

On Higher-Order Probabilistic Computation: Relational Reasoning, Termination, and Bayesian Programming

Ugo Dal Lago

(Based on joint work with *Michele Alberti, Raphaëlle
Crubillé, Charles Grellois, Davide Sangiorgi, . . .*)



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

inria
informatiques mathématiques

IFIP WG 2.2 Annual Meeting, Brno, September 17th

Probabilistic Models

- ▶ The **environment** is supposed not to behave *deterministically*, but *probabilistically*.

Probabilistic Models

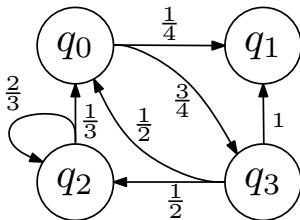
- ▶ The **environment** is supposed not to behave *deterministically*, but *probabilistically*.
- ▶ Crucial when modeling **uncertainty**.

Probabilistic Models

- ▶ The **environment** is supposed not to behave *deterministically*, but *probabilistically*.
- ▶ Crucial when modeling **uncertainty**.
- ▶ Useful to handle **complex** domains.

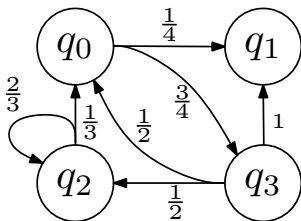
Probabilistic Models

- ▶ The **environment** is supposed not to behave *deterministically*, but *probabilistically*.
- ▶ Crucial when modeling **uncertainty**.
- ▶ Useful to handle **complex** domains.
- ▶ **Example:**



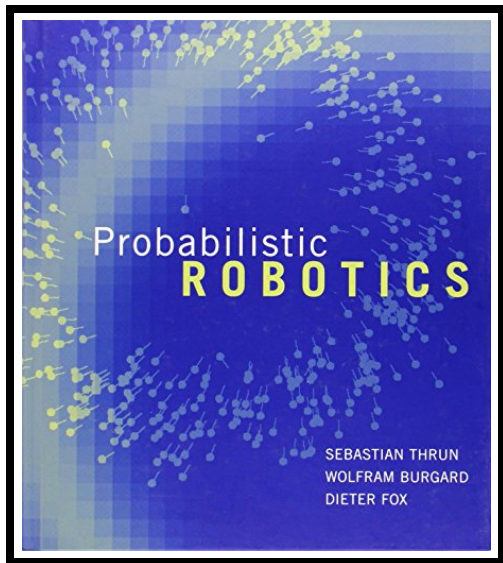
Probabilistic Models

- ▶ The **environment** is supposed not to behave *deterministically*, but *probabilistically*.
- ▶ Crucial when modeling **uncertainty**.
- ▶ Useful to handle **complex** domains.
- ▶ **Example:**



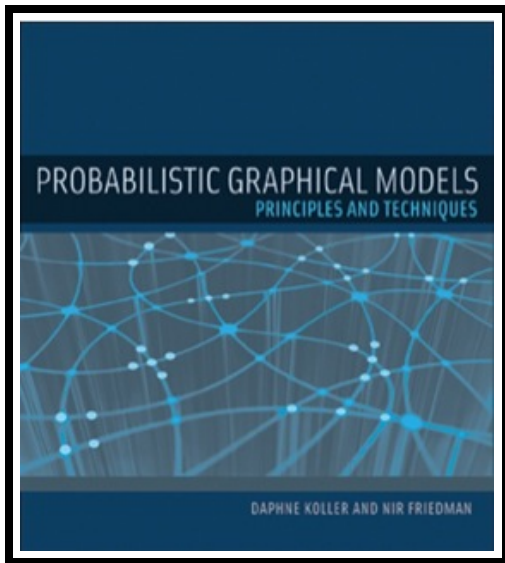
- ▶ **Abstractions:**
 - ▶ (Labelled) Markov Chains.

Probabilistic Models



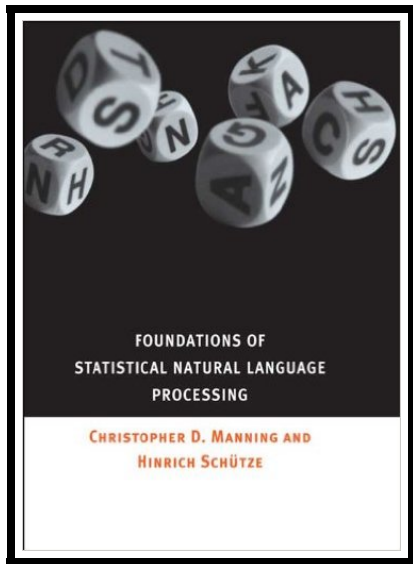
ROBOTICS

Probabilistic Models



ARTIFICIAL
INTELLIGENCE

Probabilistic Models



NATURAL LANGUAGE PROCESSING

Randomized Computation

- ▶ Algorithms and automata are assumed to have the ability to **sample** from a distribution [dLMSS1956,R1963].

Randomized Computation

- ▶ Algorithms and automata are assumed to have the ability to **sample** from a distribution [dLMSS1956,R1963].
- ▶ This is a **powerful tool** when solving computational problems.

Randomized Computation

- ▶ Algorithms and automata are assumed to have the ability to **sample** from a distribution [dLMSS1956,R1963].
- ▶ This is a **powerful tool** when solving computational problems.
- ▶ **Example:**

```
Input:  $n > 3$ , an odd integer to be tested for primality;  
Input:  $k$ , a parameter that determines the accuracy of the test  
Output: composite if  $n$  is composite, otherwise probably prime  
write  $n - 1$  as  $2^s \cdot d$  with  $d$  odd by factoring powers of 2 from  $n - 1$   
WitnessLoop: repeat  $k$  times:  
    pick a random integer  $a$  in the range  $[2, n - 2]$   
     $x \leftarrow a^d \bmod n$   
    if  $x = 1$  or  $x = n - 1$  then do next WitnessLoop  
    repeat  $s - 1$  times:  
         $x \leftarrow x^2 \bmod n$   
        if  $x = 1$  then return composite  
        if  $x = n - 1$  then do next WitnessLoop  
    return composite  
return probably prime
```

Randomized Computation

- ▶ Algorithms and automata are assumed to have the ability to **sample** from a distribution [dLMSS1956,R1963].
- ▶ This is a **powerful tool** when solving computational problems.
- ▶ **Example:**

```
Input:  $n > 3$ , an odd integer to be tested for primality;  
Input:  $k$ , a parameter that determines the accuracy of the test  
Output: composite if  $n$  is composite, otherwise probably prime  
write  $n - 1$  as  $2^s \cdot d$  with  $d$  odd by factoring powers of 2 from  $n - 1$   
WitnessLoop: repeat  $k$  times:  
    pick a random integer  $a$  in the range  $[2, n - 2]$   
     $x \leftarrow a^d \bmod n$   
    if  $x = 1$  or  $x = n - 1$  then do next WitnessLoop  
    repeat  $s - 1$  times:  
         $x \leftarrow x^2 \bmod n$   
        if  $x = 1$  then return composite  
        if  $x = n - 1$  then do next WitnessLoop  
    return composite  
return probably prime
```

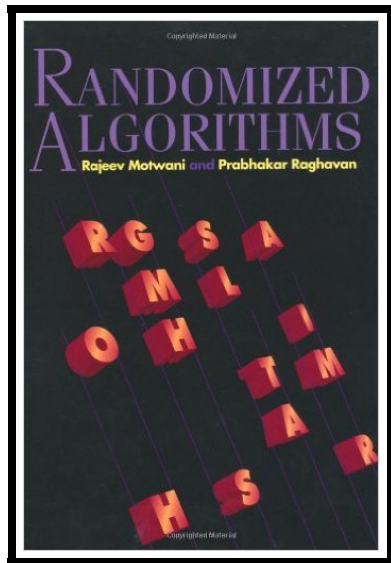
Randomized Computation

- ▶ Algorithms and automata are assumed to have the ability to **sample** from a distribution [dLMSS1956,R1963].
- ▶ This is a **powerful tool** when solving computational problems.
- ▶ **Example:**

```
Input:  $n > 3$ , an odd integer to be tested for primality;
Input:  $k$ , a parameter that determines the accuracy of the test
Output: composite if  $n$  is composite, otherwise probably prime
write  $n - 1$  as  $2^s \cdot d$  with  $d$  odd by factoring powers of 2 from  $n - 1$ 
WitnessLoop: repeat  $k$  times:
    pick a random integer  $a$  in the range  $[2, n - 2]$ 
     $x \leftarrow a^d \bmod n$ 
    if  $x = 1$  or  $x = n - 1$  then do next WitnessLoop
    repeat  $s - 1$  times:
         $x \leftarrow x^2 \bmod n$ 
        if  $x = 1$  then return composite
        if  $x = n - 1$  then do next WitnessLoop
    return composite
return probably prime
```

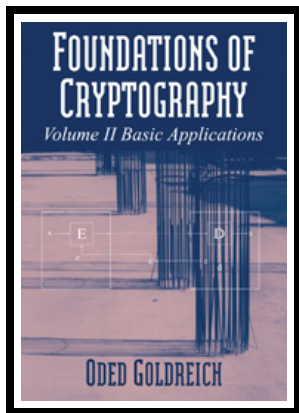
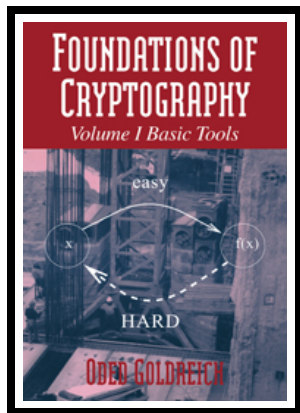
- ▶ **Abstractions:**
 - ▶ Randomized algorithms;
 - ▶ Probabilistic Turing machines.
 - ▶ Labelled Markov chains.

Randomized Computation



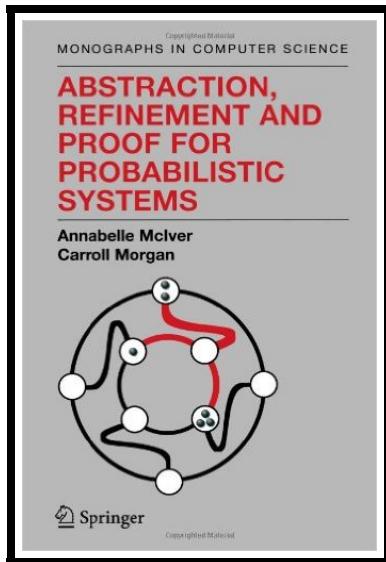
ALGORITHMICS

Randomized Computation



CRYPTOGRAPHY

Randomized Computation



PROGRAM
VERIFICATION

Higher-Order Computation

- ▶ Mainly useful in **programming**.

