A journey through the semantics of higher-order probabilistic languages, domain theory, and topology

with some parts in collaboration with
Xiaodong Jia (贾晓东),
Clément Theron

Etiolles 2023
The purpose of this talk

- I was asked the following in May 2023:
  « *Would you accept to make a long presentation on a topic related to topology in computer science (like a tutorial, or an introduction « for dummies »)?* »
  « *est-ce que tu accepterais de faire une présentation longue sur un sujet lié à la topologie en informatique (type tutoriel, ou intro "pour les nuls »)?* »

- Let me first give you an overview of my main activities in topology at LMF
Non-Hausdorff topology

Noetherian spaces

Quasi-metric spaces

Domain theory

Topological measure theory

Semantics of programming languages

Verification of infinite-state systems

Directed algebraic topology

True concurrency

The blog
Topology
What topology is, in this talk

- A topology on a set $X$ is a collection of sets, then open sets, s.t.:
  - every (arbitrary) union of open sets is open
  - every finite intersection of open sets is open
What topology is, in this talk

❖ A **topology** on a set $X$ is a collection of sets, then **open sets**, s.t.:
   — every (arbitrary) union of open sets is open
   — every finite intersection of open sets is open

❖ A **topological space** is a set $X$ with a topology
What topology is, in this talk

- A **topology** on a set $X$ is a collection of sets, then **open sets**, s.t.:
  - every (arbitrary) union of open sets is open
  - every finite intersection of open sets is open

- A **topological space** is a set $X$ with a topology

- A map $f : X \to Y$ is **continuous** iff
  for every open subset $V$ of $Y$, $f^{-1}(V)$ is open in $X$
Domain theory

- An approach to the semantics of programming languages initiated by Dana S. Scott

A type-theoretical alternative to ISWIM, CUCH, OWHY

Dana S. Scott
Carnegie-Mellon University, Pittsburgh, PA, USA, and RISC-Linz, Austria

Abstract


The paper (first written in 1969 and circulated privately) concerns the definition, axiomatization, and applications of the hereditarily iterative monad and continuous functionals generated from the integers and the Booleans (plus "undefined" elements). The system is formulated as a typed system of combinators (or as a typed λ-calculus) with a recursion operator (the least fixed-point operator), and its proof rules are contrasted to a certain extent with those of the untyped λ-calculus. For publication (1993), a new preface has been added, and many bibliographical references and comments in footnotes have been appended.

(1969, unpublished, but widely circulated - 1st published 1993)
Domain theory

- An approach to the semantics of programming languages initiated by Dana S. Scott

A type-theoretical alternative to ISWIM, CUCH, OWHY

Dana S. Scott
Carnegie-Mellon University, Pittsburgh, PA, USA, and RISC-Linz, Austria

Abstract
The paper (first written in 1969 and circulated privately) concerns the definition, axiomatization, and applications of the hereditarily monotone continuous functions generated from the integers and the Boolean algebra (plus "undefined" elements). The system is formulated as a typed system of combinators (or as a typed λ-calculus) with a recursion operator (the least fixed-point operator), and its proof rules are contrasted to a certain extent with those of the untyped λ-calculus. For publication (1993), a new preface has been added, and many bibliographical references and comments in footnotes have been appended.

- based on partial orders,
- with some particular topology

(1969, unpublished, but widely circulated - 1st published 1993)
Dcpos, partial values, and the order of information

- A dcpo (directed-complete partial order)
  is a set of «partial values»
    — total values among them
      represent terminated computations
A dcpo (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

Partially ordered:
$x \leq y$ means « $y$ contains more information, is more precise than $x$ »
Dcpos, partial values, and the order of information

❖ A dcpo (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations

❖ partially ordered:
  $x \leq y$ means « $y$ contains more information, is more precise than $x$ »

❖ with a notion of limit (=suprema) of increasing chains of partial values giving the « value at infinity »
Dcpos, partial values, and the order of information

- A **dcpo** (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

- **partially ordered:**
  \( x \leq y \) means « \( y \) contains **more information**, is more precise than \( x \) »

- with a notion of **limit** (=suprema) of increasing chains of partial values giving the « value at infinity »
Dcpos, partial values, and the order of information

