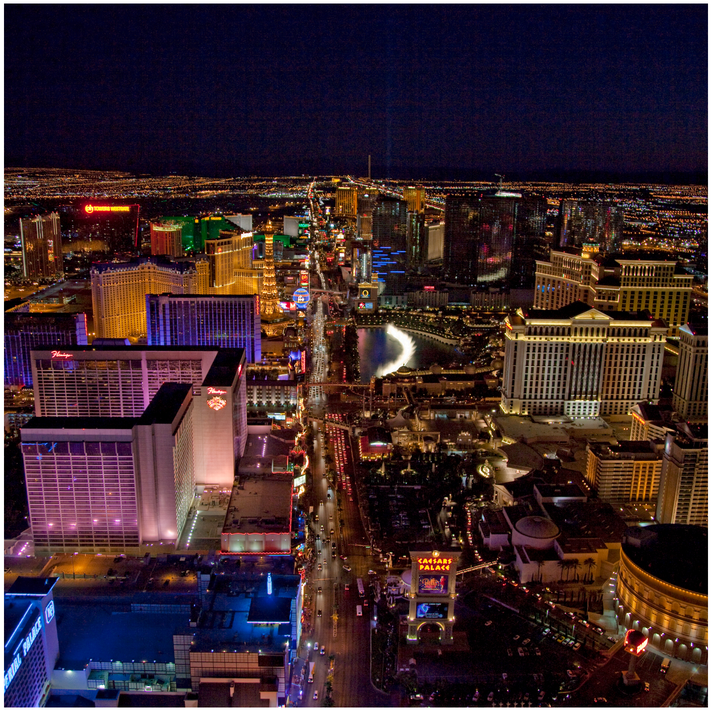
- Expected error bounds as a separation logic resource
- Modular proofs of approximate correctness
- Derived aHL rules, amortized reasoning

Section 2.

Total Eris

Monte Carlo

- *Always terminates*
- *May be incorrect*



Las Vegas

- *May not terminate*
- *Always correct*

Total Error Credits

Total Eris

Termination Bounds as a Resource

$$\boxed{\vdash [\lightning(\epsilon)]\ f\ [v.\, P]}$$

f terminates with value v <u>and</u>

$P\,v$ holds, with probability $1 - \epsilon$.

Step-indexed & **higher-order**

Mechanized in Rocq

Eris

$\vdash \{ \lightning(\epsilon) \}\ f\ \{P\}$

Total Eris

$\vdash [\lightning(\epsilon)]\ f\ [P]$

Iris **Step-indexed & higher-order**

Error Credits with Spending Splitting Averaging

Eris

$$\vdash \{↯(\epsilon)\}\ f\ \{P\}$$

Total Eris

$$\vdash [↯(\epsilon)]\ f\ [P]$$

Iris Step-indexed & higher-order

Error Credits with Spending Splitting Averaging

Recursion rule:

To prove

$$\vdash \{P\}\ (\mathsf{rec}\ f\ x\ =\ e)\ v\ \{Q\}$$

assume

$$\forall w. \{P\}\ (\mathsf{rec}\ f\ x\ =\ e)\ w\ \{Q\}$$

and show

$$\vdash \{P\}\ e[v/x][(\mathsf{rec}\ f\ x\ =\ e)/f]\ \{Q\}$$

Eris

$$\vdash \{\lightning(\epsilon)\}\ f\ \{P\}$$

Total Eris

$$\vdash [\lightning(\epsilon)]\ f\ [P]$$

Iris **Step-indexed & higher-order**

Error Credits with Spending Splitting Averaging

Recursion rule:

To prove

$$\vdash \{P\}\ (\mathsf{rec}\ f\ x\ =\ e)\ v\ \{Q\}$$

assume

$$\forall w. \{P\}\ (\mathsf{rec}\ f\ x\ =\ e)\ w\ \{Q\}$$

and show

$$\vdash \{P\}\ e[v/x][(\mathsf{rec}\ f\ x\ =\ e)/f]\ \{Q\}$$

Recursion rule does not hold!

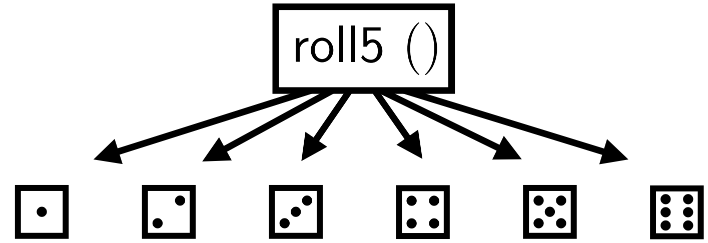
Error Induction

Rejection Sampling

$$
\begin{array}{l}
\mathsf{rec}\ \mathsf{roll5}\ \_ = \\
\quad \mathsf{let}\ \mathsf{roll} = 1 + \mathsf{sample}\ 6\ \mathsf{in} \\
\quad \mathsf{if}\ (\mathsf{roll} < 6) \\
\quad\quad \mathsf{then}\ \mathsf{roll} \\
\quad\quad \mathsf{else}\ \mathsf{roll5}\ ()
\end{array}
$$

Error Induction

Rejection Sampling



$\text{rec roll5 } \_ =$

$\quad \text{let roll} = 1 + \text{sample } 6 \text{ in}$

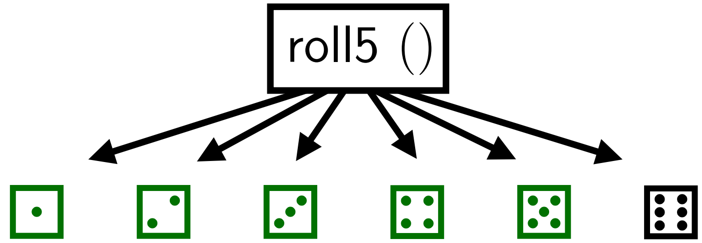
$\quad \text{if } (\text{roll} < 6)$

$\quad\quad \text{then roll}$

$\quad\quad \text{else roll5 } ()$

Error Induction

Rejection Sampling




```
rec roll5 _ =
  let roll = 1 + sample 6 in
  if (roll < 6)
    then roll
    else roll5 ()
```