Higher-Order Computation

- ▶ Mainly useful in **programming**.
- ▶ Functions are **first-class citizens**:
 - ▶ They can be passed as *arguments*;
 - ▶ They can be obtained as *results*.

Higher-Order Computation

- ▶ Mainly useful in **programming**.
- ▶ Functions are **first-class citizens**:
 - ▶ They can be passed as *arguments*;
 - ▶ They can be obtained as *results*.
- ▶ **Motivations**:
 - ▶ *Modularity*;
 - ▶ *Code reuse*;
 - ▶ *Conciseness*.

Higher-Order Computation

- ▶ Mainly useful in **programming**.
- ▶ Functions are **first-class citizens**:
 - ▶ They can be passed as *arguments*;
 - ▶ They can be obtained as *results*.
- ▶ **Motivations**:
 - ▶ *Modularity*;
 - ▶ *Code reuse*;
 - ▶ *Conciseness*.
- ▶ **Example**:

```
foldr      :: (a -> b -> b) -> b -> [a] -> b
foldr f acc []      = acc
foldr f acc (x:xs) = f x (foldr f acc xs)
```

Higher-Order Computation

- ▶ Mainly useful in **programming**.
- ▶ Functions are **first-class citizens**:
 - ▶ They can be passed as *arguments*;
 - ▶ They can be obtained as *results*.
- ▶ **Motivations**:
 - ▶ *Modularity*;
 - ▶ *Code reuse*;
 - ▶ *Conciseness*.
- ▶ **Example**:

```
foldr      :: (a -> b -> b) -> b -> [a] -> b
foldr f acc []      = acc
foldr f acc (x:xs) = f x (foldr f acc xs)
```

Higher-Order Computation

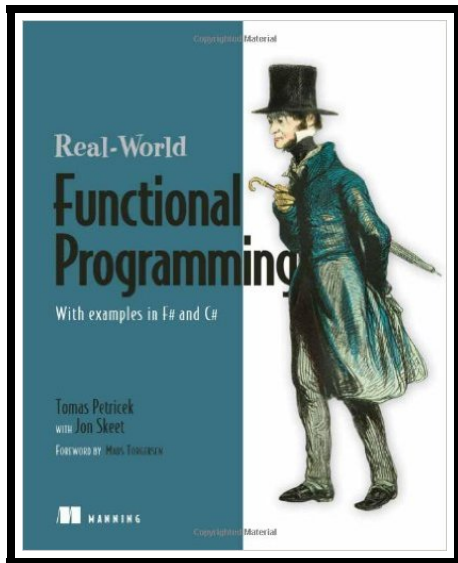
- ▶ Mainly useful in **programming**.
- ▶ Functions are **first-class citizens**:
 - ▶ They can be passed as *arguments*;
 - ▶ They can be obtained as *results*.
- ▶ **Motivations**:
 - ▶ *Modularity*;
 - ▶ *Code reuse*;
 - ▶ *Conciseness*.