- **A dcpo** (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

- **Partially ordered:**
  \( x \leq y \) means « \( y \) contains more information, is more precise than \( x \) »

- **With a notion of limit** (=suprema) of increasing chains of partial values giving the « value at infinity ».
Dcpos, partial values, and the order of information

- A dcpo (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

- partially ordered: $x \leq y$ means « $y$ contains more information, is more precise than $x$ »

- with a notion of limit (=suprema) of increasing chains of partial values giving the « value at infinity »
Dcpos, partial values, and the order of information

- A **dcpo** (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.
- **Partially ordered:** $x \leq y$ means « $y$ contains more information, is more precise than $x$ »
- With a notion of **limit** (=suprema) of increasing chains of partial values giving the « value at infinity »
A dcpo (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

- **partially ordered:**
  \( x \leq y \) means « \( y \) contains **more information**, is more precise than \( x \) »

- with a notion of **limit** (=suprema) of increasing chains of partial values giving the « value at infinity »
A **dcpo** (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations

**partially ordered:**

\[ x \leq y \text{ means « } y \text{ contains more information, is more precise than } x \text{ »} \]

**with a notion of limit (=suprema) of increasing chains of partial values giving the « value at infinity »**
Dcpos, partial values, and the order of information

- A **dcpo** (= directed-complete partial order) is a set of « partial values » — total values among them represent terminated computations.

- **Partially ordered**: 
  - $x \leq y$ means « $y$ contains more information, is more precise than $x$ »

- With a notion of **limit** (=suprema) of increasing chains of partial values giving the « value at infinity ».

Well, directed families, really:

$D$ is **directed** iff $\neq \emptyset$ and every pair of elements of $D$ has an upper bound in $D$.
Scott-continuous maps, least fixed points

- A map \( f : X \rightarrow Y \) between dcpos is **Scott-continuous** iff:
  - monotonic: \( x \leq x' \implies f(x) \leq f(x') \)
  - preserves directed suprema: \( \sup_i f(x_i) = f(\sup_i x_i) \)
Scott-continuous maps, least fixed points

- A map $f : X \to Y$ between dcpo is **Scott-continuous** iff:
  - monotonic: $x \leq x' \implies f(x) \leq f(x')$
  - preserves directed suprema: $\sup \uparrow f(x_i) = f(\sup \uparrow x_i)$

- **Theorem 1.** On a pointed dcpo $X$ (one with a least element $\bot$),
every Scott-continuous map $f : X \to X$ has a **least fixed point**.

Will serve to interpret **recursion**.
Scott-continuous maps, least fixed points

- A map \( f : X \to Y \) between dcpos is **Scott-continuous** iff:
  - monotonic: \( x \leq x' \implies f(x) \leq f(x') \)
  - preserves directed suprema: \( \sup_i f(x_i) = f(\sup_i x_i) \)

- **Theorem 1.** On a pointed dcpo \( X \) (one with a least element \( \bot \)),
every Scott-continuous map \( f : X \to X \) has a **least fixed point**.
Scott-continuous maps, least fixed points

- A map $f : X \rightarrow Y$ between dcpos is **Scott-continuous** iff:
  - monotonic: $x \leq x' \implies f(x) \leq f(x')$
  - preserves directed suprema: $\sup_i f(x_i) = f(\sup_i x_i)$

**Theorem 1.** On a pointed dcpo $X$ (one with a least element $\bot$), every Scott-continuous map $f : X \rightarrow X$ has a least fixed point.

**Proof.** Let $x \triangleq \sup_{n \in \mathbb{N}} f^n(\bot)$. Since $f$ preserves directed suprema, $f(x) = x$. That $x$ is least is left as an exercise. □
Scott-continuous maps, least fixed points

- A map $f : X \to Y$ between dcpos is **Scott-continuous** iff:
  - monotonic: $x \leq x' \implies f(x) \leq f(x')$
  - preserves directed suprema: $\sup \uparrow_i f(x_i) = f(\sup \uparrow_i x_i)$

- **Theorem 1.** On a pointed dcpo $X$ (one with a least element $\bot$), every Scott-continuous map $f : X \to X$ has a **least fixed point**.