Error Induction

Rejection Sampling

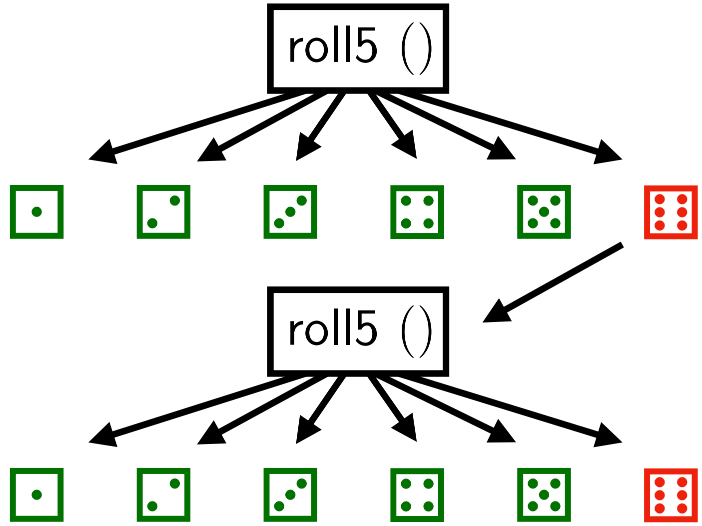
$\text{rec roll5 } \_ =$

$\quad \text{let roll} = 1 + \text{sample } 6 \text{ in}$

$\quad \text{if } (\text{roll} < 6)$

$\quad\quad \text{then roll}$

$\quad\quad \text{else roll5 } ()$



Error Induction

Rejection Sampling

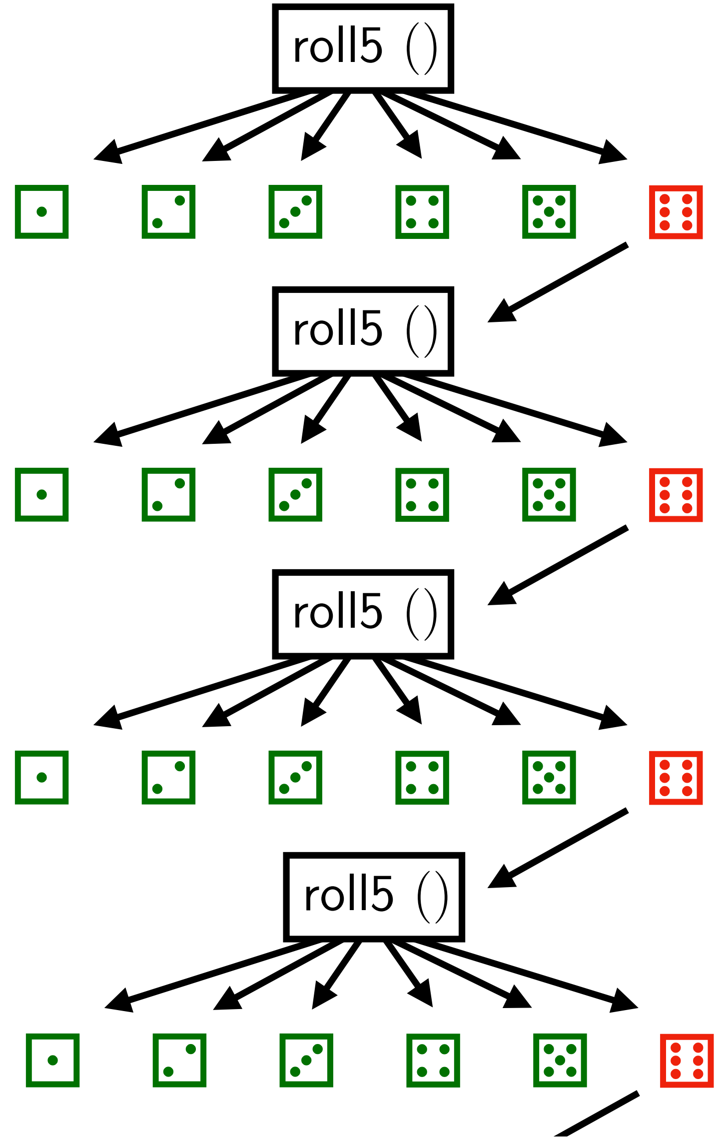
$\textsf{rec roll5}\ \_ =$

$\quad \textsf{let roll} = 1 + \textsf{sample}\, 6\ \textsf{in}$

$\quad \textsf{if}\ (\textsf{roll} < 6)$

$\qquad \textsf{then roll}$

$\qquad \textsf{else roll5}\ ()$

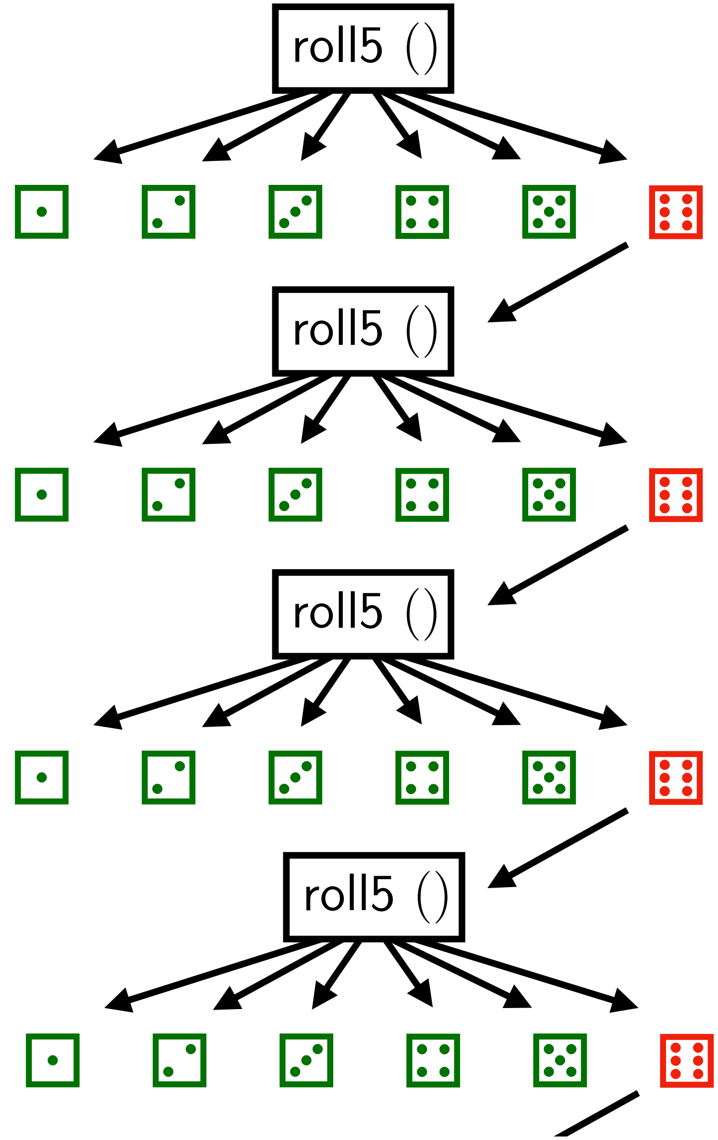


Error Induction

Rejection Sampling

$\mathsf{rec}\ \mathsf{roll5}\ \_ =$
$\quad \mathsf{let}\ \mathsf{roll} = 1 + \mathsf{sample}\ 6\ \mathsf{in}$
$\quad \mathsf{if}\ (\mathsf{roll} < 6)$
$\quad\quad \mathsf{then}\ \mathsf{roll}$
$\quad\quad \mathsf{else}\ \mathsf{roll5}\ ()$

Prove $[\top]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$ **?**



$\boxed{\text{roll5}\ ()}$ $\boxed{\lightning(\epsilon)}$

Error Induction

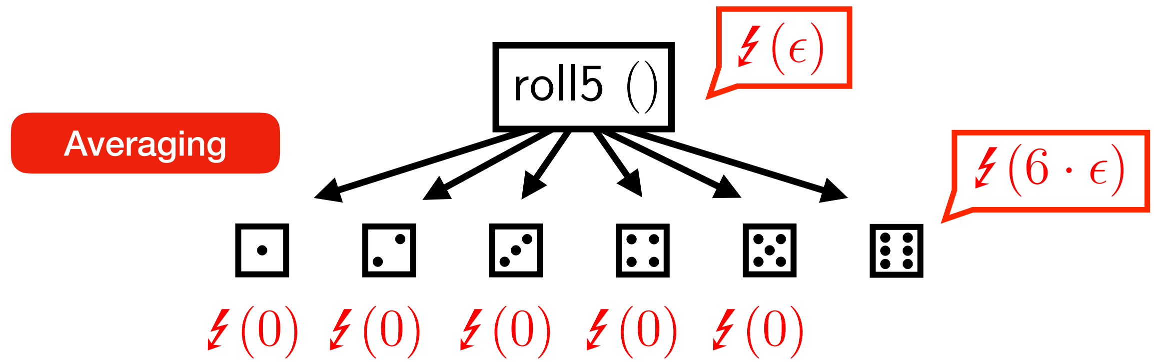
Rejection Sampling

Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

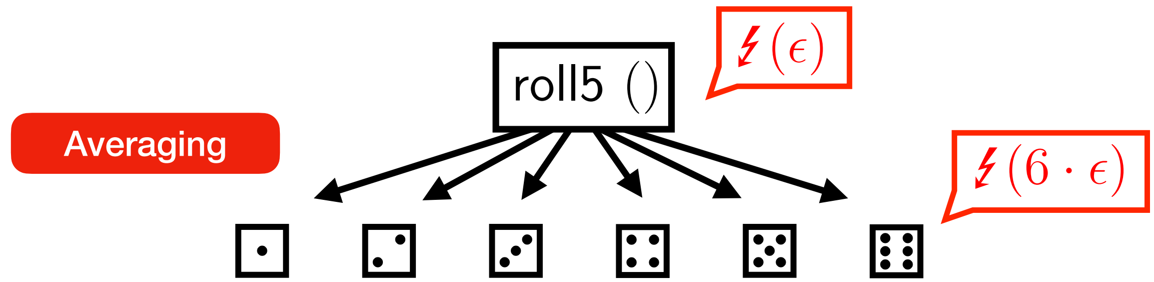


Prove that for all $0 < \epsilon$

$$[↯(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

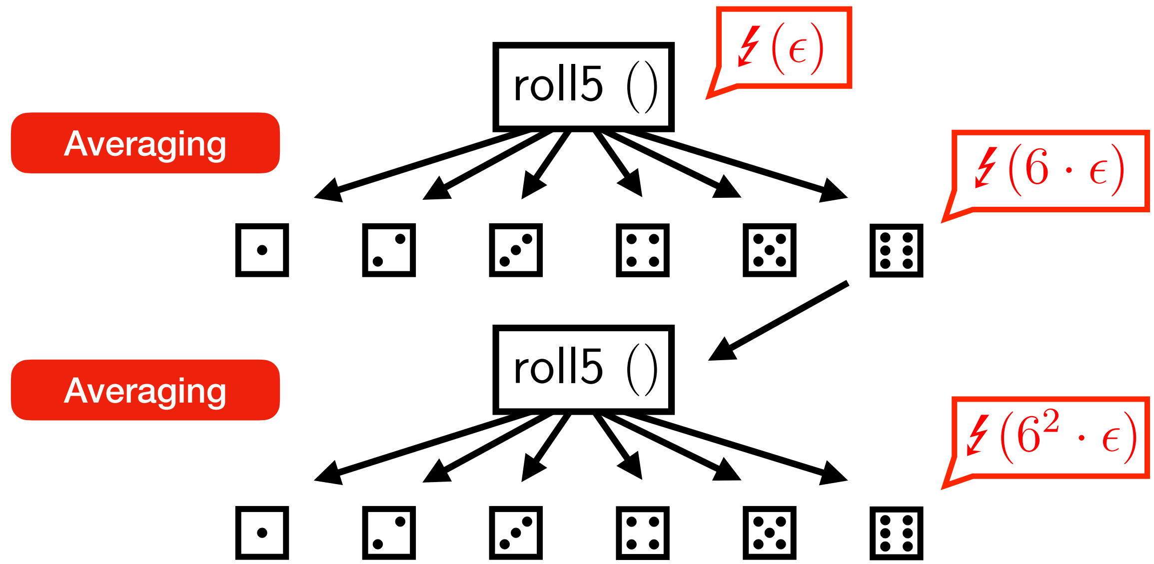


Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling



Prove that for all $0 < \epsilon$

$$[\text{↯}(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

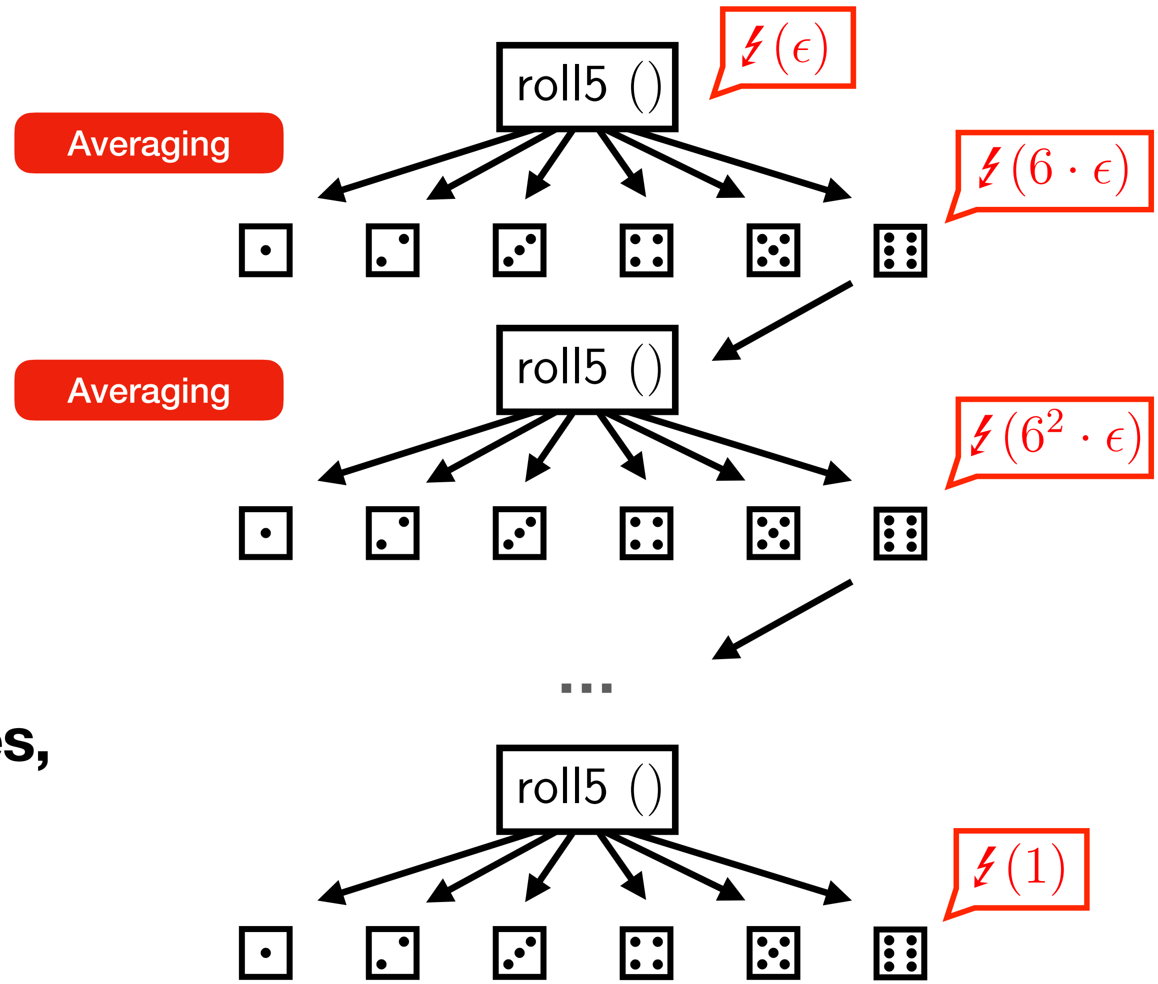
Error Induction

Rejection Sampling

Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Apply Averaging $\log_6(1/\epsilon)$ **times,**