- ▶ **Example:**

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f acc [] = acc
foldr f acc (x:xs) = f x (foldr f acc xs)
```

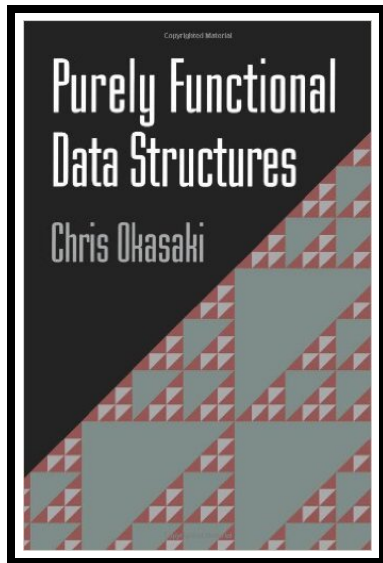
- ▶ **Models**:
 - ▶ λ -calculus

Higher-Order Computation

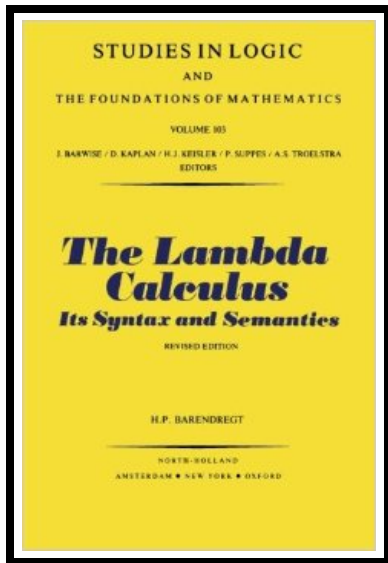


FUNCTIONAL
PROGRAMMING

Higher-Order Computation



PURELY
FUNCTIONAL
DATA STRUCTURES



λ-CALCULUS

Does it Make Sense?

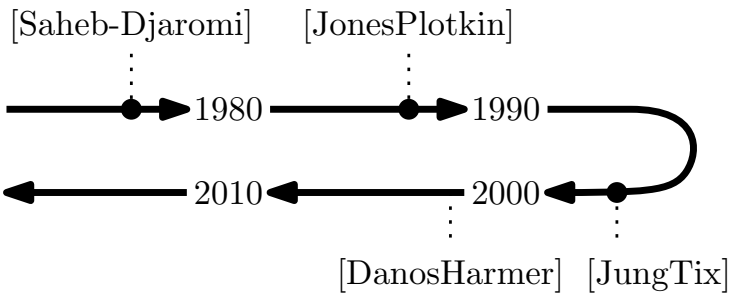
Does it Make Sense?

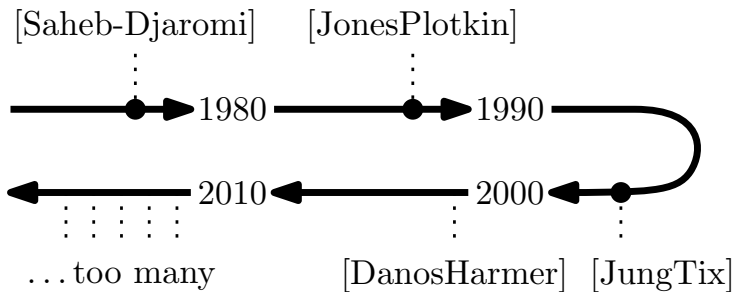
**What Kind of Metatheory
Does it Have?**

Does it Make Sense?

**What Kind of Metatheory
Does it Have?**

Applications?





Outline

Part I Relational Reasoning

Part II Bayesian Functional Programming

Part III Termination

Part I

Relational Reasoning

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M;$

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M;$
- ▶ **Values:** $V ::= \lambda x.M;$

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M;$
- ▶ **Values:** $V ::= \lambda x.M;$
- ▶ **Value Distributions:**

$$V \xrightarrow{\mathcal{D}} \mathcal{D}(V) \in \mathbb{R}_{[0,1]} \qquad \sum \mathcal{D} = \sum_V \mathcal{D}(V) \leq 1.$$

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M;$
- ▶ **Values:** $V ::= \lambda x.M;$
- ▶ **Value Distributions:**

$$V \xrightarrow{\mathcal{D}} \mathcal{D}(V) \in \mathbb{R}_{[0,1]} \qquad \sum \mathcal{D} = \sum_V \mathcal{D}(V) \leq 1.$$

- ▶ **Semantics:** $\llbracket M \rrbracket = \sup_{M \Downarrow \mathcal{D}} \mathcal{D};$

Syntax and Operational Semantics of Λ_{\oplus}

$$\begin{array}{c}
 \frac{}{M \Downarrow \emptyset} \quad \frac{}{V \Downarrow \{V^1\}} \quad \frac{M \Downarrow \mathcal{D} \quad N \Downarrow \mathcal{E}}{M \oplus N \Downarrow \frac{1}{2}\mathcal{D} + \frac{1}{2}\mathcal{E}} \\
 \\
 \frac{M \Downarrow \mathcal{K} \quad \{P[N/x] \Downarrow \mathcal{E}_P\}_{\lambda x. P \in \mathcal{S}\mathcal{K}}}{MN \Downarrow \sum_{\lambda x. P \in \mathcal{S}\mathcal{K}} \mathcal{K}(\lambda x. P) \cdot \mathcal{E}_P}
 \end{array}$$

$\sum_V \mathcal{D} = \sum_V \mathcal{D}(V) \leq 1.$

- **Semantics:** $\llbracket M \rrbracket = \sup_{M \Downarrow \mathcal{D}} \mathcal{D};$

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M$;
- ▶ **Values:** $V ::= \lambda x.M$;
- ▶ **Value Distributions:**

$$V \xrightarrow{\mathcal{D}} \mathcal{D}(V) \in \mathbb{R}_{[0,1]} \qquad \sum \mathcal{D} = \sum_V \mathcal{D}(V) \leq 1.$$

- ▶ **Semantics:** $\llbracket M \rrbracket = \sup_{M \Downarrow \mathcal{D}} \mathcal{D}$;
- ▶ **Context Equivalence:** $M \equiv N$ iff for every context C it holds that $\sum \llbracket C[M] \rrbracket = \sum \llbracket C[N] \rrbracket$.

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M$;
- ▶ **Values:** $V ::= \lambda x.M$;
- ▶ **Value Distributions:**

$$C ::= [\cdot] \mid \lambda x.C \mid CM \mid MC \mid C \oplus M \mid M \oplus C \quad (V) \leq 1.$$

- ▶ **Semantics:** $\llbracket M \rrbracket = \sup_{M \ll \mathcal{D}} \mathcal{D}$;
- ▶ **Context Equivalence:** $M \equiv N$ iff for every context C it holds that $\sum \llbracket C[M] \rrbracket = \sum \llbracket C[N] \rrbracket$.

Syntax and Operational Semantics of Λ_{\oplus}

- ▶ **Terms:** $M ::= x \mid \lambda x.M \mid MM \mid M \oplus M;$
- ▶ **Values:** $V ::= \lambda x.M;$
- ▶ **Value Distributions:**

$$V \xrightarrow{\mathcal{D}} \mathcal{D}(V) \in \mathbb{R}_{[0,1]} \qquad \sum \mathcal{D} = \sum_V \mathcal{D}(V) \leq 1.$$

- ▶ **Semantics:** $\llbracket M \rrbracket = \sup_{M \Downarrow \mathcal{D}} \mathcal{D};$
- ▶ **Context Equivalence:** $M \equiv N$ iff for every context C it holds that $\sum \llbracket C[M] \rrbracket = \sum \llbracket C[N] \rrbracket.$
- ▶ **Context Distance:**
 $\delta^C(M, N) = \sup_C \left| \sum \llbracket C[M] \rrbracket - \sum \llbracket C[N] \rrbracket \right|.$

Examples

$$I \oplus \Omega \quad \text{vs.} \quad I$$

Examples

$\lambda x.x$



$I \oplus \Omega$

vs.

I

Examples

$$\Delta\Delta = (\lambda x.xx)(\lambda x.xx)$$

$I \oplus \Omega$

vs.

I

Exam

Not Context Equivalent: $C = [\cdot]$.

Context Distance? Consider $C_n = (\lambda x. \underbrace{x \dots x}_n)[\cdot]$.

$I \oplus \Omega$ vs. I

Examples

$$I \oplus \Omega \quad \text{vs.} \quad I$$

$$I \oplus \Omega \quad \text{vs.} \quad \Omega$$

Examples

Not Context Equivalent: $C = [\cdot]$.
Context Distance? Cannot Easily Amplify.

$I \oplus \Omega$ vs. Ω

Examples

$$I \oplus \Omega \quad \text{vs.} \quad I$$

$$I \oplus \Omega \quad \text{vs.} \quad \Omega$$

$$(\lambda x. I) \oplus (\lambda x. \Omega) \quad \text{vs.} \quad \lambda x. I \oplus \Omega$$

Examples

$I \oplus \Omega$ vs. I

Not Context Equivalent in CBV: $C = (\lambda x.x(xI))[\cdot]$
Apparently Context Equivalent in CBN.

$(\lambda x.I) \oplus (\lambda x.\Omega)$ vs. $\lambda x.I \oplus \Omega$

Examples

$$I \oplus \Omega \quad \text{vs.} \quad I$$

$$I \oplus \Omega \quad \text{vs.} \quad \Omega$$

$$(\lambda x. I) \oplus (\lambda x. \Omega) \quad \text{vs.} \quad \lambda x. I \oplus \Omega$$

$$Y_1 \quad \text{vs.} \quad Y_2$$

Examples

$$I \oplus \Omega \quad \text{vs.} \quad I$$

$$I \oplus \Omega \quad \text{vs.} \quad \Omega$$

$$Y_1 M \rightarrow^* M(Y_2 M) \oplus M(Y_3 M)$$

$$Y_2 M \rightarrow^* M(Y_1 M) \oplus M(Y_3 M)$$

$$Y_3 M \rightarrow^* M(Y_1 M) \oplus M(Y_2 M)$$

$$Y_1 \quad \text{vs.} \quad Y_2$$

A Labelled Markov Chain for Λ_{\oplus}

Terms

A Labelled Markov Chain for Λ_{\oplus}

Terms

Values

A Labelled Markov Chain for Λ_{\oplus}

Terms

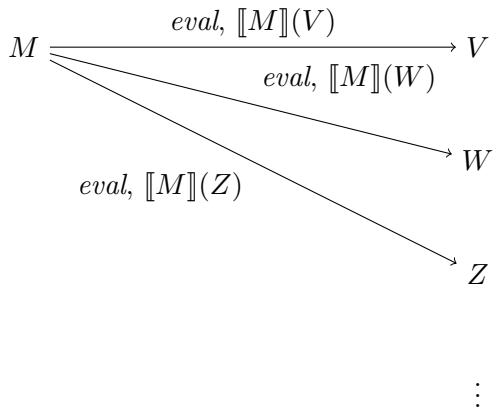
Values

M

A Labelled Markov Chain for Λ_{\oplus}

Terms

Values



A Labelled Markov Chain for Λ_{\oplus}

Terms

Values

$\lambda x.N$

A Labelled Markov Chain for Λ_{\oplus}

Terms

Values

$$N\{W/x\} \xleftarrow{W, 1} \lambda x.N$$

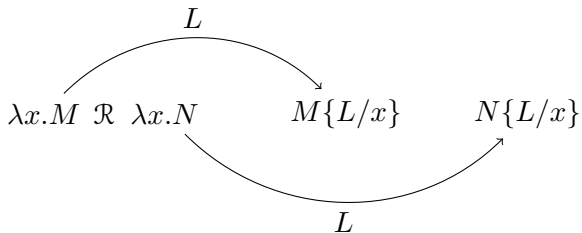
Probabilistic Applicative Bisimulation

$$\lambda x.M \mathcal{R} \lambda x.N$$

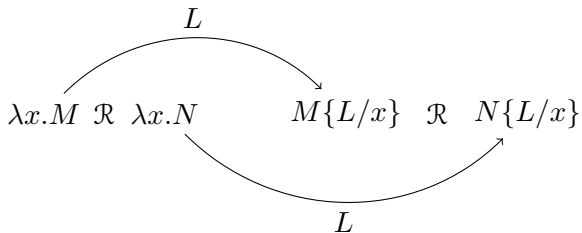
Probabilistic Applicative Bisimulation

$$\lambda x.M \mathcal{R} \lambda x.N \quad \xrightarrow{L} \quad M\{L/x\}$$

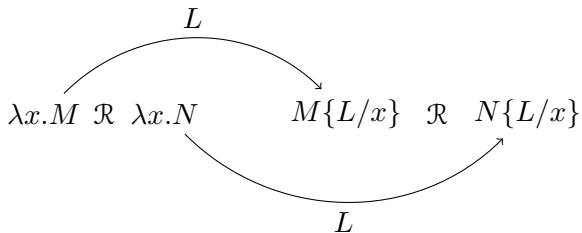
Probabilistic Applicative Bisimulation



Probabilistic Applicative Bisimulation

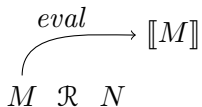
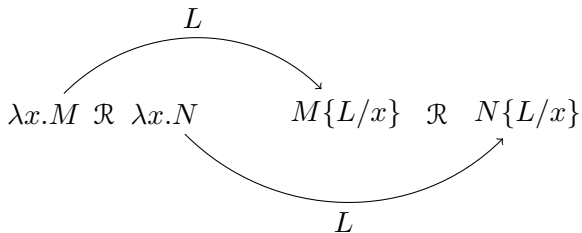


Probabilistic Applicative Bisimulation

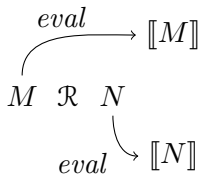
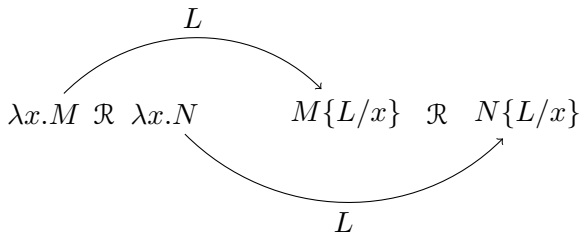


$$M \mathcal{R} N$$

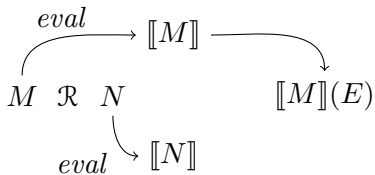
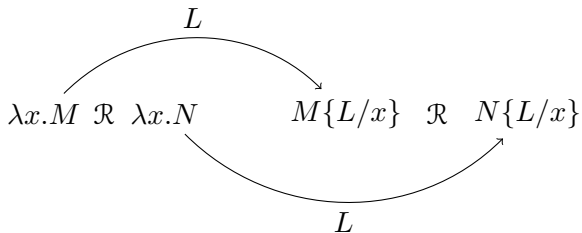
Probabilistic Applicative Bisimulation



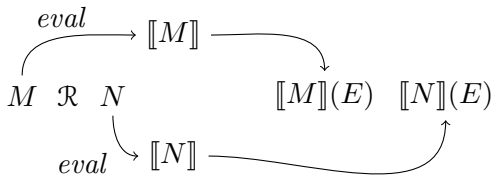
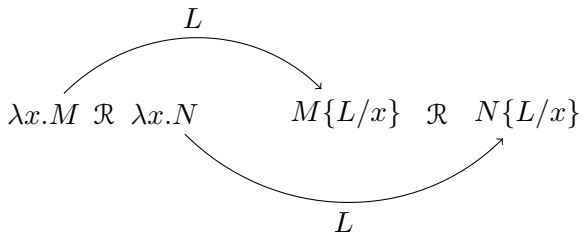
Probabilistic Applicative Bisimulation



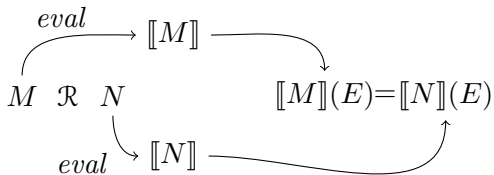
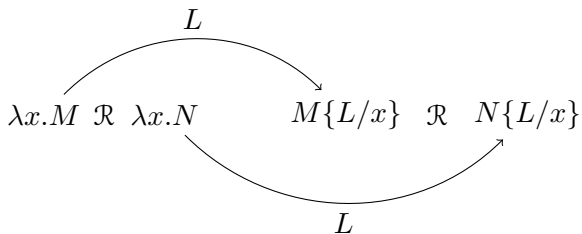
Probabilistic Applicative Bisimulation



Probabilistic Applicative Bisimulation



Probabilistic Applicative Bisimulation



Applicative Bisimilarity vs. Context Equivalence

- ▶ **Bisimilarity**: the union \sim of all bisimulation relations.
- ▶ Is it that \sim is included in \equiv ? How to prove it?
- ▶ Natural strategy: is \sim a congruence?

- ▶ If this is the case:

$$\begin{aligned}M \sim N &\implies C[M] \sim C[N] \implies \sum \llbracket C[M] \rrbracket = \sum \llbracket C[N] \rrbracket \\ &\implies M \equiv N.\end{aligned}$$

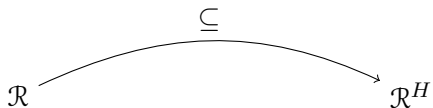
- ▶ This is a necessary sanity check anyway.
- ▶ The naïve proof by induction **fails**, due to application:
from $M \sim N$, one cannot directly conclude that $LM \sim LN$.

Howe's Technique

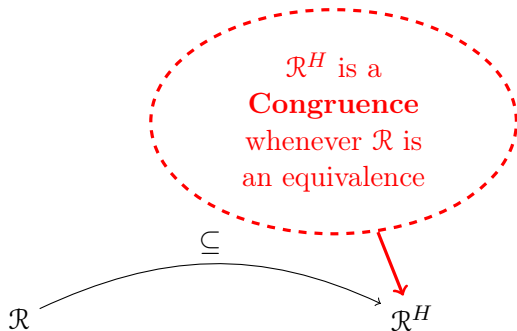
\mathcal{R}

\mathcal{R}^H

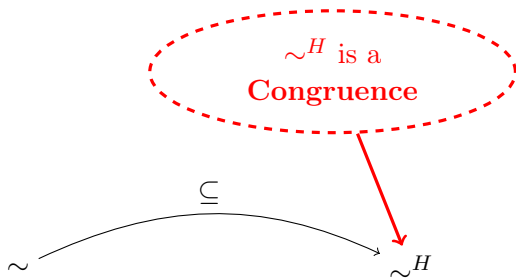
Howe's Technique



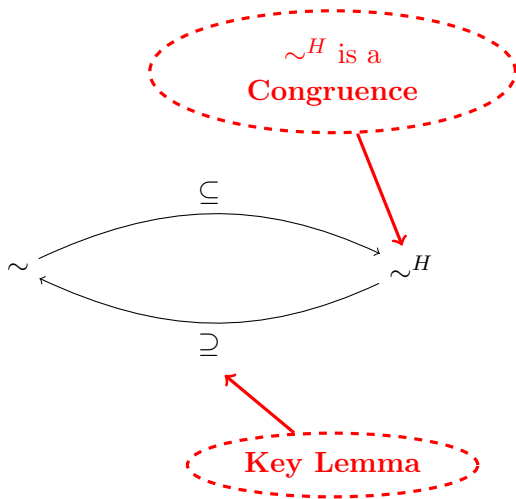
Howe's Technique



Howe's Technique



Howe's Technique



Our Neighborhood

- ▶ Λ , where we observe **convergence**

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✓
<i>CBV</i>	✓	✓

[Abramsky1990, Howe1993]

- ▶ Λ_{\oplus} with nondeterministic semantics, where we observe **convergence**, in its **may** or **must** flavors.

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✗

[Ong1993, Lassen1998]

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✓

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✓

- ▶ Counterexample for CBN: $(\lambda x.I) \oplus (\lambda x.\Omega) \not\sim \lambda x.I \oplus \Omega$
- ▶ **Where** these discrepancies come from?

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✓

- ▶ Counterexample for CBN: $(\lambda x.I) \oplus (\lambda x.\Omega) \not\approx \lambda x.I \oplus \Omega$
- ▶ **Where** these discrepancies come from?
- ▶ From **testing!**

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\sim \subseteq \equiv$	$\equiv \subseteq \sim$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✓

- ▶ Counterexample for CBN: $(\lambda x.I) \oplus (\lambda x.\Omega) \not\sim \lambda x.I \oplus \Omega$
- ▶ **Where** these discrepancies come from?
- ▶ From **testing!**
- ▶ Bisimulation can be characterized by testing equivalence as follows:

Calculus	Testing
Λ	$T ::= \omega \mid a \cdot T$
$P\Lambda_{\oplus}$	$T ::= \omega \mid a \cdot T \mid \langle T, T \rangle$
$N\Lambda_{\oplus}$	$T ::= \omega \mid a \cdot T \mid \bigwedge_{i \in I} T_i \mid \dots$

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\approx \subseteq \leq$	$\leq \subseteq \approx$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✗

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\approx \subseteq \leq$	$\leq \subseteq \approx$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✗

- ▶ Probabilistic simulation can be characterized by testing as follows:

$$T ::= \omega \mid a \cdot T \mid \langle T, T \rangle \mid T \vee T$$

The Probabilistic Case

- ▶ Λ_{\oplus} with probabilistic semantics.

	$\approx \subseteq \leq$	$\leq \subseteq \approx$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✗

- ▶ Probabilistic simulation can be characterized by testing as follows:

$$T ::= \omega \mid a \cdot T \mid \langle T, T \rangle \mid T \vee T$$

- ▶ Full abstraction can be recovered if endowing Λ_{\oplus} with parallel disjunction [CDLSV2015].

	$\approx \subseteq \leq$	$\leq \subseteq \approx$
<i>CBN</i>	✓	✗
<i>CBV</i>	✓	✓

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega;$

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.

Cor

$$\frac{}{\Gamma, x \vdash x} \quad \frac{x, \Gamma \vdash M}{\Gamma \vdash \lambda x.M} \quad \frac{\Gamma \vdash M \quad \Delta \vdash N}{\Gamma, \Delta \vdash MN} \quad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash M \oplus N}$$

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
- ▶ **Behavioural Distance** δ^b .
 - ▶ The metric analogue to bisimilarity.

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
- ▶ **Behavioural Distance** δ^b .
 - ▶ The metric analogue to bisimilarity.
- ▶ **Trace Distance** δ^t .
 - ▶ The maximum distance induced by traces, i.e., sequences of actions: $\delta^t(M, N) = \sup_{\Upsilon} |Pr(M, \Upsilon) - Pr(N, \Upsilon)|$.

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
- ▶ **Behavioural Distance** δ^b .
 - ▶ The metric analogue to bisimilarity.
- ▶ **Trace Distance** δ^t .
 - ▶ The maximum distance induced by traces, i.e., sequences of actions: $\delta^t(M, N) = \sup_{\mathbb{T}} |Pr(M, \mathbb{T}) - Pr(N, \mathbb{T})|$.
- ▶ **Soundness and Completeness Results:**

$\delta^b \leq \delta^c$	$\delta^c \leq \delta^b$	$\delta^t \leq \delta^c$	$\delta^c \leq \delta^t$
✓	✗	✓	✓

Context Distance: the Affine Case [CDL2015]

- ▶ Let us consider a simple fragment of Λ_{\oplus} , first.
- ▶ **Preterms:** $M, N ::= x \mid \lambda x.M \mid MM \mid M \oplus M \mid \Omega$;
- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
- ▶ **Behavioural Distance** δ^b .
 - ▶ The metric analogue to bisimilarity.
- ▶ **Trace Distance** δ^t .
 - ▶ The maximum distance induced by traces, i.e., sequences of actions: $\delta^t(M, N) = \sup_{\mathbb{T}} |Pr(M, \mathbb{T}) - Pr(N, \mathbb{T})|$.
- ▶ **Soundness and Completeness Results:**

$\delta^b \leq \delta^c$	$\delta^c \leq \delta^b$	$\delta^t \leq \delta^c$	$\delta^c \leq \delta^t$
✓	✗	✓	✓

- ▶ **Example:** $\delta^t(I, I \oplus \Omega) = \delta^t(I \oplus \Omega, \Omega) = \frac{1}{2}$.

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have have worked so far with induces **unsound** metrics for $\Lambda_{\oplus} \dots$

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have have worked so far with induces **unsound** metrics for Λ_{\oplus} ...
- ▶ ...because it **does not** adequately model copying.

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have have worked so far with induces **unsound** metrics for Λ_{\oplus} ...
- ▶ ...because it **does not** adequately model copying.
- ▶ **A Tuple LMC.**
 - ▶ **Preterms:**
 $M ::= x \mid \lambda x.M \mid \lambda!x.M \mid MM \mid M \oplus M \mid !M$
 - ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
 - ▶ **States:** *sequences* of terms, rather than terms.
 - ▶ **Actions** not only model parameter passing, but also *copying* of terms.

Context Distance: the General Case [CDL2016]

- ▶ **T** $\frac{! \Gamma, x \vdash x}{! \Gamma, x \vdash x}$ $\frac{! \Gamma, !x \vdash x}{! \Gamma, !x \vdash x}$ $\frac{x, \Gamma \vdash M}{\Gamma \vdash \lambda x.M}$ $\frac{!x, \Gamma \vdash M}{\Gamma \vdash \lambda !x.M}$
 ▶ **u**
 ▶ **:** $\frac{! \Gamma \vdash M}{! \Gamma \vdash !M}$ $\frac{\Gamma, !\Theta \vdash M \quad \Delta, !\Theta \vdash N}{\Gamma, \Delta, \Theta \vdash MN}$ $\frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash M \oplus N}$
 ▶ **A**

▶ **Preterms:**

$M ::= x \mid \lambda x.M \mid \lambda !x.M \mid MM \mid M \oplus M \mid !M$

▶ **Terms:** any preterm M such that $\Gamma \vdash M$.

▶ **States:** *sequences* of terms, rather than terms.

▶ **Actions** not only model parameter passing, but also *copying* of terms.

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have have worked so far with induces **unsound** metrics for $\Lambda_{\oplus} \dots$
- ▶ \dots because it **does not** adequately model copying.
- ▶ **A Tuple LMC.**
 - ▶ **Preterms:**
 $M ::= x \mid \lambda x.M \mid \lambda!x.M \mid MM \mid M \oplus M \mid !M$
 - ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
 - ▶ **States:** *sequences* of terms, rather than terms.
 - ▶ **Actions** not only model parameter passing, but also *copying* of terms.
- ▶ **Soundness and Completeness Results:**

$\delta^t \leq \delta^c$	$\delta^c \leq \delta^t$
✓	✓

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have have worked so far with induces **unsound** metrics for $\Lambda_{\oplus} \dots$
- ▶ \dots because it **does not** adequately model copying.
- ▶ **A Tuple LMC.**

- ▶ **Preterms:**

$M ::= x \mid \lambda x.M \mid \lambda !x.M \mid MM \mid M \oplus M \mid !M$

- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
 - ▶ **States:** *sequences* of terms, rather than terms.