- **Proof.** Let $x \overset{=} = \sup \uparrow_n f^n(\bot)$. Since $f$ preserves directed suprema, $f(x) = x$. That $x$ is least is left as an exercise. □

- **Theorem 2.** $[X \to Y] \overset{=} = \{f : X \to Y \text{ Scott-continuous}\}$ is a dcpo, where $f \leq g$ iff for every $x \in X, f(x) \leq g(x)$; it is pointed if $Y$ is.
Consider the higher-order, functional programming language PCF [Plotkin 77]

\[ M, N, P, \ldots ::= x, y, z, \ldots \]

- variables
- \[ MN \] application
- \[ \lambda x_\sigma . M \] abstraction
- \[ \text{rec}(M) \] recursion
- \[ 0 | 1 | 2 | \ldots \] natural numbers
- \[ s(M) \] successor
- \[ p(M) \] predecessor
- \[ \text{if } M = 0 \text{ then } N \text{ else } P \] conditional
Consider the higher-order, functional programming language PCF [Plotkin 77]

\[ M, N, P, \ldots ::= x, y, z, \ldots \]

- variables
- application
- abstraction
- recursion
- natural numbers
- successor
- predecessor
- conditional

(simply-typed) lambda-calculus
Consider the higher-order, functional programming language PCF [Plotkin 77]

\[ M, N, P, \ldots ::= x, y, z, \ldots \]

| \( MN \) | application
| \( \lambda x_\sigma . M \) | abstraction
| \( \text{rec}(M) \) | recursion
| 0 | 1 | 2 | \ldots | natural numbers
| \( s(M) \) | successor
| \( p(M) \) | predecessor
| \( \text{if } M = 0 \text{ then } N \text{ else } P \) | conditional
Consider the higher-order, functional programming language PCF [Plotkin 77]

\[ M, N, P, \ldots ::= x, y, z, \ldots \]

\[ | MN \quad \text{application} \]

\[ | \lambda x_\sigma . M \quad \text{abstraction} \]

\[ | \operatorname{rec}(M) \quad \text{recursion} \]

\[ | 0 \mid 1 \mid 2 \mid \ldots \quad \text{natural numbers} \]

\[ | s(M) \quad \text{successor} \]

\[ | p(M) \quad \text{predecessor} \]

\[ | \text{if } M = 0 \text{ then } N \text{ else } P \quad \text{conditional} \]
Types

❖ PCF terms are typed: \( \sigma, \tau, \ldots ::= \text{nat} \mid \sigma \to \tau \)
Types

- PCF terms are **typed**: \( \sigma, \tau, \ldots ::= \text{nat} | \sigma \rightarrow \tau \)

- Semantics of types: \([\tau]\) will be a pointed dcpo
Types

❖ PCF terms are typed: \( \sigma, \tau, \ldots ::= \text{nat} \mid \sigma \to \tau \)

❖ Semantics of types: \( \llbracket \tau \rrbracket \) will be a pointed dcpo

❖ \( \llbracket \text{nat} \rrbracket \triangleq \mathbb{N}_{\perp} \)
— add a fresh \( \perp \), representing non-termination
Types

- PCF terms are typed: \( \sigma, \tau, \ldots ::= \text{nat} \mid \sigma \to \tau \)

- Semantics of types: \( \llbracket \tau \rrbracket \) will be a pointed dcpo
  - \( \llbracket \text{nat} \rrbracket \triangleq \mathbb{N}_\perp \)
    - add a fresh \( \perp \), representing non-termination
  - \( \llbracket \sigma \to \tau \rrbracket \triangleq \llbracket [\sigma] \to [\tau] \rrbracket \)
    - space of Scott-continuous maps from \( \llbracket [\sigma] \rrbracket \) to \( \llbracket [\tau] \rrbracket \)
A denotational semantics for PCF

- Design (denotational) semantics \([M]\) of terms \(M : \tau\) so that \([M]_\rho \in [\tau]\)
  for every environment \(\rho\) mapping variables to values (of the right types)
A denotational semantics for PCF

- Design (denotational) semantics $[M]$ of terms $M : \tau$ so that $[M] \rho \in \llbracket \tau \rrbracket$ for every environment $\rho$ mapping variables to values (of the right types)