Error Induction

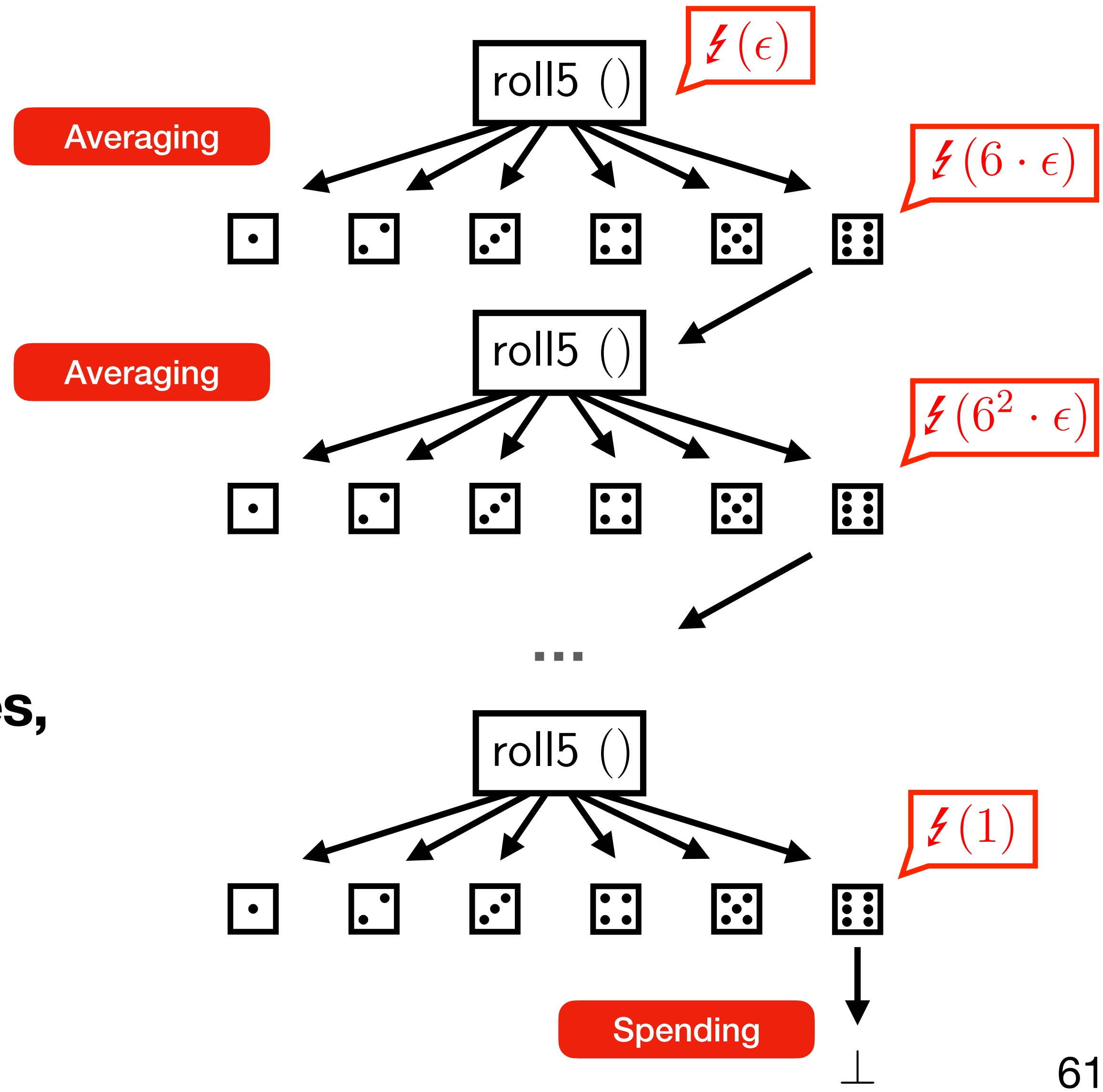
Rejection Sampling

Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Apply Averaging $\log_6(1/\epsilon)$ **times,**

Apply Spending **once.**



Error Induction

Rejection Sampling

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

Total Eris $\vdash [\lightning(\epsilon)]\; f\; [P]$

f terminates with value v and $P\,v$ holds, with probability $1 - \epsilon$

"roll5 terminates with a value less than 6 with arbitrarily high probability"

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\; \mathsf{roll5}\; ()\; [v.\, v < 6]$$

Error Induction

Rejection Sampling

Total Eris $\vdash [\lightning(\epsilon)]\ f\ [P]$

f terminates with value v and $P\,v$ holds, with probability $1 - \epsilon$

"roll5 terminates with a value less than 6 with arbitrarily high probability"

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Continuity

$$\vdash [\top]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

"roll5 terminates with a value less than 6 with probability 1"

Error Induction

*Assume a **nonzero amount of credit**,*

*Prove that the **error increases** in every recursive case,*

*Perform induction on the **number of rounds**,*

*Conclude by **continuity**.*

Error Amplification

Symbolic Execution

Credit Reasoning

Error Amplification

→

Deterministic

Symbolic Execution

Presampling Tapes

Clutch

- Future random events as ghost state

Label

Bound

Presamples

$$\iota \hookrightarrow (N;\ [\ell_1, \ell_2, \ldots])$$

Oddershede Gregersen, Aguirre, Haselwarter, Tassarotti, and Birkedal. *Asynchronous Probabilistic Couplings in Higher-Order Separation Logic.*

Presampling Tapes

Clutch

Label

Bound

Presamples

$$\iota \hookrightarrow (N;\ [\ell_1, \ell_2, \ldots])$$

- **Future random events as ghost state**

Starting with…	*Execute…*	*To get…*
$\iota \hookrightarrow (N;\ [\ell_1, \ell_2, \ldots])$	$\mathsf{rand}_\iota(N) \rightsquigarrow \ell_1$	$\iota \hookrightarrow (N;\ [\ell_2, \ldots])$
$\iota \hookrightarrow (N;\ [])$	$\mathsf{rand}_\iota(N) \rightsquigarrow_{1/N} n$	$\iota \hookrightarrow (N;\ [])$
$\iota \hookrightarrow (N;\ [\ell_1, \ldots, \ell_k])$	e	$\exists \ell, \iota \hookrightarrow (N;\ [\ell_1, \ldots, \ell_k, \ell])$

Oddershede Gregersen, Aguirre, Haselwarter, Tassarotti, and Birkedal. *Asynchronous Probabilistic Couplings in Higher-Order Separation Logic.*

Presampling Tapes

Eris

- **Future random events as ghost state**

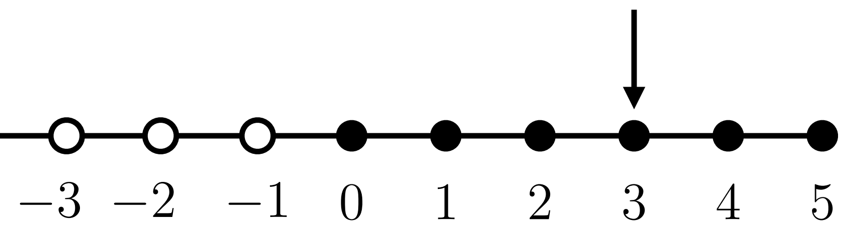
Label, Bound, Presamples

$$\iota \hookrightarrow (N;\ [\ell_1, \ell_2, \ldots])$$

Starting with…	*Execute…*	*To get…*
$\iota \hookrightarrow (N;\ [\ell_1, \ell_2, \ldots])$	$\mathsf{rand}_\iota(N) \rightsquigarrow \ell_1$	$\iota \hookrightarrow (N;\ [\ell_2, \ldots])$
$\iota \hookrightarrow (N;\ [])$	$\mathsf{rand}_\iota(N) \rightsquigarrow_{1/N} n$	$\iota \hookrightarrow (N;\ [])$
$\iota \hookrightarrow (N;\ [\ell_1, \ldots, \ell_k]) * ↯(\epsilon) * \epsilon = \mathbb{E}[\epsilon']$	e	$\exists \ell, \iota \hookrightarrow (N;\ [\ell_1, \ldots, \ell_k, \ell]) * ↯(\epsilon'(\ell))$

Oddershede Gregersen, Aguirre, Haselwarter, Tassarotti, and Birkedal. *Asynchronous Probabilistic Couplings in Higher-Order Separation Logic.*

Planner Rule

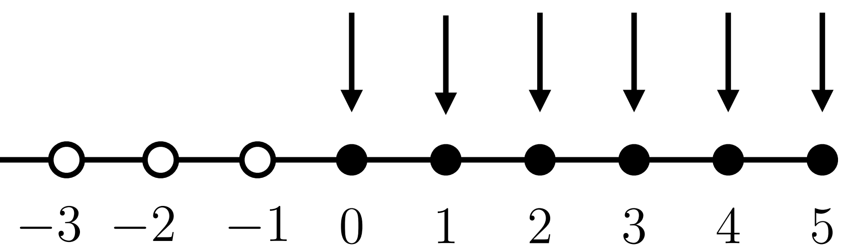


$$
\begin{array}{l}
\textsf{rec cliff } n = \\
\quad \textsf{if } (n < 0) \\
\qquad \textsf{then } n \\
\qquad \textsf{else let d} = \textsf{rand}_\iota\ 2 \textsf{ in} \\
\qquad\qquad \textsf{cliff } \max(n - 1 + d, 5)
\end{array}
$$

You will fall off of the cliff with probability 1.

Planner Rule

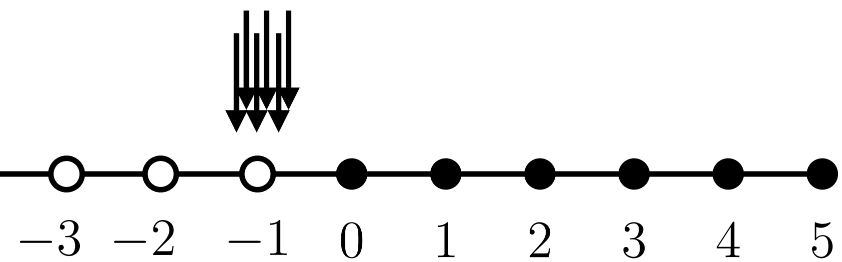
- If the sequence [0, 0, 0, 0, 0, 0] is ever sampled, the process will end



- Presample-amplify six samples at a time

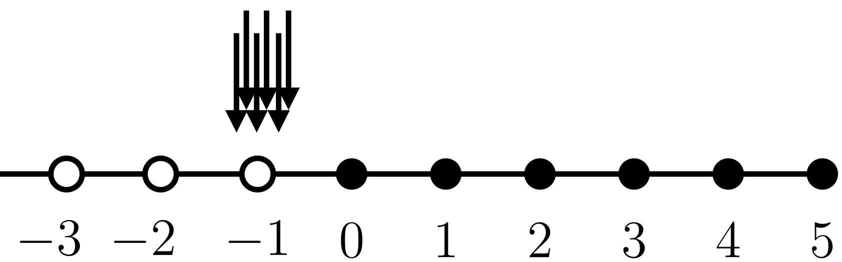
Planner Rule

- If the sequence [0, 0, 0, 0, 0, 0] is ever sampled, the process will end



Planner Rule

- If the sequence [0, 0, 0, 0, 0, 0] is ever sampled, the process will end



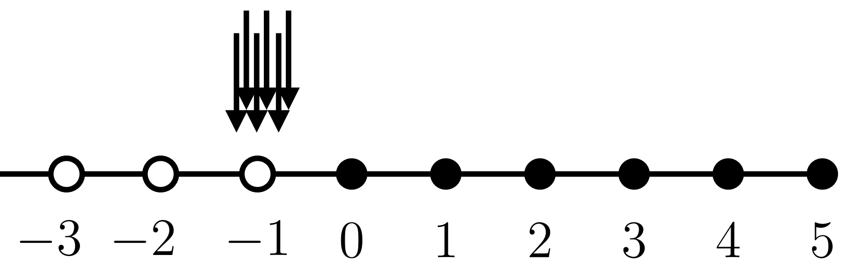
- Presample-amplify six samples at a time

$$\lightning(\epsilon) * \iota \hookrightarrow (2;\ L) \xrightarrow{3^{-6}} \lightning(0) * \iota \hookrightarrow (2;\ L + [0, 0, 0, 0, 0, 0])$$

$$\lightning(\epsilon) * \iota \hookrightarrow (2;\ L) \xrightarrow{1 - 3^{-6}} \lightning\left(\frac{1}{1-3^{-6}}\right) * \iota \hookrightarrow (2;\ L + L')$$

Planner Rule

- If the sequence [0, 0, 0, 0, 0, 0] is ever sampled, the process will end



- Presample-amplify six samples at a time

Arbitrarily small error credit….