 - ▶ **Actions** not only model parameter passing, but also *copying* of terms.
- ▶ **Soundness and Completeness Results:**

$\delta^t \leq \delta^c$	$\delta^c \leq \delta^t$
✓	✓

- ▶ **Examples:** $\delta^t(! (I \oplus \Omega), !\Omega) = \frac{1}{2}$ $\delta^t(! (I \oplus \Omega), !I) = 1$.

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have worked so far with induces **unsound** metrics for $\Lambda_{\oplus} \dots$
- ▶ \dots because it **does not** adequately model copying.
- ▶ **A Tuple LMC.**
 - ▶ **Preterms:**
 $M ::= x \mid \lambda x.M \mid \lambda !x.M \mid MM \mid M \oplus M \mid !M$
 - ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.
 - ▶ **States:** *sequences* of terms, rather than terms.
 - ▶ **Actions** not only model parameter passing, but also *copying* of terms.

- ▶ **Soundness and Completeness Results:**

$\delta^t \leq \delta^c$	$\delta^c \leq \delta^t$
✓	✓

- ▶ **Examples:** $\delta^t(! (I \oplus \Omega), !\Omega) = \frac{1}{2}$ $\delta^t(! (I \oplus \Omega), !I) = 1$.
- ▶ **Trivialisation:** the context distance *collapses* to an equivalence in *strongly normalising* fragments or in presence of *parallel disjunction*.

Context Distance: the General Case [CDL2016]

- ▶ The LMC we have worked so far with induces **unsound** metrics for $\Lambda_{\oplus} \dots$
- ▶ ... because it **does not** adequately model copying.
- ▶ **A Tuple LMC.**

- ▶ **Preterms:**

$M ::= x \mid \lambda x.M \mid \lambda !x.M \mid MM \mid M \oplus M \mid !M$

- ▶ **Terms:** any preterm M such that $\Gamma \vdash M$.

- ▶ **States:** *sequences* of terms, rather than terms.

- ▶ **Actions** not only model parameter passing, but also *copying* of terms.

- ▶ **Soundness and Completeness**

What would a *sensible* notion of distance look like?

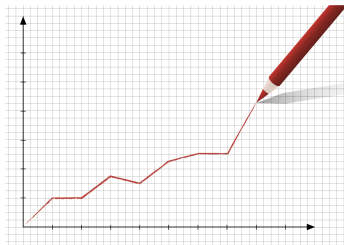
$\delta^t \leq 0$	$0 \leq 0$
✓	✓

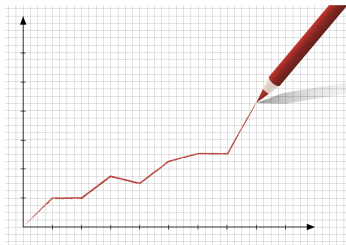
- ▶ **Examples:** $\delta^t(! (I \oplus \Omega), !\Omega) = \frac{1}{2}$ $\delta^t(! (I \oplus \Omega), !I) = 1$.
- ▶ **Trivialisation:** the context distance *collapses* to an equivalence in *strongly normalising* fragments or in presence of *parallel disjunction*.

Part II

Bayesian Functional Programming





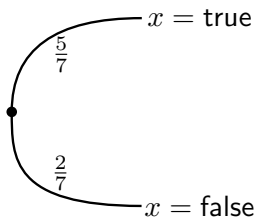


1. normalize(
2. let $x = \text{sample}(\text{bern} \left(\frac{5}{7} \right))$ in
3. let $r = \text{if } x \text{ then } 10 \text{ else } 3$ in
4. observe 4 from *poisson*(r);
5. return(x))

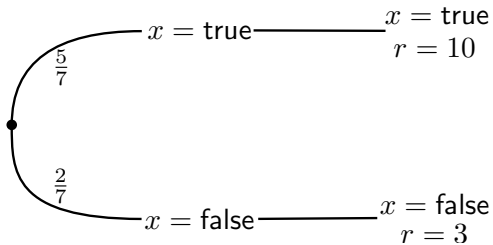
1. **normalize(**
2. let $x = \text{sample}(\text{bern}(\frac{5}{7}))$ in
3. let $r = \text{if } x \text{ then } 10 \text{ else } 3$ in
4. observe 4 from $\text{poisson}(r)$;
5. return(x)



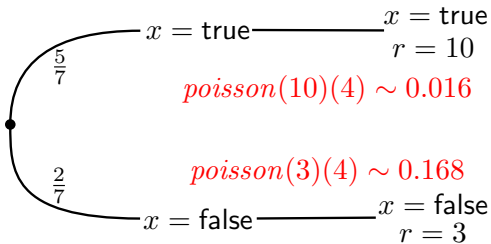
1. normalize(
2. **let $x = \text{sample}(\text{bern}(\frac{5}{7}))$ in**
3. **let $r =$ if x then 10 else 3 in**
4. **observe 4 from $\text{poisson}(r)$;**
5. **return(x)**)



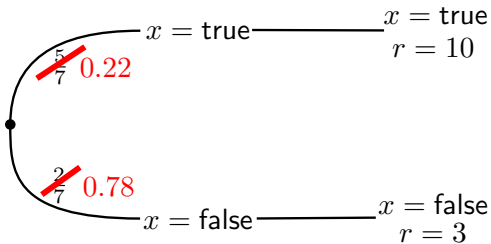
1. normalize(
2. let $x = \text{sample}(\text{bern}\left(\frac{5}{7}\right))$ in
3. **let $r = \text{if } x \text{ then } 10 \text{ else } 3$ in**
4. observe 4 from $\text{poisson}(r)$;
5. return(x)



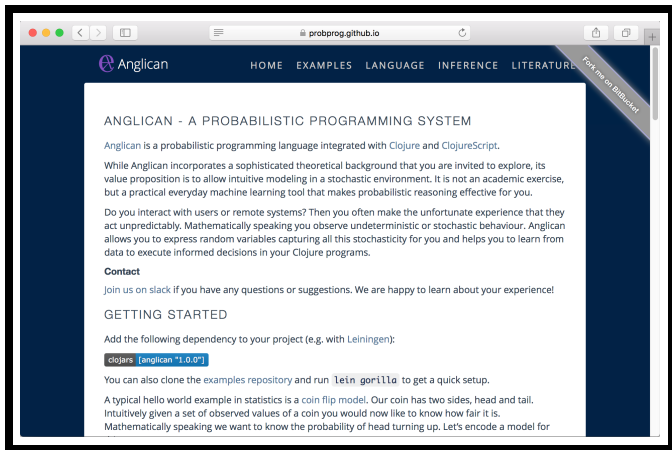
1. normalize(
2. let $x = \text{sample}(\text{bern}(\frac{5}{7}))$ in
3. let $r = \text{if } x \text{ then } 10 \text{ else } 3$ in
4. observe 4 from *poisson*(r);
5. return(x))



1. `normalize(`
2. `let $x = \text{sample}(\text{bern}(\frac{5}{7}))$ in`
3. `let $r = \text{if } x \text{ then } 10 \text{ else } 3$ in`
4. `observe 4 from $\text{poisson}(r)$;`
5. `return(x)`



Bayesian Functional Programming

A screenshot of a web browser displaying the homepage of the Anglican project. The browser's address bar shows 'probprog.github.io'. The website has a dark blue header with the Anglican logo and navigation links: HOME, EXAMPLES, LANGUAGE, INFERENCE, and LITERATURE. A diagonal banner on the right side of the header says 'Fork me on Bitbucket'. The main content area is white and contains the following text:

ANGLICAN - A PROBABILISTIC PROGRAMMING SYSTEM

Anglican is a probabilistic programming language integrated with Clojure and ClojureScript.

While Anglican incorporates a sophisticated theoretical background that you are invited to explore, its value proposition is to allow intuitive modeling in a stochastic environment. It is not an academic exercise, but a practical everyday machine learning tool that makes probabilistic reasoning effective for you.

Do you interact with users or remote systems? Then you often make the unfortunate experience that they act unpredictably. Mathematically speaking you observe undeterministic or stochastic behaviour. Anglican allows you to express random variables capturing all this stochasticity for you and helps you to learn from data to execute informed decisions in your Clojure programs.

Contact

Join us on slack if you have any questions or suggestions. We are happy to learn about your experience!

GETTING STARTED

Add the following dependency to your project (e.g. with Leiningen):

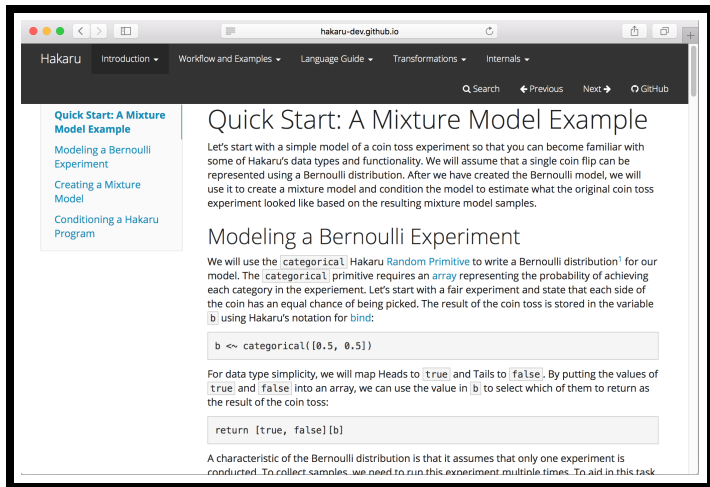
```
clojars [anglican "1.0.0"]
```

You can also clone the examples repository and run `lein gorilla` to get a quick setup.

A typical hello world example in statistics is a coin flip model. Our coin has two sides, head and tail. Intuitively given a set of observed values of a coin you would now like to know how fair it is. Mathematically speaking we want to know the probability of head turning up. Let's encode a model for

ANGLICAN

Bayesian Functional Programming



The screenshot shows a web browser window with the URL `hakaru-dev.github.io`. The navigation bar includes links for `Introduction`, `Workflow and Examples`, `Language Guide`, `Transformations`, and `Internals`. A search bar and navigation arrows are also present. The main content area is titled `Quick Start: A Mixture Model Example`. A sidebar on the left lists several articles, with the first one, `Quick Start: A Mixture Model Example`, highlighted in blue. The main text begins with an introduction to a coin toss experiment model. Below the text, there are two code blocks: the first defines a variable `b` using `categorical([0.5, 0.5])`, and the second shows a function that returns `[true, false][b]`. The text concludes with a note about the Bernoulli distribution and a partially visible sentence about running the experiment multiple times.

Hakaru Introduction Workflow and Examples Language Guide Transformations Internals

Search Previous Next GitHub

Quick Start: A Mixture Model Example

Let's start with a simple model of a coin toss experiment so that you can become familiar with some of Hakaru's data types and functionality. We will assume that a single coin flip can be represented using a Bernoulli distribution. After we have created the Bernoulli model, we will use it to create a mixture model and condition the model to estimate what the original coin toss experiment looked like based on the resulting mixture model samples.

Modeling a Bernoulli Experiment

We will use the `categorical` Hakaru [Random Primitive](#) to write a Bernoulli distribution¹ for our model. The `categorical` primitive requires an [array](#) representing the probability of achieving each category in the experiment. Let's start with a fair experiment and state that each side of the coin has an equal chance of being picked. The result of the coin toss is stored in the variable `b` using Hakaru's notation for `bind`:

```
b <- categorical([0.5, 0.5])
```

For data type simplicity, we will map Heads to `true` and Tails to `false`. By putting the values of `true` and `false` into an array, we can use the value in `b` to select which of them to return as the result of the coin toss:

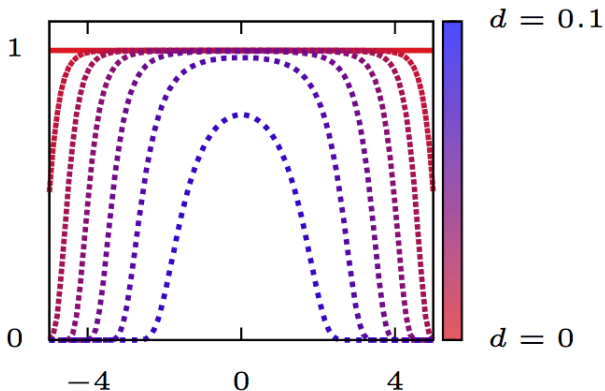
```
return [true, false][b]
```

A characteristic of the Bernoulli distribution is that it assumes that only one experiment is conducted. To collect samples, we need to run this experiment multiple times. To aid in this task

HAKARU

1. normalize(
2. let $x = \text{sample}(\text{gauss}(0, 1))$ in
4. observe d from $\text{exp}(1/f(x))$;
5. return(x)

1. normalize(
2. let $x = \text{sample}(\text{gauss}(0, 1))$ in
4. observe d from $\text{exp}(1/f(x))$;
5. return(x))



Bayesian Programming: Semantics

- ▶ Giving semantics to programming languages like Anglican or Hakaru is nontrivial:
 - ▶ Real numbers;
 - ▶ Sampling from **continuous** distributions;
 - ▶ **Conditioning**.

Bayesian Programming: Semantics

- ▶ Giving semantics to programming languages like Anglican or Hakaru is nontrivial:
 - ▶ Real numbers;
 - ▶ Sampling from **continuous** distributions;
 - ▶ **Conditioning**.
- ▶ **Key ingredients:**
 - ▶ In $M \Downarrow \mathcal{D}$, we need \mathcal{D} to be a *measure*, because the set of term is not countable anymore.

Bayesian Programming: Semantics

- ▶ Giving semantics to programming languages like Anglican or Hakaru is nontrivial:
 - ▶ Real numbers;
 - ▶ Sampling from **continuous** distributions;
 - ▶ **Conditioning**.
- ▶ **Key ingredients:**
 - ▶ In $M \Downarrow \mathcal{D}$, we need \mathcal{D} to be a *measure*, because the set of term is not countable anymore.
 - ▶ Terms must thus be equipped with the structure of a measurable space.

Bayesian Programming: Semantics

- ▶ Giving semantics to programming languages like Anglican or Hakaru is nontrivial:
 - ▶ Real numbers;
 - ▶ Sampling from **continuous** distributions;
 - ▶ **Conditioning**.
- ▶ **Key ingredients:**
 - ▶ In $M \Downarrow \mathcal{D}$, we need \mathcal{D} to be a *measure*, because the set of term is not countable anymore.
 - ▶ Terms must thus be equipped with the structure of a measurable space.

- ▶ From

$$\frac{M \Downarrow \mathcal{K} \quad \{P[N/x] \Downarrow \mathcal{E}_P\}_{\lambda x.P \in \mathcal{S}\mathcal{K}}}{MN \Downarrow \sum_{\lambda x.P \in \mathcal{S}\mathcal{K}} \mathcal{K}(\lambda x.P) \cdot \mathcal{E}_P}$$

we go to

$$\frac{M \Downarrow \mathcal{K} \quad \{P[N/x] \Downarrow \mathcal{E}_P\}_{\lambda x.P \in \mathcal{S}\mathcal{K}}}{MN \Downarrow \int \mathcal{E}_P \cdot d\mathcal{K}(\lambda x.P)}$$

Bayesian Programming: Semantics

- ▶ Giving semantics to programming languages like Anglican or Hakaru is nontrivial:
 - ▶ Real numbers;
 - ▶ Sampling from **continuous** distributions;
 - ▶ **Conditioning**.

- ▶ **Key ingredients:**

- ▶ In $M \Downarrow$ term is \dots use the set of \dots
- ▶ This Lebesgue integral does not necessarily exist.
- ▶ We must ensure that \Downarrow gives rise to a *stochastic kernel*. \dots structure of a \dots
- ▶ From \dots $\in \mathcal{S}_{\mathcal{X}}$

we go to

$$\frac{M \Downarrow \mathcal{K} \quad \{P[N/x] \Downarrow \mathcal{E}_P\}_{\lambda x. P \in \mathcal{S}_{\mathcal{X}}}}{MN \Downarrow \int \mathcal{E}_P \cdot d\mathcal{K}(\lambda x. P)}$$

Part III

Termination

The Landscape: *Type* Theory

Simple Types

$\tau ::= \iota \mid \tau \rightarrow \tau$

The Landscape: *Type Theory*

Simple Types

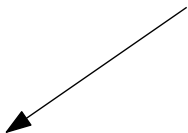
$\tau ::= \iota \quad \tau \rightarrow \tau$

- ▶ Sound for termination, in absence of recursion.
- ▶ Poor expressive power.
- ▶ Intuitionistic Logic.

The Landscape: *Type Theory*

Simple Types

$\tau ::= \iota \mid \tau \rightarrow \tau$



Polymorphic
Types

$\tau ::= \dots \mid \alpha \mid \forall \alpha. \tau$

The Landscape: *Type* Theory

- ▶ Second-order Logic.
- ▶ Very expressive, extensionally.
- ▶ Still poor, intensionally.

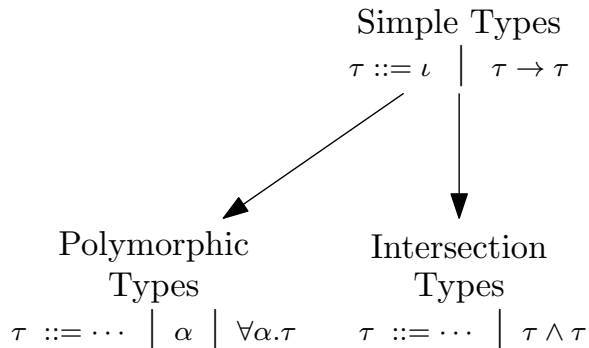
Types

$\tau \rightarrow \tau$

Polymorphic
Types

$\tau ::= \dots \mid \alpha \mid \forall \alpha. \tau$

The Landscape: *Type Theory*



The Landscape: *Type Theory*

- ▶ Motivated by Semantics.
- ▶ *Complete* for termination.
- ▶ Type inference is undecidable.

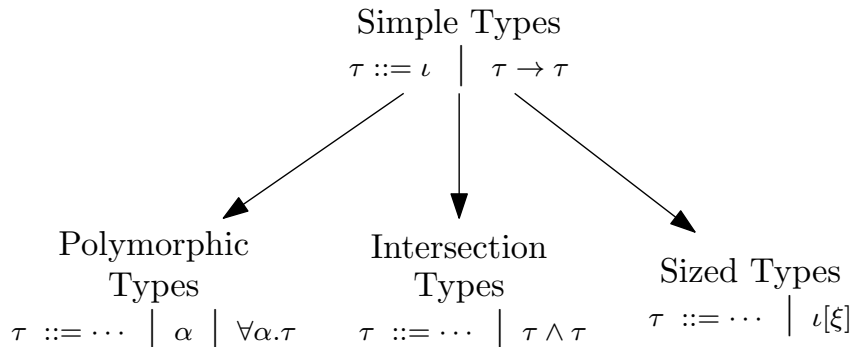
Polymorphic
Types

$\tau ::= \dots \mid \alpha \mid \forall \alpha. \tau$

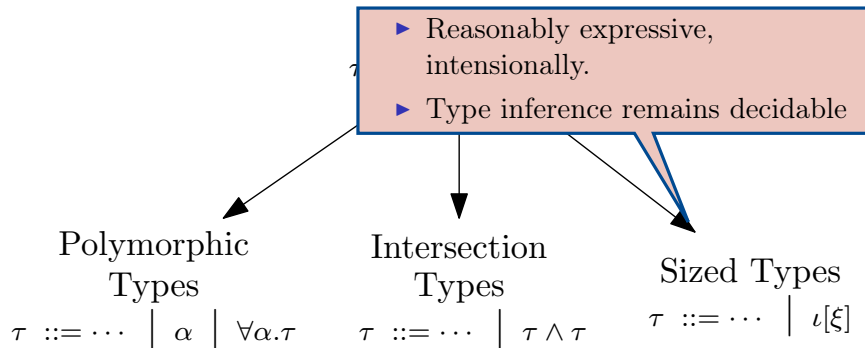
Intersection
Types

$\tau ::= \dots \mid \tau \wedge \tau$

The Landscape: *Type Theory*



The Landscape: *Type Theory*



Determinism

$$M_{\bar{s}} \rightarrow^* N_s$$

The Landscape: *Recursion* Theory

Determinism

$$M\bar{s} \rightarrow^* N_s$$

Probabilism

$$\llbracket M\bar{s} \rrbracket = \mathcal{D}_s$$

The Landscape: *Recursion* Theory

$\sum \mathcal{D}_s$ can be smaller than 1.

Determinism

$$M\bar{s} \rightarrow^* N_s$$

Probabilism

$$[[M\bar{s}]] = \mathcal{D}_s$$

The Landscape: *Recursion* Theory

Determinism

$$M\bar{s} \rightarrow^* N_s$$

Probabilism

$$\llbracket M\bar{s} \rrbracket = \mathcal{D}_s$$

Termination

$$\exists N_s \in NF$$

The Landscape: *Recursion* Theory

Undecidable;
 Σ_1^0 -complete.

sm

Probabilism

$$M\bar{s} \rightarrow^* N_s$$

$$[[M\bar{s}]] = \mathcal{D}_s$$

Termination

$$\exists N_s \in NF$$

The Landscape: *Recursion* Theory

Determinism

Probabilism

$$M\bar{s} \rightarrow^* N_s$$

$$[[M\bar{s}]] = \mathcal{D}_s$$

Termination

$$\exists N_s \in NF$$

$$\sum \mathcal{D}_s = 1$$

The Landscape: *Recursion* Theory

Almost-Sure Termination
 Π_2^0 -complete.

Dc

$$M\bar{s} \rightarrow^* N_s$$

$$[[M]] = \mathcal{D}_s$$

Termination

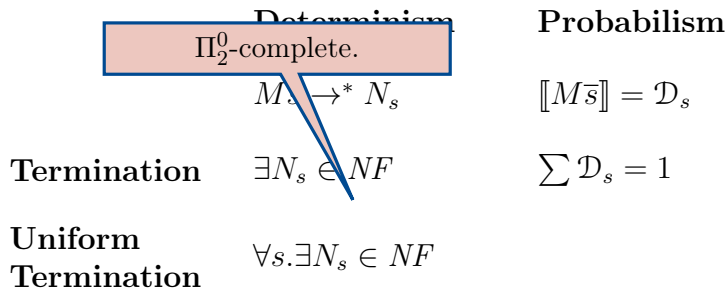
$$\exists N_s \in NF$$

$$\sum \mathcal{D}_s = 1$$

The Landscape: *Recursion* Theory

	Determinism	Probabilism
	$M\bar{s} \rightarrow^* N_s$	$\llbracket M\bar{s} \rrbracket = \mathcal{D}_s$
Termination	$\exists N_s \in NF$	$\sum \mathcal{D}_s = 1$
Uniform Termination	$\forall s. \exists N_s \in NF$	

The Landscape: *Recursion* Theory



The Landscape: *Recursion* Theory

	Determinism	Probabilism
	$M\bar{s} \rightarrow^* N_s$	$\llbracket M\bar{s} \rrbracket = \mathcal{D}_s$
Termination	$\exists N_s \in NF$	$\sum \mathcal{D}_s = 1$
Uniform Termination	$\forall s. \exists N_s \in NF$	$\forall s. \sum \mathcal{D}_s = 1$

The Landscape: *Recursion* Theory

	Determinism	Probabilism
	$M\bar{s} \rightarrow^* N_s$	$\llbracket M\bar{s} \rrbracket = \mathcal{D}_s$
Termination	$\exists N_s \in NF$	$\sum \mathcal{D}_s = 1$
Uniform Termination	$\forall s. \exists N_s \in NF$	$\forall s. \sum \mathcal{D}_s = 1$

Π_2^0 -complete.

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ For every type τ , define a set of reducible terms Red_τ .
- ▶ Prove that all reducible terms are normalizing...
- ▶ ...and that all typable terms are reducible.

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?

$$(\mathbf{fix} \ x.M)V \rightarrow M\{\mathbf{fix} \ x.M/x\}V$$

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?
- ▶ All the termination properties are **lost**, for very good reasons.

Deterministic Sized Types

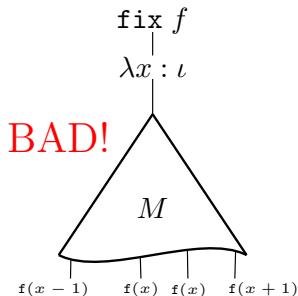
- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?
- ▶ All the termination properties are **lost**, for very good reasons.
- ▶ Is **everything** lost?

Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?
- ▶ All the termination properties are **lost**, for very good reasons.
- ▶ Is **everything** lost?
- ▶ **NO!**

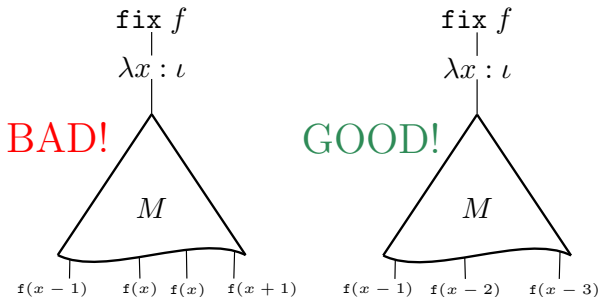
Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?
- ▶ All the termination properties are **lost**, for very good reasons.
- ▶ Is **everything** lost?
- ▶ **NO!**



Deterministic Sized Types

- ▶ Pure λ -calculus with simple types is terminating.
 - ▶ This can be proved in many ways, including by **reducibility**.
 - ▶ But useless as a programming language.
- ▶ What if we endow it with **full recursion** as a **fix** binder?
- ▶ All the termination properties are **lost**, for very good reasons.
- ▶ Is **everything** lost?
- ▶ **NO!**



Deterministic Sized Types, Technically

► **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

Deterministic Sized Types, Technically

► **Types.**

$\xi ::= a \mid \omega \mid \xi + 1;$

$\tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$



Index Terms

Deterministic Sized Types, Technically

► **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

► **Typing Fixpoints.**

$$\frac{\Gamma, x : \iota[a] \rightarrow \tau \vdash M : \iota[a + 1] \rightarrow \tau}{\Gamma \vdash \mathbf{fix} \ x.M : \iota[\xi] \rightarrow \tau}$$

Deterministic Sized Types, Technically

- ▶ **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma, x : \iota[a] \rightarrow \tau \vdash M : \iota[a + 1] \rightarrow \tau}{\Gamma \vdash \mathbf{fix} \ x.M : \iota[\xi] \rightarrow \tau}$$

- ▶ **Quite Powerful.**

- ▶ Can type many forms of structural recursion.

Deterministic Sized Types, Technically

- ▶ **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma, x : \iota[a] \rightarrow \tau \vdash M : \iota[a + 1] \rightarrow \tau}{\Gamma \vdash \mathbf{fix} x.M : \iota[\xi] \rightarrow \tau}$$

- ▶ **Quite Powerful.**

- ▶ Can type many forms of structural recursion.

- ▶ **Termination.**

- ▶ Proved by **Reducibility**.
- ▶ ... but of an indexed form.

Deterministic Sized Types, Technically

- ▶ **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

- ▶ **Typing Fixpoints.**

- ▶ Reducibility sets are of the form Red_{τ}^{θ} .
- ▶ θ is an environment for index variables.
- ▶ Proof of reducibility for $\mathbf{fix} \ x.M$ is rather delicate.

- ▶ **Q**

Can type many forms of structural recursion.

- ▶ **Termination.**

- ▶ Proved by **Reducibility**.
- ▶ ...but of an indexed form.

Deterministic Sized Types, Technically

- ▶ **Types.**

$$\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma, x : \iota[a] \rightarrow \tau \vdash M : \iota[a + 1] \rightarrow \tau}{\Gamma \vdash \mathbf{fix} x.M : \iota[\xi] \rightarrow \tau}$$

- ▶ **Quite Powerful.**

- ▶ Can type many forms of structural recursion.

- ▶ **Termination.**

- ▶ Proved by **Reducibility**.
- ▶ ...but of an indexed form.

- ▶ **Type Inference.**

- ▶ It is indeed *decidable*.
- ▶ But *nontrivial*.

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if BiasedCoin then f(x + 1) else x.
```