On nat:
- $[0] \rho \triangleq 0, \ [1] \rho \triangleq 1$, etc.
- $[s(M)] \rho \triangleq [M] \rho + 1$,
- $[p(M)] \rho \triangleq [M] \rho - 1$ if $[M] \rho \neq 0$, ⊥ otherwise
- $[\text{if } M = 0 \text{ then } N \text{ else } P] \rho \triangleq$
  - $[N] \rho$ if $[M] \rho = 0$
  - $[P] \rho$ if $[M] \rho \neq 0$, ⊥
  - ⊥ if $[M] \rho = \bot$
A denotational semantics for PCF

- Design (denotational) semantics $\llbracket M \rrbracket$ of terms $M : \tau$ so that $\llbracket M \rrbracket_\rho \in \llbracket \tau \rrbracket$ for every environment $\rho$ mapping variables to values (of the right types)

On nat:
- $\llbracket 0 \rrbracket_\rho \triangleq 0$, $\llbracket 1 \rrbracket_\rho \triangleq 1$, etc.
- $\llbracket s(M) \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho + 1$,
  $\llbracket p(M) \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho - 1$ if $\llbracket M \rrbracket_\rho \neq 0$, ⊥ otherwise
- $\llbracket \text{if } M = 0 \text{ then } N \text{ else } P \rrbracket_\rho \triangleq$
  $\llbracket N \rrbracket_\rho$ if $\llbracket M \rrbracket_\rho = 0$
  $\llbracket P \rrbracket_\rho$ if $\llbracket M \rrbracket_\rho \neq 0$, ⊥
  ⊥ if $\llbracket M \rrbracket_\rho = \perp$

Lambda-calculus:
- $\llbracket x \rrbracket_\rho \triangleq \rho(x)$
- $\llbracket MN \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho(\llbracket N \rrbracket_\rho)$
- $\llbracket \lambda x . M \rrbracket_\rho \triangleq (V \mapsto \llbracket M \rrbracket_\rho[x \mapsto V])$
- $\llbracket \text{rec}(M) \rrbracket_\rho \triangleq \text{lfp}(\llbracket M \rrbracket_\rho)$
A denotational semantics for PCF

- Design (denotational) semantics $\llbracket M \rrbracket$ of terms $M : \tau$ so that $\llbracket M \rrbracket_\rho \in \llbracket \tau \rrbracket$ for every environment $\rho$ mapping variables to values (of the right types)

On nat:

- $\llbracket 0 \rrbracket_\rho \triangleq 0$, $\llbracket 1 \rrbracket_\rho \triangleq 1$, etc.
- $\llbracket s(M) \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho + 1$,
  $\llbracket p(M) \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho - 1$ if $\llbracket M \rrbracket_\rho \neq 0$, ⊥ otherwise
- $\llbracket \text{if } M = 0 \text{ then } N \text{ else } P \rrbracket_\rho \triangleq$
  $\llbracket N \rrbracket_\rho$ if $\llbracket M \rrbracket_\rho = 0$
  $\llbracket P \rrbracket_\rho$ if $\llbracket M \rrbracket_\rho \neq 0$, ⊥
  ⊥ if $\llbracket M \rrbracket_\rho = \bot$

Lambda-calculus:

- $\llbracket x \rrbracket_\rho \triangleq \rho(x)$
- $\llbracket MN \rrbracket_\rho \triangleq \llbracket M \rrbracket_\rho(\llbracket N \rrbracket_\rho)$
- $\llbracket \lambda x . M \rrbracket_\rho \triangleq (V \mapsto \llbracket M \rrbracket(\rho[x \mapsto V]))$
- $\llbracket \text{rec}(M) \rrbracket_\rho \triangleq \text{lfp}(\llbracket M \rrbracket_\rho)$

Theorem 1. On a pointed dcpo $X$, every Scott-continuous map $f : X \to X$ has a least fixed point.
A denotational semantics for PCF

- Design (denotational) semantics $[M]$ of terms $M : \tau$ so that $[M] \rho \in [\tau]$ for every environment $\rho$ mapping variables to values (of the right types)

On nat:
- $[0] \rho = 0$, $[1] \rho = 1$, etc.
- $[s(M)] \rho = [M] \rho + 1$,
  $[p(M)] \rho = [M] \rho - 1$ if $[M] \rho \neq 0$, $\bot$ otherwise
- $[\text{if } M = 0 \text{ then } N \text{ else } P] \rho =$
  $[N] \rho$ if $[M] \rho = 0$
  $[P] \rho$ if $[M] \rho \neq 0$, $\bot$
  $\bot$ if $[M] \rho = \bot$