…[0, 0, 0, 0, 0, 0] eventually occurs

$$\lightning(\epsilon) * \iota \hookrightarrow (2;\ L) \longrightarrow \exists L'\ \iota \hookrightarrow (2;\ L + L' + [0,0,0,0,0,0])$$

Planner Rule

Given arbitrarily small error credit….

$$\frac{0 < \varepsilon \qquad \forall s.\ |z(s)| \leq L \qquad \vdash \langle \exists ys.\ \iota \hookrightarrow (N, xs \mathbin{+\!\!+} ys \mathbin{+\!\!+} z(xs \mathbin{+\!\!+} ys)) \rangle\ e\ \langle \phi \rangle}{\vdash \langle \iota \hookrightarrow (N, xs) * \text{↯}(\varepsilon) \rangle\ e\ \langle \phi \rangle}\ \textsc{presample-planner}$$

…any possible event eventually occurs.

Planner Rule

Amir Pnueli

Parameterized Verification by Probabilistic Abstraction*

Tamarah Arons[1], Amir Pnueli[1], and Lenore Zuck[2]

[1] Weizmann Institute of Science, Rehovot, Israel,
{amir,tamarah}@wisdom.weizmann.ac.il
[2] New York University, New York,
zuck@cs.nyu.edu

In this paper we propose two novel approaches to the problem. The first is based on *Planners* and the second on the notion of γ-*fairness* introduce in [ZPK02]. When activated, a planner pre-determines the results of a the next k consecutive "random" choices, allowing these next choices to be performed in a completely non-deterministic manner.

Arons, Pnueli, Zuck. *Pareterized Verification by Probabilistic Abstraction.*

Total Eris

- Expected termination bounds as a separation logic resource
- Modular proofs of termination bounds
- Error induction, planner rule

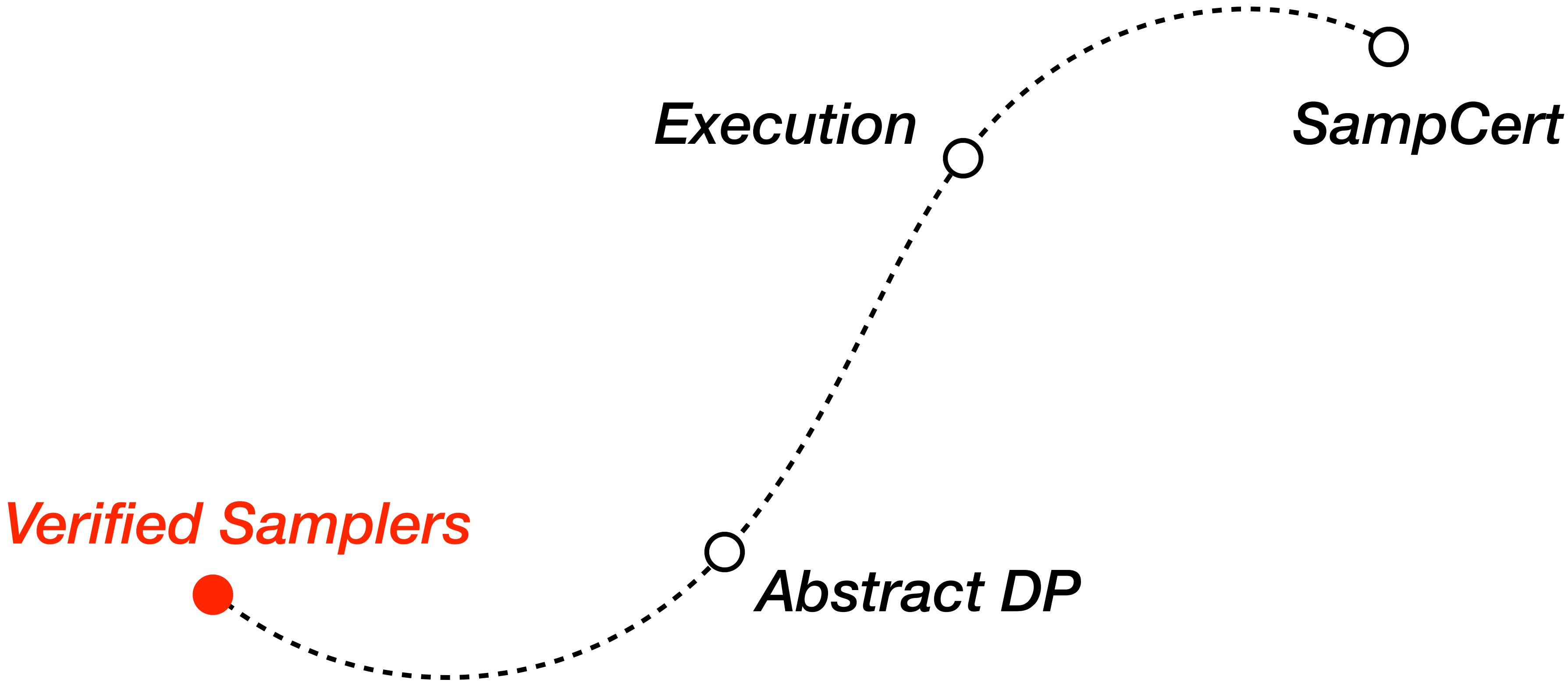
Section 3.

SampCert

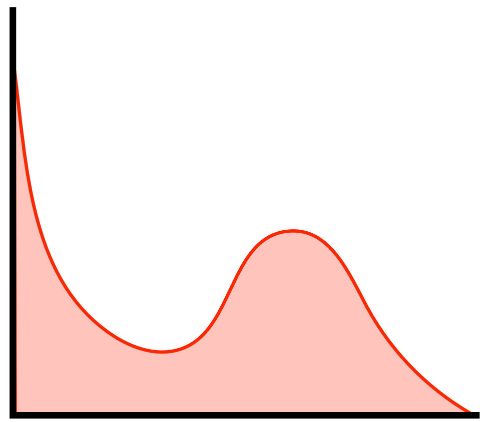
Verified Differential Privacy

SampCert

- Foundational Verification
- Reuse established DP theory
- Deployable code


Execution
SampCert
Verified Samplers
Abstract DP

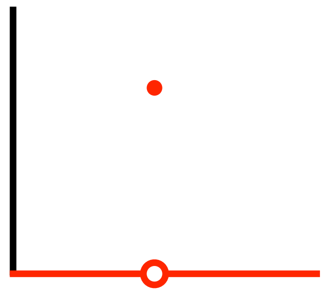
Shallowly Embedded Probability

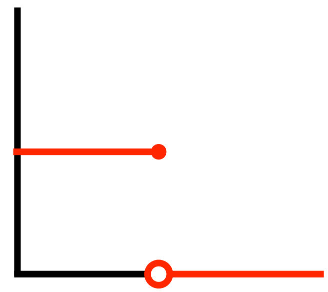


```
abbev PMF (T : Type _) := T → ℝ
```

SLang

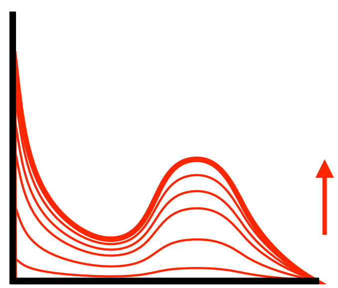
Return

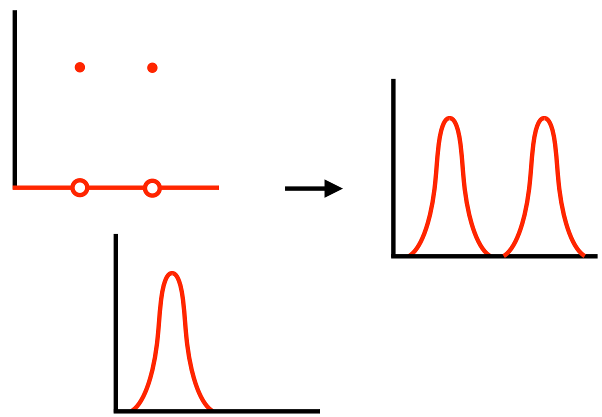




Uniform

While



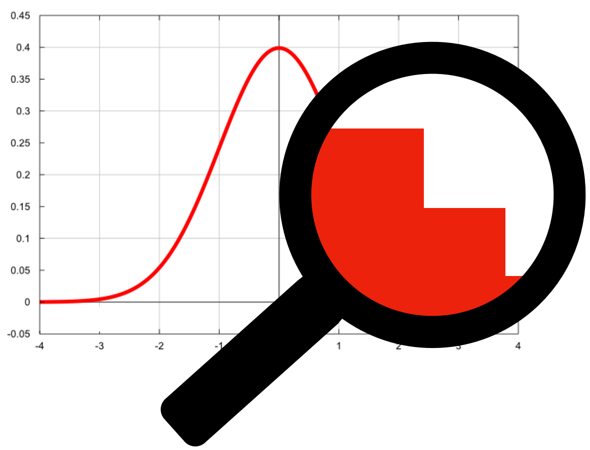


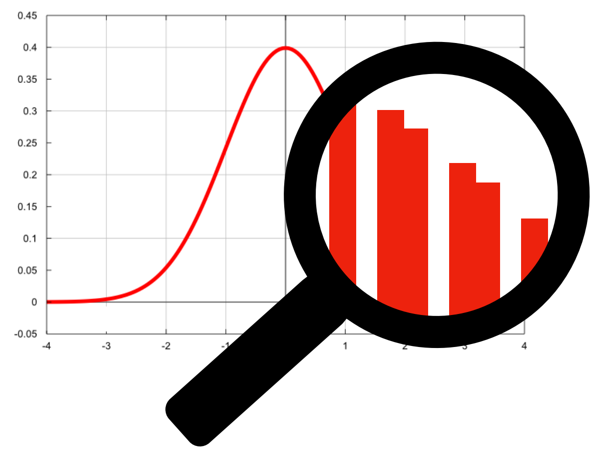
Bind

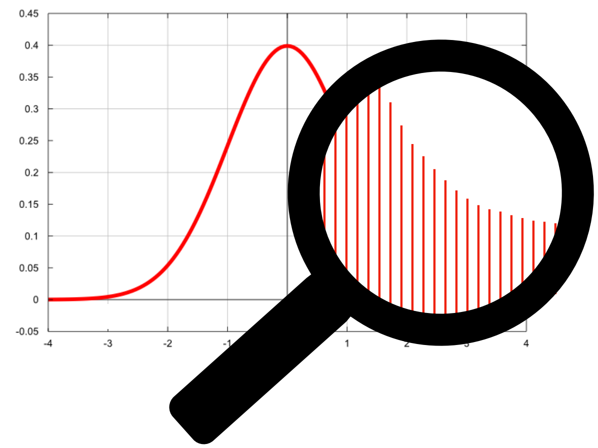
Differentially Private Sampling

Snapping

Discrete Sampling







Barthe, Köpf, Olmedo, and Zanella-Béguelin. *Probabilistic Relational Reasoning for Differential Privacy.*
Mironov. *On significance of the least significant bits for differential privacy.*