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
```

```
Unbiased Random Walk then f(x + 1) else x.
```

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if x > 0 then if BiasedCoin then f(x + 1) else x.
```

Unbiased Random Walk

Biased Randomn Walk

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
```

```
fix f.λx.if BiasedCoin then f(x + 1) else x.
```

► **Non-Examples:**

```
fix f.λx.if FairCoin then f(x - 1) else (f(x + 1); f(x + 1));
```

```
fix f.λx.if BiasedCoin then f(x + 1) else f(x - 1);
```

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);  
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);  
fix f.λx.if BiasedCoin then f(x + 1) else x.
```

► **Non-Examples:**

```
fix f.λx.if FairCoin then f(x - 1) else (f(x + 1); f(x + 1));  
fix f.λx.if BiasedCoin then f(x + 1) else f(x - 1);
```

Unbiased Random Walk, with **two** upward calls.

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);  
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);  
fix f.λx.if BiasedCoin then f(x + 1) else x.
```

► **Non-Examples:**

```
fix f.λx.if FairCoin then f(x - 1) else (f(x + 1); f(x + 1));  
fix f.λx.if BiasedCoin then f(x + 1) else f(x - 1);
```

Unbiased Random Walk, with **two** upward calls.

Biased Random Walk, the “wrong” way.

Probabilistic Termination

► **Examples:**

```
fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);  
fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);  
fix f.λx.if BiasedCoin then f(x + 1) else x.
```

► **Non-Examples:**