Lambda-calculus:
- $[x] \rho = \rho(x)$
- $[MN] \rho = [M] \rho([N] \rho)$
- $[\lambda x . M] \rho = (V \mapsto [M] \rho([x \mapsto V]))$
- $[\text{rec}(M)] \rho = \text{lfp}([M] \rho)$

Theorem 1. On a pointed dcpo $X$, every Scott-continuous map $f : X \to X$ has a least fixed point.

- Expressions have transparent semantics (functions are functions, application is application, etc.)
- compositional semantics: $[M] \rho$ defined from the semantics of immediate subterms of $M$

- No execution mechanism involved
An operational semantics for PCF

- An abstract machine (à la Krivine) = a transition relation between configurations \( C, M \)

Contexts \( C ::= _- | C[_N] | C[s(_)] | C[p(_)] | C[if _ = 0 then N else P] \)
An operational semantics for PCF

- An abstract machine (à la Krivine) = a transition relation between configurations $C, M$

**Contexts**

\[ C ::= _ \mid C[_N] \mid C[s(_)] \mid C[p(_)] \mid C[if \_ = 0 \ then \ N \ else \ P] \]

**Exploration rules** (looking for redexes)

\[
\begin{align*}
C, MN &\rightarrow C[_N], M \\
C, s(M) &\rightarrow C[s(_)], M \\
C, p(M) &\rightarrow C[p(_)], M \\
C, \text{if } M = 0 \ \text{then } N \ \text{else } P &\rightarrow C[\text{if } \_ = 0 \ \text{then } N \ \text{else } P], M
\end{align*}
\]
An operational semantics for PCF

- An abstract machine (à la Krivine) = a transition relation between configurations $C, M$

**Contexts** $C ::= _ \mid C[_N] \mid C[s(_)] \mid C[p(_)] \mid C[if \_ = 0 then \_ else \_ P]$  

**Exploration rules** (looking for redexes)

- $C, MN \rightarrow C[_N], M$
- $C, s(M) \rightarrow C[s(_)], M$
- $C, p(M) \rightarrow C[p(_)], M$
- $C, if M = 0 then N else P \rightarrow C[if \_ = 0 then \_ N else \_ P], M$

**Computation rules**

- $C[_N], \lambda x. M \rightarrow C, M[x := N]$
- $C[s(_)], n \rightarrow C, n + 1$
- $C[p(_)], n + 1 \rightarrow C, n$
- $C[if \_ = 0 then \_ N else \_ P], 0 \rightarrow C, N$
- $C[if \_ = 0 then \_ N else \_ P], n + 1 \rightarrow C, P$
- $C, \text{rec}(M) \rightarrow C, M(\text{rec}(M))$
... i.e., an interpreter
... i.e., an interpreter

type term = V of string (* variables *)
  | A of term * term (* applications *)
  | L of string * term (* abstractions *)
  | R of term (* rec(M) *)
  | C of int (* 0, 1, 2, ... *)
  | S of term (* s(M) *)
  | P of term (* p(M) *)
  | I of term * term * term (* if *)

type ctx = Ec (* _) 
  | Ac of ctx * term (* C[_N] *)
  | Sc of ctx (* C[s(_)] *)
  | Pc of ctx (* C[p(_)] *)
  | Ic of ctx * term * term (* C[if...] *)
... i.e., an interpreter

```ocaml
let rec exec (c:ctx) (t:term) =
  match t with
  (* exploration *)
  | A(m,n) -> exec (Ac(c,n),m)
  | S(m) -> exec (Sc(c),m)
  | P(m) -> exec (Pc(c),m)
  | I(m,n,p) -> exec (Ic(c,n,p),m)
  (* computation *)
  | _ -> compute c t
```

```
type term =
  V of string (* variables *)
  | A of term * term (* applications *)
  | L of string * term (* abstractions *)
  | R of term (* rec(M) *)
  | C of int (* 0, 1, 2, ... *)
  | S of term (* s(M) *)
  | P of term (* p(M) *)
  | I of term * term * term (* if *)

let rec exec (c:ctx) (t:term) =
  match t with
  (* exploration *)
  | A(m,n) -> exec (Ac(c,n),m)
  | S(m) -> exec (Sc(c),m)
  | P(m) -> exec (Pc(c),m)
  | I(m,n,p) -> exec (Ic(c,n,p),m)
  (* computation *)
  | _ -> compute c t
```
... i.e., an interpreter