Differentially Private Sampling

```
/-- Sample a candidate for the Discrete Gaussian with variance ``num/den``. -/
def DiscreteGaussianSampleLoop (num den t : PNat) (mix : ℕ) : SLang (Int × Bool) := do
  let Y : Int ← DiscreteLaplaceSampleMixed t 1 mix
  let y : Nat := Int.natAbs Y
  let n : Nat := (Int.natAbs (Int.sub (y * t * den) num))^2
  let d : PNat := 2 * num * t^2 * den
  let C ← BernoulliExpNegSample n d
  return (Y,C)

/-- Sample a value from the Discrete Gaussian with variance ``(num/den)``^2. -/
def DiscreteGaussianSample (num : PNat) (den : PNat) (mix : ℕ) : SLang ℤ := do
  let ti : Nat := num.val / den
  let t : PNat := ⟨ ti + 1 , by simp only [add_pos_iff, zero_lt_one, or_true] ⟩
  let num := num^2
  let den := den^2
  let r ← probUntil (DiscreteGaussianSampleLoop num den t mix) (λ x : Int × Bool => x.2)
  return r.1
```

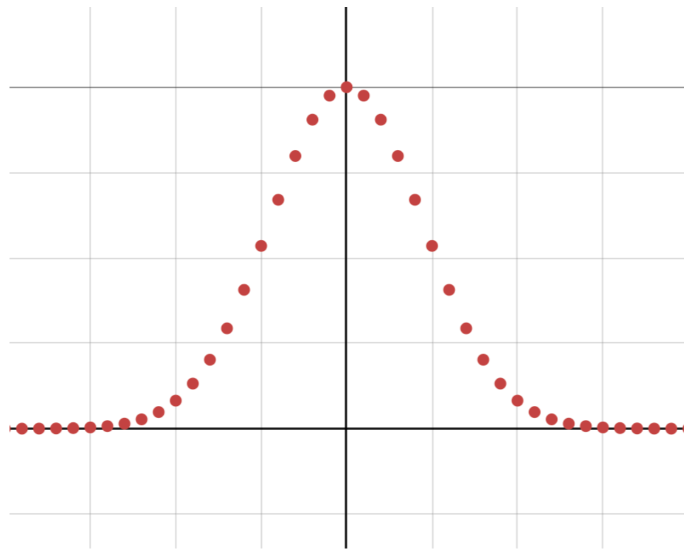
Program ⟶ `PMF T`

Proof ⟶ **Lean Proof**

Specification ⟶ `(PMF T) → Prop`

Verified Samplers

$$\frac{e^{-(z-\mu)^2/2\sigma^2}}{\sum_{z\in\mathbb{Z}} e^{-(z-\mu)^2/2\sigma^2}}$$



Canone, Kamath, Steinke. *The discrete Gaussian for differential privacy.*

Verified Samplers

Algorithm 2 Algorithm for Sampling a Discrete Laplace

Input: Parameters $s, t \in \mathbb{Z}$, $s, t \geq 1$.
Output: One sample from $\mathrm{Lap}_{\mathbb{Z}}(t/s)$.

```
loop
    Sample U ∈ {0, 1, 2, ··· , t − 1} uniformly at random.
    Sample D ← Bernoulli(exp(−U/t)).
    if D = 0 then reject and restart.
    Initialize V ← 0.
    loop
        Sample A ← Bernoulli(exp(−1)).
        if A = 0 then break the loop.
        if A = 1 then set V ← V + 1 and continue.
    Set X ← U + t · V.
    Set Y ← ⌊X/s⌋
    Sample B ← Bernoulli(1/2).
    if B = 1 and Y = 0 then reject and restart.
    return Z ← (1 − 2B) · Y.
```

Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$.

```
Set t ← ⌊σ⌋ + 1
loop
    Sample Y ← Lap_Z(t)
    Sample C ← Bernoulli(exp(−(|Y| − σ²/t)²/2σ²)).
    If C = 0, reject and restart.
    If C = 1, return Y as output.
```

Canone, Kamath, Steinke. *The discrete Gaussian for differential privacy.*

Verified Samplers

Algorithm 2 Algorithm for Sampling a Discrete Laplace

Input: Parameters $s, t \in \mathbb{Z}$, $s, t \geq 1$.
Output: One sample from $\mathrm{Lap}_{\mathbb{Z}}(t/s)$.

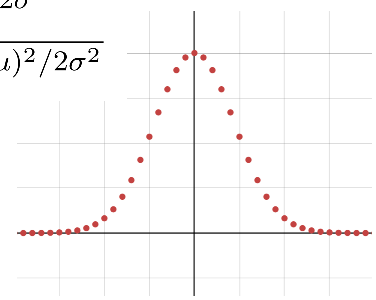
- **loop**
  - Sample $U \in \{0, 1, 2, \cdots, t-1\}$ uniformly at random.
  - Sample $D \leftarrow \mathsf{Bernoulli}(\exp(-U/t))$.
  - **if** $D = 0$ **then** reject and restart.
  - Initialize $V \leftarrow 0$.
  - **loop**
    - Samp
    - **if** A
    - **if** A
  - Set $X \leftarrow$
  - Set $Y \leftarrow$
  - Sample
  - **if** $B = 1$
  - **return**

Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$.

- Set $t \leftarrow \lfloor \sigma \rfloor + 1$
- **loop**
  - Sample $Y \leftarrow \mathrm{Lap}_{\mathbb{Z}}(t)$
  - Sample $C \leftarrow \mathsf{Bernoulli}(\exp(-(|Y| - \sigma^2/t)^2/2\sigma^2))$.
  - If $C = 0$, reject and restart.
  - If $C = 1$, return Y as output.

$$\frac{e^{-(z-\mu)^2/2\sigma^2}}{\sum_{z \in \mathbb{Z}} e^{-(z-\mu)^2/2\sigma^2}}$$



```
do let Y : Int ← DiscreteLaplaceSampleMixed t 1 mix
   let y : Nat := Int.natAbs Y
   let n : Nat := (Int.natAbs (Int.sub (y * t * den) num))^2
   let d : PNat := 2 * num * t^2 * den
   /— continued ... -/
```

```
PMF ℤ
  =
```

```
def gauss_term_ℝ (σ μ : ℝ) (x : ℝ) : ℝ :=
  Real.exp ((-(x - μ)^2) / (2 * σ^2))
def discrete_gaussian (σ μ : ℝ) (x : ℝ) : ℝ :=
  gauss_term_ℝ σ μ x / ∑' x : ℤ, gauss_term_ℝ σ μ x
```

Poisson Summation Formula *Complex Limits*

Extended Reals *Topology* *Jensen's Inequality*

```
do
  let v ← program
  ...
```

```
do
  let v ← spec
  ...
```

Program

PMF ℤ

=

Specification

PMF ℤ

```
do
  let v ← program
  ...
```

```
do
  let v ← spec
  ...
```

Program PMF ℤ	=	**Specification** PMF ℤ

Termination (AST) Normalization Support	$\Longleftarrow$	Termination (AST) Normalization Support

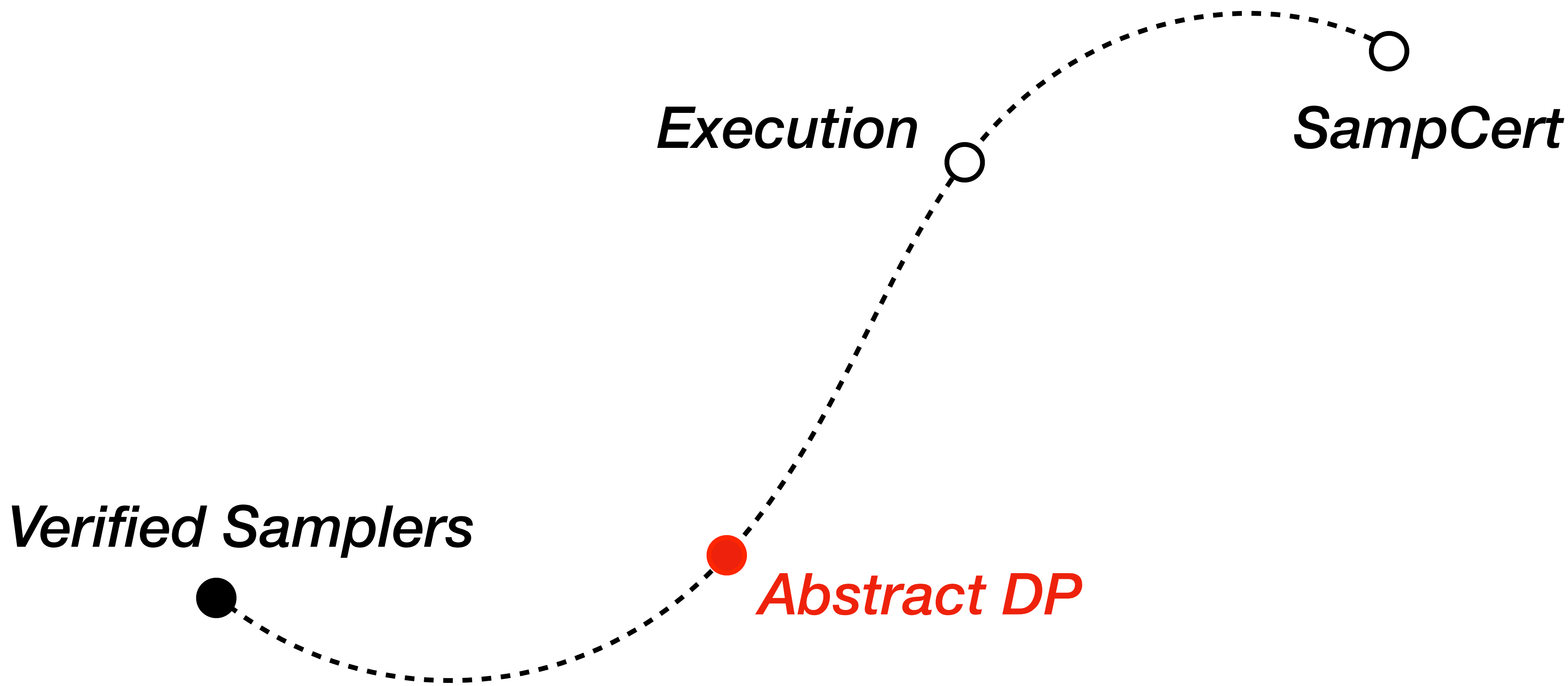
```
do
  let v ← optimized
  …
```

```
do
  let v ← program
  …
```

```
do
  let v ← spec
  …
```