```
fix f.λx.if FairCoin then f(x - 1) else (f(x + 1); f(x + 1));  
fix f.λx.if BiasedCoin then f(x + 1) else f(x - 1);
```

► Probabilistic termination **is** thus:

- Sensitive to *the actual distribution* from which we sample.
- Sensitive to *how many recursive calls* we perform.

One-Counter Blind Markov Chains

- ▶ They are automata of the form (Q, δ) where
 - ▶ Q is a finite set of *states*.
 - ▶ $\delta : Q \rightarrow \text{Dist}(Q \times \{-1, 0, 1\})$.
- ▶ They are a very special form of One-Counter Markov Decision Processes [BBEK2011].
 - ▶ The model is fully probabilistic, there is no nondeterminism.
 - ▶ The counter value is ignored.

One-Counter Blind Markov Chains

- ▶ They are automata of the form (Q, δ) where
 - ▶ Q is a finite set of *states*.
 - ▶ $\delta : Q \rightarrow \text{Dist}(Q \times \{-1, 0, 1\})$.
- ▶ They are a very special form of One-Counter Markov Decision Processes [BBEK2011].
 - ▶ The model is fully probabilistic, there is no nondeterminism.
 - ▶ The counter value is ignored.
- ▶ The probability of reaching a configuration where the counter is 0 can be approximated arbitrarily well *in polynomial time*.

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

Every higher-order variable occurs **at most once**.

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC}(\sigma) \text{ terminates.}}{\Gamma \mid x : \sigma \vdash V : \iota[\xi] \rightarrow \tau}$$

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC}(\sigma) \text{ terminates.}}{\Gamma \mid x : \sigma \vdash V : \iota[\xi] \rightarrow \tau}$$

This is sufficient for typing:

- ▶ Unbiased random walks;
- ▶ Biased random walks.

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC}(\sigma) \text{ terminates.}}{\Gamma \mid x : \sigma \vdash V : \iota[\xi] \rightarrow \tau}$$

- ▶ **Typing Probabilistic Choice**

$$\frac{\Gamma \mid \Delta \vdash M : \tau \quad \Gamma \mid \Omega \vdash N : \rho}{\Gamma \mid \frac{1}{2}\Delta + \frac{1}{2}\Omega \vdash M \oplus N : \frac{1}{2}\tau + \frac{1}{2}\rho}$$

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

- ▶ **Typing Fixpoints.**

$$\frac{\Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC}(\sigma) \text{ terminates.}}{\Gamma \mid x : \sigma \vdash V : \iota[\xi] \rightarrow \tau}$$

- ▶ **Typing Probabilistic Choice**

$$\frac{\Gamma \mid \Delta \vdash M : \tau \quad \Gamma \mid \Omega \vdash N : \rho}{\Gamma \mid \frac{1}{2}\Delta + \frac{1}{2}\Omega \vdash M \oplus N : \frac{1}{2}\tau + \frac{1}{2}\rho}$$

- ▶ **Termination.**

- ▶ By a quantitative nontrivial refinement of reducibility.

Probabilistic Sized Types [DLGrellois2017]

- ▶ **Basic Idea:** craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.
- ▶ **Judgments.**

$$\Gamma \mid \Delta \vdash M : \mu$$

- ▶ **Typing Fixpoints.**

- ▶ Reducibility sets are now on the form $Red_{\tau}^{\theta,p}$ states.
- ▶ p stands for the *probability* of being reducible.
- ▶ Reducibility sets are continuous:

$$Red_{\tau}^{\theta,p} = \bigcup_{q < p} Red_{\tau}^{\theta,q}$$

- ▶ **Termination.**
 - ▶ By a quantitative nontrivial refinement of reducibility.

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?
- ▶ Very simple examples of normalizing terms which *cannot* be typed:

$$\Delta = \lambda x.xx \qquad \Delta(\lambda x.x).$$

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?
- ▶ Very simple examples of normalizing terms which *cannot* be typed:

$$\Delta = \lambda x.xx \qquad \Delta(\lambda x.x).$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow B \qquad A ::= \{\tau_1, \dots, \tau_n\}$$

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?
- ▶ Very simple examples of normalizing terms which *cannot* be typed:

$$\Delta = \lambda x.xx \qquad \Delta(\lambda x.x).$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow B \qquad A ::= \{\tau_1, \dots, \tau_n\}$$

- ▶ **Typing Rules: Examples**

$$\frac{\{\Gamma \vdash M : \tau_i\}_{1 \leq i \leq n}}{\Gamma \vdash M : \{\tau_1, \dots, \tau_n\}} \qquad \frac{\Gamma \vdash M : \{A \rightarrow B\} \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?
- ▶ Very simple examples of normalizing terms which *cannot* be typed:

$$\Delta = \lambda x.xx \qquad \Delta(\lambda x.x).$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow B \qquad A ::= \{\tau_1, \dots, \tau_n\}$$

- ▶ **Typing Rules: Examples**

$$\frac{\{\Gamma \vdash M : \tau_i\}_{1 \leq i \leq n}}{\Gamma \vdash M : \{\tau_1, \dots, \tau_n\}} \qquad \frac{\Gamma \vdash M : \{A \rightarrow B\} \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

- ▶ **Termination**
 - ▶ Again by reducibility.

Deterministic Intersection Types

- ▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?
- ▶ Very simple examples of normalizing terms which *cannot* be typed:

$$\Delta = \lambda x.xx \qquad \Delta(\lambda x.x).$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow B \qquad A ::= \{\tau_1, \dots, \tau_n\}$$

- ▶ **Typing Rules: Examples**

$$\frac{\{\Gamma \vdash M : \tau_i\}_{1 \leq i \leq n}}{\Gamma \vdash M : \{\tau_1, \dots, \tau_n\}} \qquad \frac{\Gamma \vdash M : \{A \rightarrow B\} \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

- ▶ **Termination**

- ▶ Again by reducibility.

- ▶ **Completeness**

- ▶ By *subject expansion*, the dual of subject reduction.

Oracle Intersection Types [BreuvarDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \mathbf{if} \textit{BitInput} \mathbf{then} M \mathbf{else} N$$

Oracle Intersection Types [BreuvarDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \text{if } \textit{BitInput} \text{ then } M \text{ else } N$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \dots, \tau_n\} \quad s \in \{0, 1\}^*$$

Oracle Intersection Types [BreuvartDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \text{if } \textit{BitInput} \text{ then } M \text{ else } N$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \dots, \tau_n\} \quad s \in \{0, 1\}^*$$

- ▶ **Typing Rules: Examples**

$$\frac{\Gamma \vdash M : s \cdot A}{\Gamma \vdash M \oplus N : 0s \cdot A} \quad \frac{\Gamma \vdash M : r \cdot \{A \rightarrow s \cdot B\} \quad \Gamma \vdash N : q \cdot A}{\Gamma \vdash MN : (rqs) \cdot B}$$

Oracle Intersection Types [BreuvartDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \text{if } \textit{BitInput} \text{ then } M \text{ else } N$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \dots, \tau_n\} \quad s \in \{0, 1\}^*$$

- ▶ **Typing Rules: Examples**

$$\frac{\Gamma \vdash M : s \cdot A}{\Gamma \vdash M \oplus N : 0s \cdot A} \quad \frac{\Gamma \vdash M : r \cdot \{A \rightarrow s \cdot B\} \quad \Gamma \vdash N : q \cdot A}{\Gamma \vdash MN : (rqs) \cdot B}$$

- ▶ **Termination and Completeness**

- ▶ Formulated in a rather *unusual* way.
- ▶ Proved as usual, but relative to a single probabilistic branch

Oracle Intersection Types [BreuvarDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \text{if } \textit{BitInput} \text{ then } M \text{ else } N$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \dots, \tau_n\} \quad s \in \{0, 1\}^*$$

- ▶ **Ty**

$$\mathbb{P}(M \downarrow) = \sum_{\vdash M : s \cdot \star} 2^{|s|}$$
$$\frac{\Gamma \vdash M : s \cdot A}{\Gamma \vdash M \oplus N : 0s \cdot A} \quad \frac{\Gamma \vdash M \quad r \cdot \{A \rightarrow s \cdot B\} \quad \Gamma \vdash N : q \cdot A}{\Gamma \vdash MN : (rqs) \cdot B}$$

- ▶ **Termination and Completeness**

- ▶ Formulated in a rather *unusual* way.
- ▶ Proved as usual, but relative to a single probabilistic branch

Oracle Intersection Types [BreuvartDL2017]

- ▶ Probabilistic choice can be seen as a form of read operation:

$$M \oplus N = \text{if } \textit{BitInput} \text{ then } M \text{ else } N$$

- ▶ **Types**

$$\tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \dots, \tau_n\} \quad s \in \{0, 1\}^*$$

- ▶ **Ty**

$$\mathbb{P}(M \downarrow) = \sum_{\vdash M : s \cdot \star} 2^{|s|}$$

This is **unavoidable**, due to recursion theory.

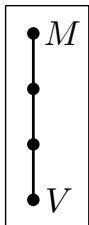
$$\frac{}{\vdash M \oplus N : 0s \cdot A}$$

$$\frac{\vdash N : q \cdot A}{\vdash MN : (rqs) \cdot B}$$

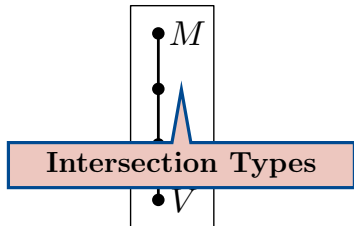
- ▶ **Termination and Completeness**

- ▶ Formulated in a rather *unusual* way.
- ▶ Proved as usual, but relative to a single probabilistic branch

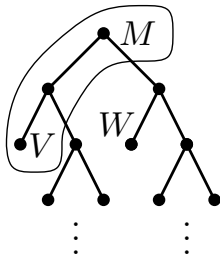
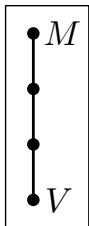
Intersection Types and Computations



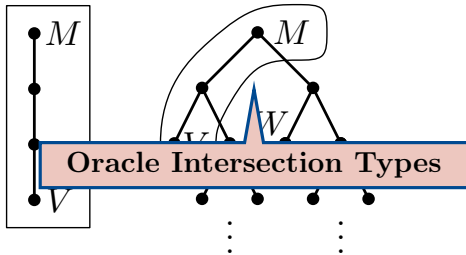
Intersection Types and Computations



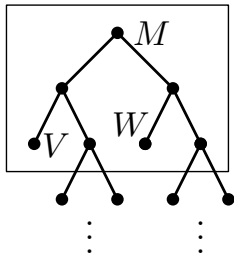
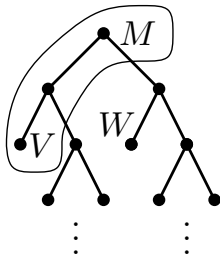
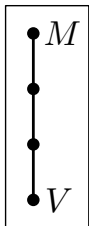
Intersection Types and Computations



Intersection Types and Computations



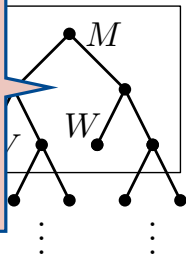
Intersection Types and Computations



Intersection Types and Computations

Monadic Intersection Types [BDL2017]

- ▶ They are a combination of oracle and sized types.
- ▶ Intersections are needed for preciseness.
- ▶ Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.



These Slides, and More...



These Slides, and More...



Questions?