\[
\text{type term} = \text{V of string} \\
| \text{A of term} * \text{term} \\
| \text{L of string} * \text{term} \\
| \text{R of term} \\
| \text{C of int} \\
| \text{S of term} \\
| \text{P of term} \\
| \text{I of term} * \text{term} * \text{term}
\]

(* variables *)

(* applications *)

(* abstractions *)

(* rec(M) *)

(* 0, 1, 2, ... *)

(* s(M) *)

(* p(M) *)

(* if *)

\[
\text{type ctx} = \text{Ec} \\
| \text{Ac of ctx} * \text{term} \\
| \text{Sc of ctx} \\
| \text{Pc of ctx} \\
| \text{Ic of ctx} * \text{term} * \text{term}
\]

(* _ *)

(* C[\_N] *)

(* C[s(\_)] *)

(* C[p(\_)] *)

(* C[if... \_] *)

let rec exec (c:ctx) (t:term) =
match t with
(* exploration *)
| \text{A}(m,n) -> exec (Ac(c,n),m) \\
| \text{S}(m) -> exec (Sc(c),m) \\
| \text{P}(m) -> exec (Pc(c),m) \\
| \text{I}(m,n,p) -> exec (Ic(c,n,p),m) \\
(* computation *)
| _ -> compute c t

\[
C,\text{MN} \rightarrow C[\_N],M \\
C,\text{s}(M) \rightarrow C[\text{s}(\_)],M \\
C,\text{p}(M) \rightarrow C[\text{p}(\_)],M
\]

C, if M = 0 then N else P \rightarrow C[\text{if}_ = 0 \text{ then N else P}],M

and compute (c:ctx) (t:term) =
match c, t with
| \text{Ac}(c',n), \text{L}(x,m) -> exec c' (subst m x n) \\
| \text{Sc}(c'), \text{I}(i) -> exec c' (I(i+1)) \\
| \text{Pc}(c'), \text{I}(i) when i>0 -> exec c' (I(i-1)) \\
| \text{Ic}(c',n,p), \text{I}(0) -> exec c' n \\
| \text{Ic}(c',n,p), \text{I}(i) when i>0 -> exec c' p \\
| _, \text{R}(m) -> exec c (\text{A}(m,t)) \\
| Ec, result -> result (* finished! *)
| _, _ -> failwith « stumped »

\[
C[\_N],i.x,M \rightarrow C,M[x := N] \\
C[\text{s}(\_)],n \rightarrow C,n+1 \\
C[\text{p}(\_)],n+1 \rightarrow C,n \\
C[\text{if}_ = 0 \text{ then N else P}],0 \rightarrow C,N \\
C[\text{if}_ = 0 \text{ then N else P}],n+1 \rightarrow C,P
\]

C, if M = 0 then N else P \rightarrow C,M[\text{if}_ = 0 \text{ then N else P}],M
... which really runs

goubault@macbook-pro-de-jean Topics2023 % gimml
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "pcf.ml"
> •
... which really runs

```
goubault@macbook-pro-de-jean Topics2023 % gimml
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "pcf.ml"
> 
```

\[
\begin{array}{l}
\text{Plus } \equiv \text{rec}(\lambda p. \lambda m. \lambda n. \\
\quad \text{if } n = 0 \text{ then } m \\
\quad \text{else } \text{plus } m (p(n))) \\

\text{Times } \equiv \text{rec}(\lambda n. \lambda m. \lambda p. \\
\quad \text{if } n = 0 \text{ then } 0 \\
\quad \text{else } \text{Plus } m \\
\quad \quad \quad \quad \text{(times } m (p(n)))) \\