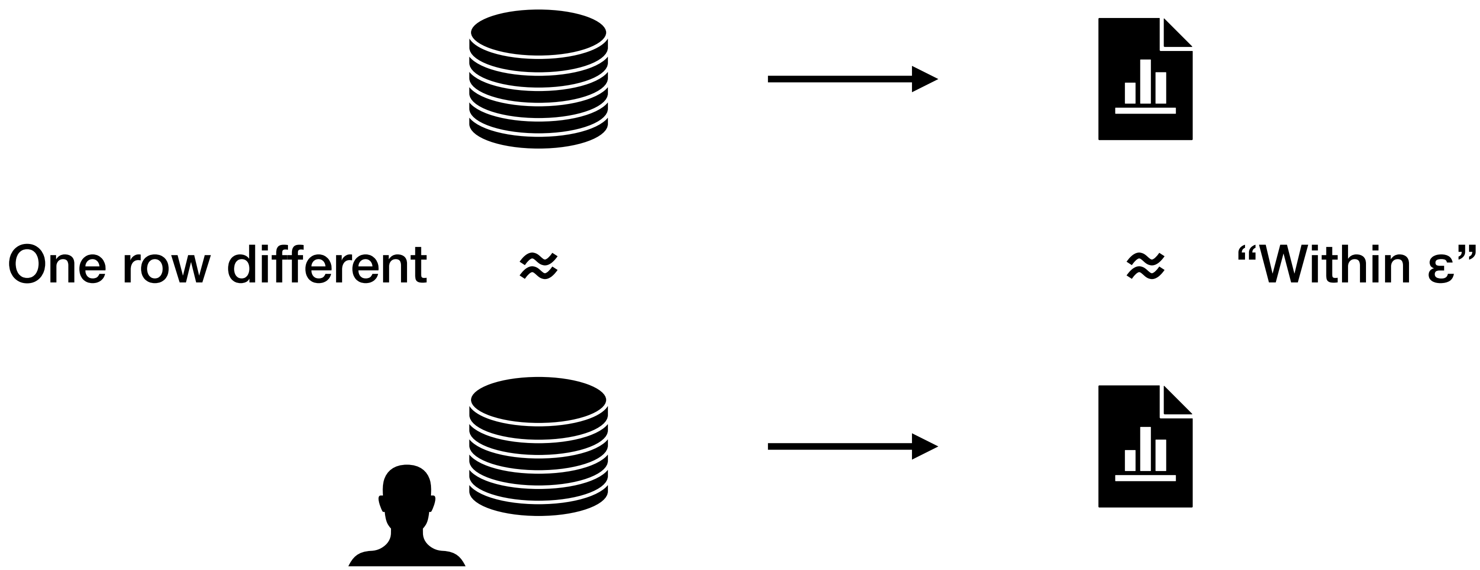
Optimized		**Program**		**Specification**
PMF ℤ	=	PMF ℤ	=	PMF ℤ

Termination (AST) Normalization Support	$\Longleftarrow$	Termination (AST) Normalization Support	$\Longleftarrow$	Termination (AST) Normalization Support


Execution
SampCert
Verified Samplers
Abstract DP

Differential Privacy

A program is ε-DP when…



Differential Privacy

Zero-concentrated DP

Approximate DP

Rényi DP

Concentrated DP

Pure DP

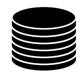
Approximate CDP

Approximate zCDP

Mean-concentrated DP

Steinke. *Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds.*

Abstract Differential Privacy

- (1/3) Privacy Definition `ε-ADP(` → `) : Prop`
- (2/3) Composition Rules

bind	`ε-ADP` additive with respect to `>>=`
map	`ε-ADP` constant with respect to `<$>`
return	`0-ADP (λ_ ⇒ pure v)`

- (3/3) Noise Rule

noise	For `f : T → ℤ` with bounded sensitivity,
`N : PMF ℤ`	`ε-ADP (f + N)`

Abstract Differential Privacy

Abstract DP

- *Privacy*
- *Composition*
- *Noise*

Pure DP

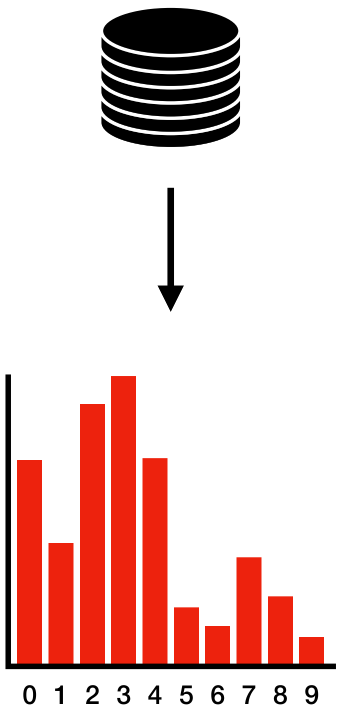
- Pointwise bound
- Discrete Laplace

zCDP

- Information bound
- Discrete Gaussian

Abstract Differential Privacy

```
def privHistogram (D : Database) (n : ℕ) := do
  match n with
  | .zero ⇒ return (emptyHistogram)
  | n’.succ ⇒ do
     let h ← (privHistogram D (n - 1))
     let c ← (count D n + noise ε)
     updateHistogram h n c
```



Abstract Differential Privacy

```
def privHistogram (D : Database) (n : ℕ) := do
  match n with
  | .zero ⇒ return (emptyHistogram)
  | n'.succ ⇒ do
    let h ← (privHistogram D (n - 1))
    let c ← (count D n + noise ε)
    updateHistogram h n c
```

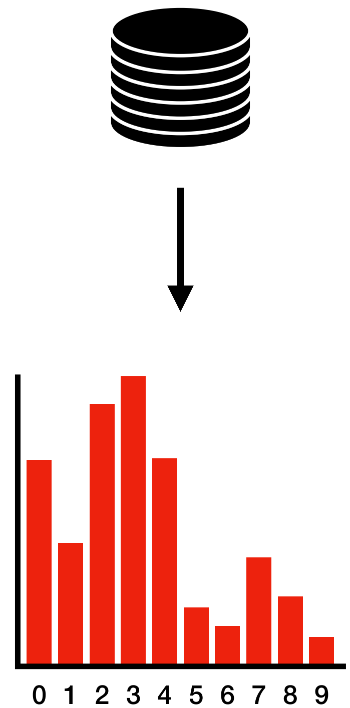
return

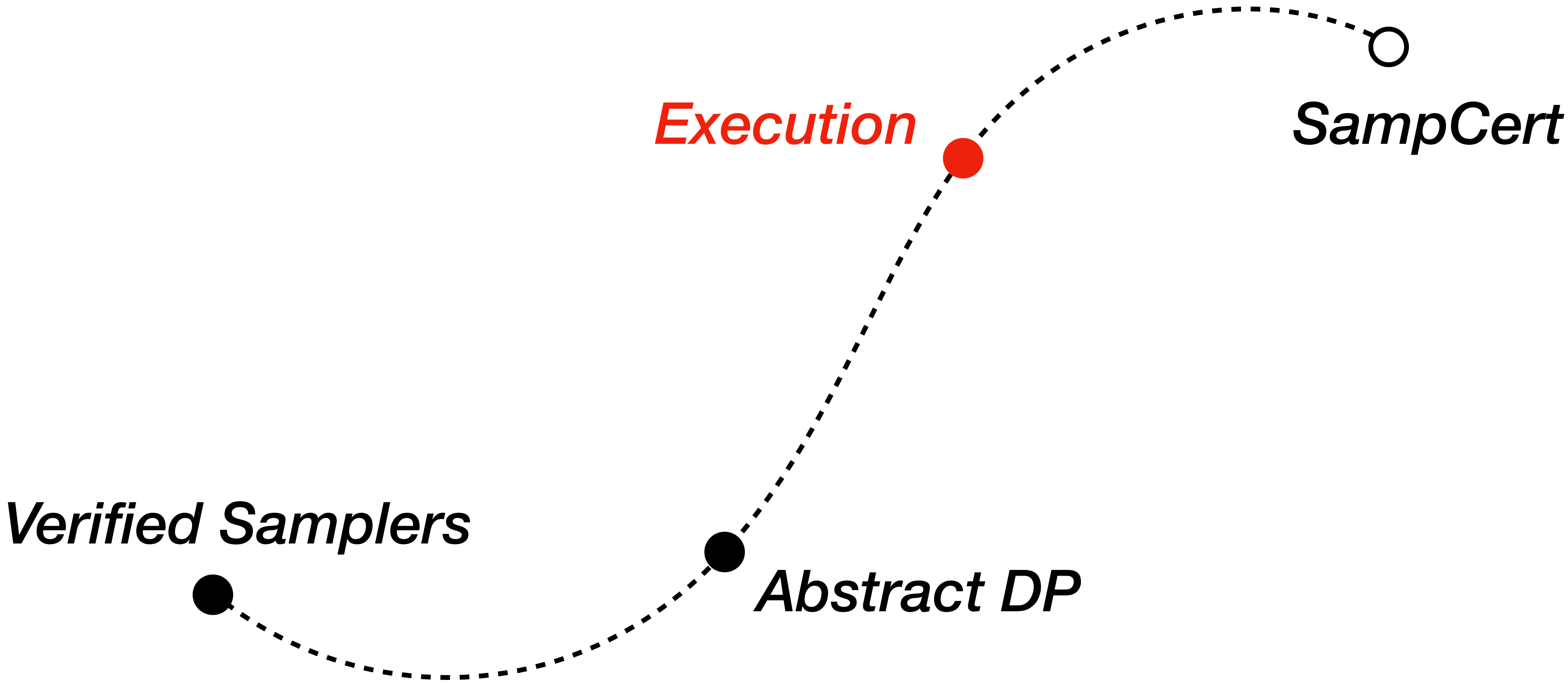
bind

bind

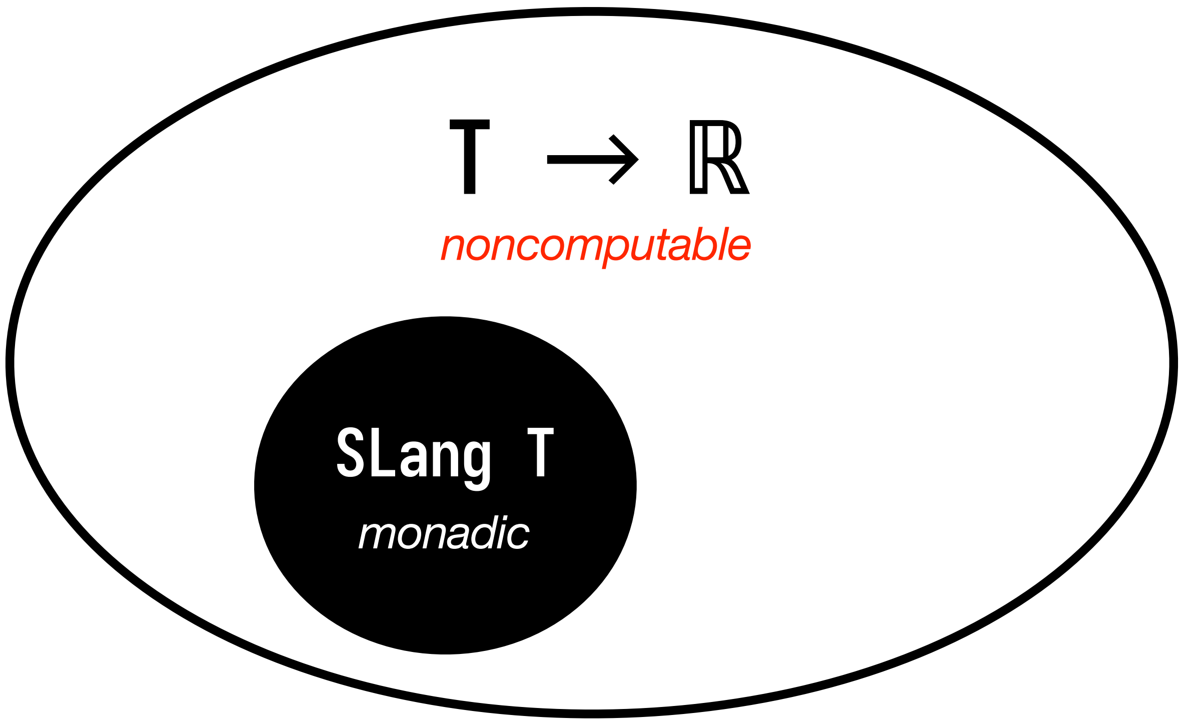
map

noise



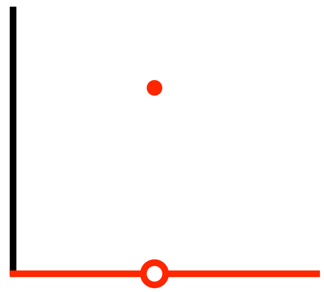

Execution
SampCert
Verified Samplers
Abstract DP

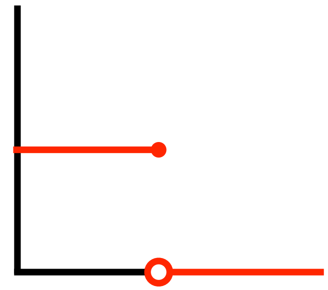
Execution



Execution

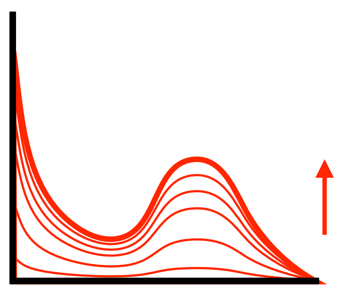
Return

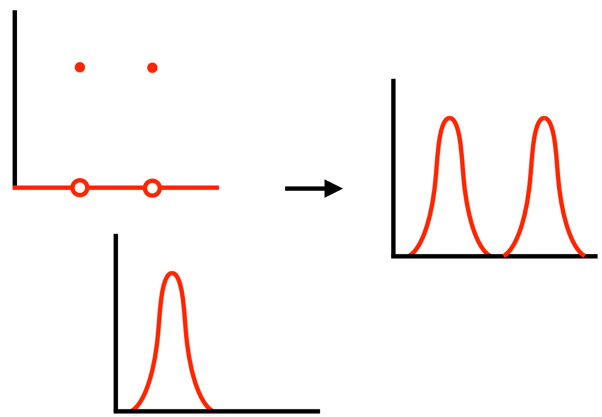




Uniform

While





Bind

Execution

Return

```
return v;
```

Uniform

```
unsigned char r;
read (random, &r, 1);
return r;
```

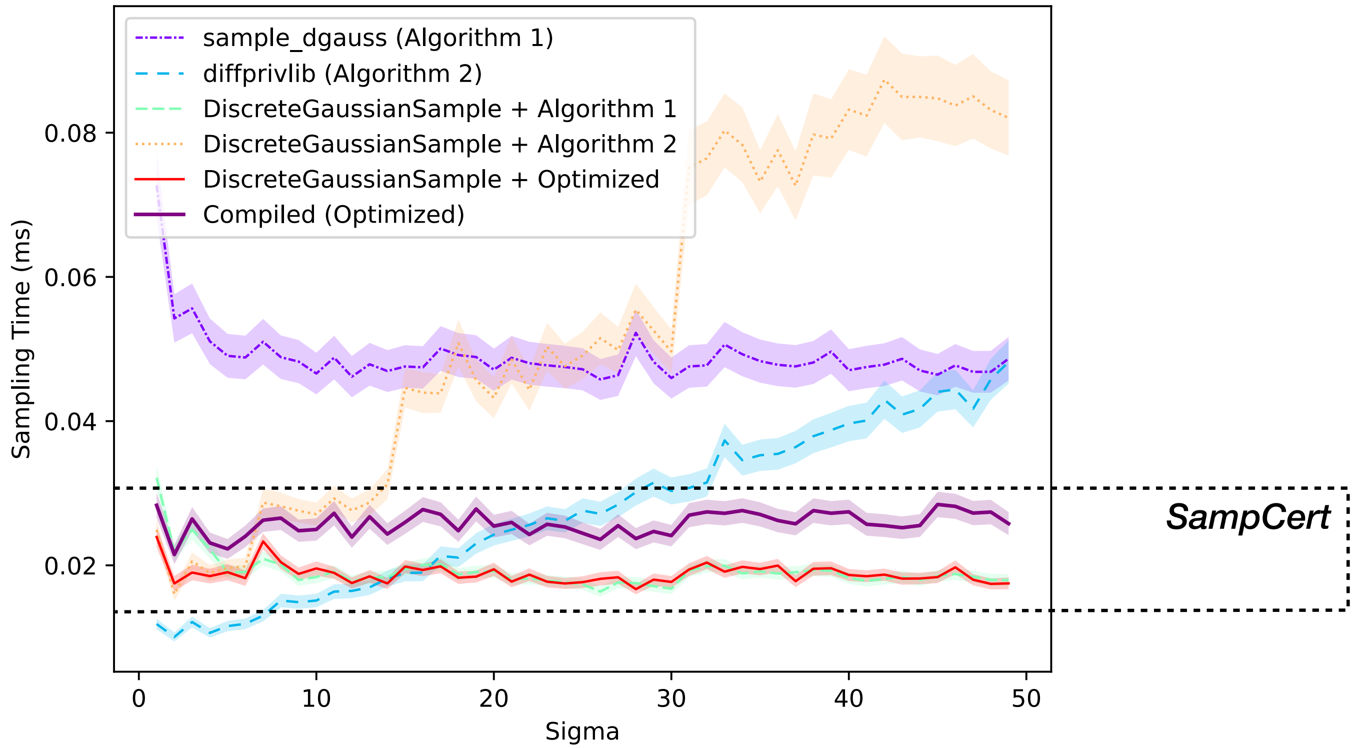
While

```
lean_object* st = init;
uint8_t c = lean_apply_1 (c, st);
while (c) {
    s = lean_apply_2 (b, st);
    c = lean_apply_1 (c, st); }
return st;
```

Bind

```
lean_object* e = lean_apply_1 f;
lean_object* p =
  lean_apply_2 (p, e);
return p;
```

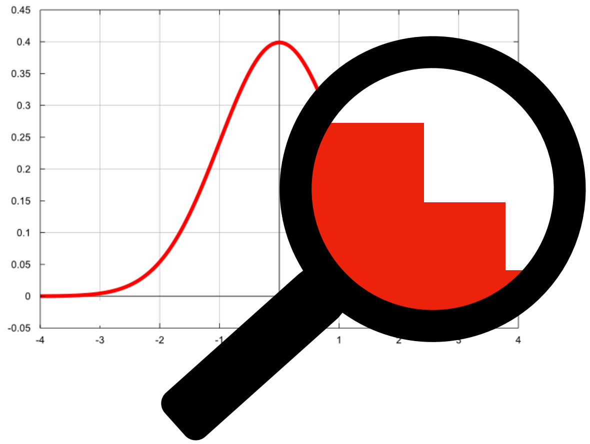
Execution


sample_dgauss (Algorithm 1)
diffprivlib (Algorithm 2)
DiscreteGaussianSample + Algorithm 1
DiscreteGaussianSample + Algorithm 2
DiscreteGaussianSample + Optimized
Compiled (Optimized)
Sampling Time (ms)
0.08
0.06
0.04
0.02
0
10
20
30
40
50
Sigma
SampCert

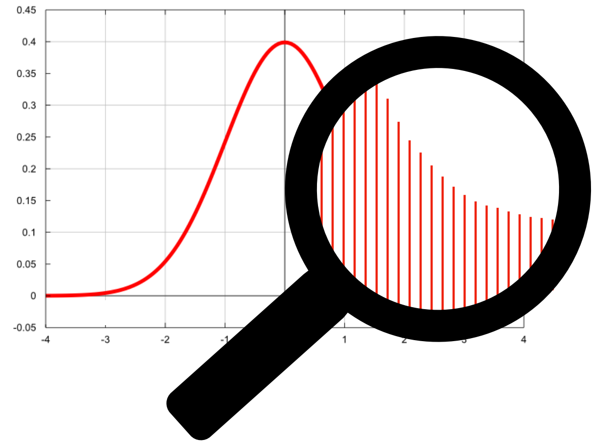
Section 4.

Continuous Eris

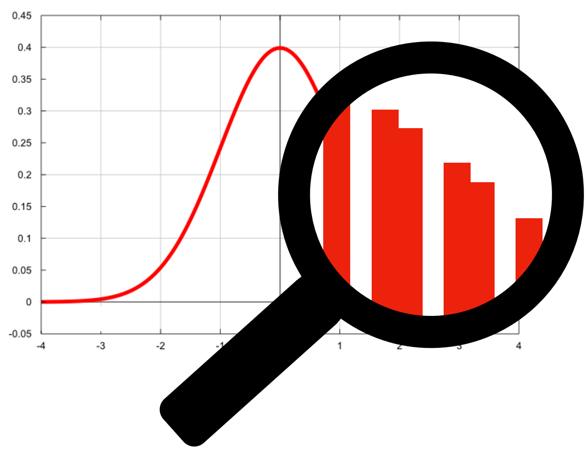
Snapping



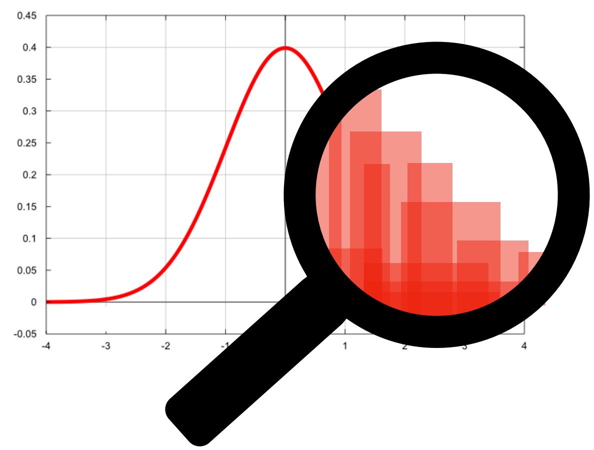
Discrete



Floating Point



Exact Reals



Exact Reals

$$\mathbb{R} \approx \mathbb{N} \to \{0,1\}$$

- Arithmetic (+, -, *, /) ●
- Transcendental functions ●
- Inequalities ◐

Ménissier-Morain. *Arbitrary precision real arithmetic: design and algorithms.*

Exact Reals

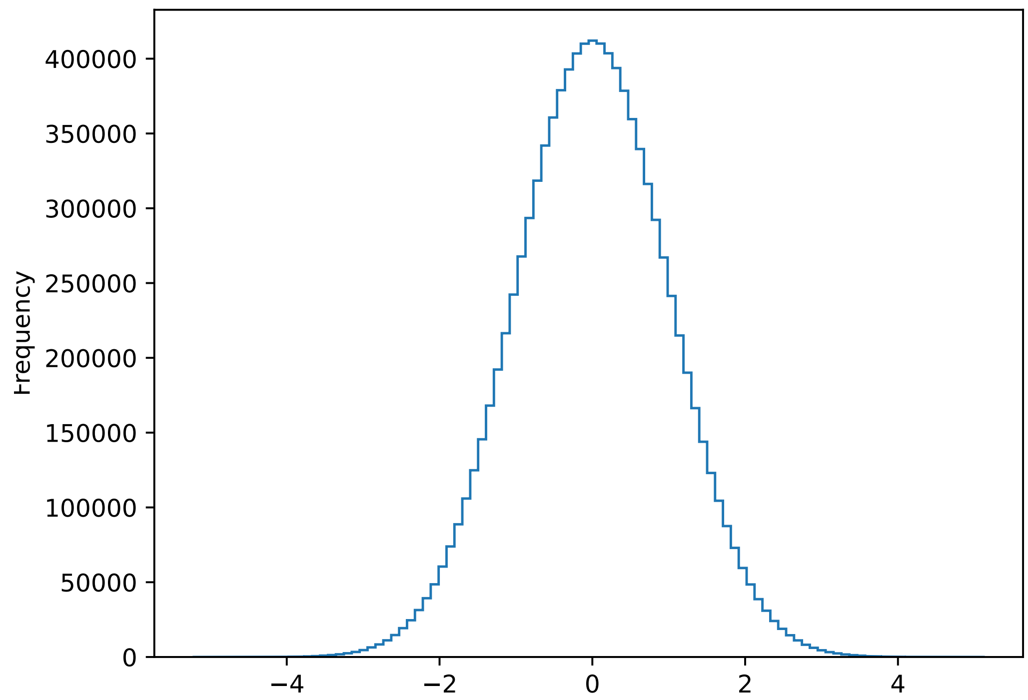
$$
\begin{array}{l}
\textsf{rec}\ \textsf{makeUniform}\ \_ = \\
\quad \textsf{let}\ \textsf{c} = make\_cache\ \textsf{in} \\
\quad (\lambda n. \\
\quad\quad \textsf{if}\ n \notin c\ \textsf{then}\ c[n] \leftarrow \textsf{rand}\ 2 \\
\quad\quad c[n])
\end{array}
$$

- Higher-order, stateful program

Exact Reals

```
type zu = {
  mutable zpart : Z.t;
  mutable upart : Int32.t array
}
```

- Arbitrary precision samples
- Fast integer comparisons



Karney. *Sampling exactly from the normal distribution.*

Continuous Eris

- Version of Eris supporting a generalized averaging rule

$$\epsilon' : [0,1] \to \mathbb{R}_{\geq 0} \qquad \epsilon = \int_0^1 \epsilon'(x)dx$$

$$\frac{\forall r \in [0,1],\ \{\lightning(\epsilon'(r))\}\ \mathsf{realOf}\ r\ \{P\}}{\{\lightning(\epsilon)\}\ \mathsf{makeUniform}\ \{P\}}$$

Von Neumann's Algorithm

- **Input: an exact real** $x \in [0, 1]$
- **Sample a decreasing sequence of uniform reals**

$$x > \mathcal{U}_1 > \mathcal{U}_2 > \ldots \leq \mathcal{U}_k$$

Terminates with probability 1
Requires only integer comparisons

- **Return True if** k **is even.**

Probability that the algorithm returns true?

$$\begin{aligned}
&\mathbb{P}[\mathsf{true} \leftarrow \mathcal{V}] \\
&= \sum_{n \text{ even}} \mathbb{P}[k = n] \\
&= \sum_{n \text{ even}} \mathbb{P}[n \leq k \wedge n + 1 \not\leq k] \\
&= \sum_{n \text{ even}} \left(\frac{x^n}{n!} - \frac{x^{n+1}}{(n+1)!} \right) \\
&= \sum_{n \in \mathbb{N}} \frac{(-x)^n}{n!} \\
&= e^{-x}
\end{aligned}$$

Karney. *Sampling exactly from the normal distribution*.

Von Neumann's Algorithm verified

- Correctness of the sampler in Eris?

$$\mu(\mathsf{True}) = e^{-x} \qquad\qquad \mu(\mathsf{False}) = 1 - e^{-x}$$

$$\forall \epsilon_{\mathsf{True}}, \epsilon_{\mathsf{False}} \in \mathbb{R}_{\geq 0},\ \{\lightning(\mu(\mathsf{True})\epsilon_{\mathsf{True}} + \mu(\mathsf{False})\epsilon_{\mathsf{False}})\}\ \mathcal{V}\ \{b,\ \lightning(\epsilon_b)\}$$

- Proven by Löb induction, using the integral expectation rule for each uniform

Von Neumann's Algorithm verified

- Sample x using $g : [0,1] \to \mathbb{R}$ and e using $h_x : \mathbb{N} \to \mathbb{R}$ so that we get the system of equations

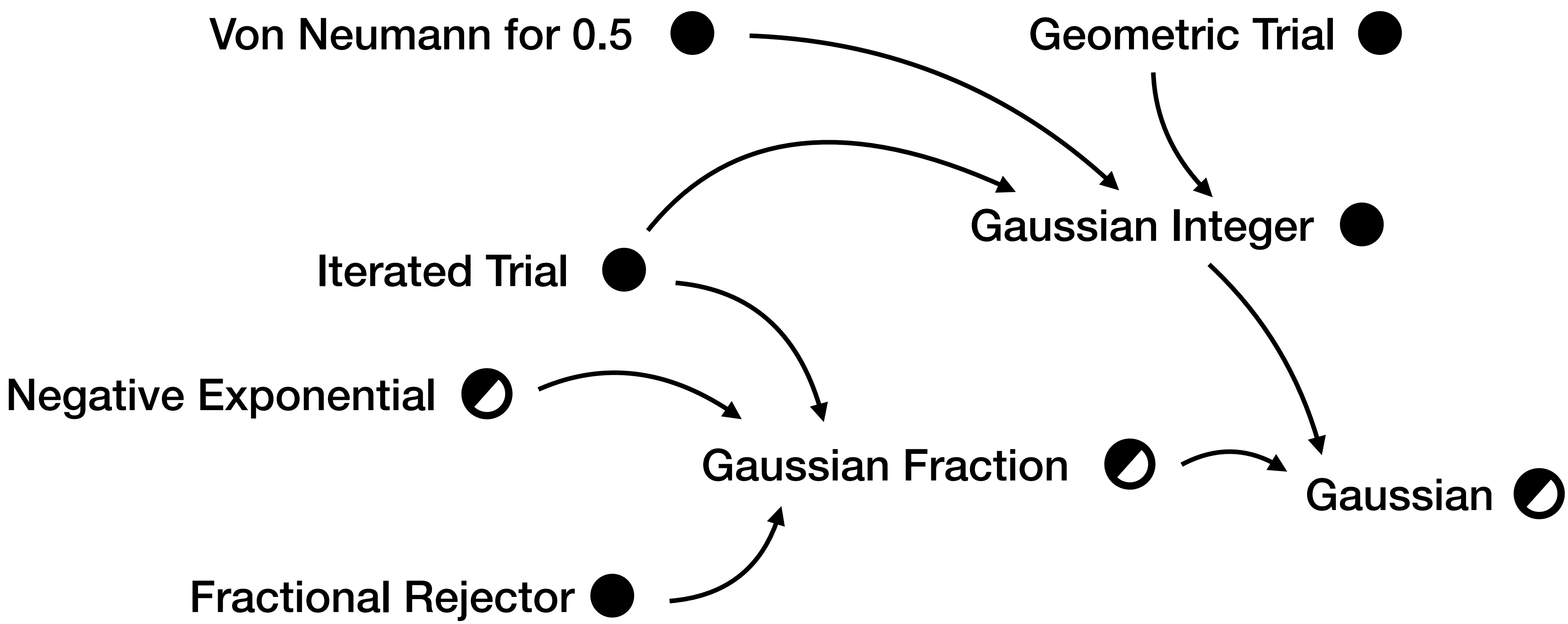
$$e^{-1/2}F(\text{True}) + (1 - e^{-1/2})F(\text{False}) = \int_0^1 g(x)dx$$
$$\forall x > 1/2,\ g(x) = F(\text{True})$$
$$\forall x \le 1/2,\ g(x) = \sum_{n\in\mathbb{N}} \mu(x,n)h_x(n)$$
$$\forall x \le 1/2,\ h_x(n) = F(n\%2 = 1)$$

Or in other words

$$\begin{aligned}
e^{-1/2}F(\text{True}) + (1 - e^{-1/2})F(\text{False}) &= \int_0^1 [\![x > 1/2]\!]F(\text{True}) + [\![x \le 1/2]\!] \sum_{n\in\mathbb{N}} \mu(x,n)F(n\%2 = 1)dx \\
&= F(\text{True})/2 + \int_0^1 [\![x \le 1/2]\!] \sum_{n\in\mathbb{N}} \mu(x,n)F(n\%2 = 1)dx \\
&= F(\text{True})/2 + \sum_{n\in\mathbb{N}} F(n\%2 = 1) \int_0^1 [\![x \le 1/2]\!]\mu(x,n)dx \\
&= F(\text{True})/2 + \sum_{n\in\mathbb{N}} F(n\%2 = 1)\ (\mu(1/2, n+1) - \mu(0, n+1)) \\
&= F(\text{True})/2 + \sum_{n\in\mathbb{N}} F(n\%2 = 1)\mu(1/2, n+1) \\
&= F(\text{True})/2 - F(\text{True})\mu(1/2, 0) + \sum_{n\in\mathbb{N}} F(n\%2 = 0)\mu(1/2, n) \\
&= \sum_{n\in\mathbb{N}} F(n\%2 = 0)\mu(1/2, n) \\
&= \sum_{n\in\mathbb{N}} [\![n \text{ even}]\!]F(\text{True})\mu(1/2, n) + \sum_{n\in\mathbb{N}} [\![n \text{ odd}]\!]F(\text{False})\mu(1/2, n) \\
&= e^{1/2}F(\text{True}) + (1 - e^{-1/2})F(\text{False})
\end{aligned}$$

- System of credit equations
- Solve using Fubini's Theorem

Karney's Sampling Algorithm



Verifying Probabilistic Programs

Verify approximate fixed-point samplers

Convergence for approximate programs (MCMC)

- Eris
- Total Eris
- Tachis
- SampCert
- Continuous Eris

Verified semantic security models

Verify sampler optimizations

Verified DP type systems

DP for snapping floating-point samplers

Verify compilation step

Verified statistical models

Continuous relational logic

Verified exact stationary samplers (CFTP)