\text{Fact } \equiv \text{rec}(\lambda n. \lambda p. \\
\quad \text{if } n = 0 \text{ then } 1 \\
\quad \text{else } \text{Times } n (p(n))) \\
\end{array}
\]
... which really runs

```
goubault@macbook-pro-de-jean Topics2023 % gimml
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "pcf.ml"
> 
> Plus ≜ rec(λplus . λm . λn .
  if n = 0 then m
  else plus m (p(n)))

> Times ≜ rec(λtimes . λm . λn .
  if n = 0 then 0
  else Plus m
    (times m (p(n))))

> Fact ≜ rec(λfact . λn .
  if n = 0 then 1
  else Times n (fact (p(n))))
```

```
The two semantics are related

- **Theorem (soundness).** If $C, M \rightarrow^* C', M'$ then $\llbracket C[M] \rrbracket_\rho = \llbracket C'[M'] \rrbracket_\rho$
The two semantics are related

- **Theorem (soundness).** If $C, M \rightarrow^* C', M'$ then $\llbracket C[M] \rrbracket_\rho = \llbracket C'[M'] \rrbracket_\rho$

- In particular, if $\_ , M \rightarrow^* \_ , n (n \in \mathbb{N})$, then $\llbracket M \rrbracket_\rho = n$
The two semantics are related

- **Theorem (soundness).** If $C, M \rightarrow^* C', M'$ then $\llbracket C[M] \rrbracket_\rho = \llbracket C'[M'] \rrbracket_\rho$

- In particular, if $\_ , M \rightarrow^* \_ , n \ (n \in \mathbb{N})$, then $\llbracket M \rrbracket_\rho = n$

- **Theorem (adequacy).** If $\llbracket M \rrbracket_\rho = n \in \mathbb{N} \ (\neq \bot)$, then the machine **terminates**: $\_ , M \rightarrow^* \_ , n$
The two semantics are related

- **Theorem (soundness).** If $C, M \rightarrow^* C', M'$ then $\llbracket C[M]\rrbracket_\rho = \llbracket C'[M']\rrbracket_\rho$

- In particular, if $\_ , M \rightarrow^* \_ , n \ (n \in \mathbb{N})$, then $\llbracket M\rrbracket_\rho = n$

- **Theorem (adequacy).** If $\llbracket M\rrbracket_\rho = n \in \mathbb{N} \ (\neq \bot)$, then the machine **terminates**: $\_ , M \rightarrow^* \_ , n$

- That is trickier to prove, and uses **logical relations** [Plotkin 77].
The two semantics are related

- **Theorem (soundness).** If $C, M \rightarrow^* C', M'$ then $\llbracket C[M] \rrbracket \rho = \llbracket C'[M'] \rrbracket \rho$

- In particular, if $\_\_ \rightarrow^* \_\_, n$ $(n \in \mathbb{N})$, then $\llbracket M \rrbracket \rho = n$

- **Theorem (adequacy).** If $\llbracket M \rrbracket \rho = n \in \mathbb{N}$ $(\neq \bot)$, then the machine terminates: $\_\_, M \rightarrow^* \_\_, n$

- That is trickier to prove, and uses logical relations [Plotkin 77].

- **Note:** only works at the observable type `nat`
  (although some variants can be made to work at function types, too [BloomRiecke 88])
Observational equivalence

Let $M \cong N$ if and only if

$M$ and $N$ are (operationally) indistinguishable
by any context of observable type
Observational equivalence

Let $M \cong N$ if and only if $M$ and $N$ are (operationally) indistinguishable by any context of observable type.

I.e., you can replace $M$ by $N$ (or conversely) in any program, and you will never see the difference (except for timing, e.g.).
Observational equivalence

- Let $M \cong N$ if and only if $M$ and $N$ are (operationally) indistinguishable by any context of observable type

- Formally: $M \cong N$ (where $M$, $N$ are closed) iff
  for every context $C : \text{nat}$, for every $n \in \mathbb{N}$,
  $C, M \rightarrow^*_\_ , n$ if and only if $C, N \rightarrow^*_\_ , n$

I.e., you can replace $M$ by $N$ (or conversely) in any program, and you will never see the difference (except for timing, e.g.)
Observational equivalence

- Let $M \equiv N$ if and only if
  $M$ and $N$ are (operationally) indistinguishable by any context of observable type

- Formally: $M \equiv N$ (where $M, N$ are closed) iff
  for every context $C : \text{nat}$, for every $n \in \mathbb{N}$,
  $C, M \rightarrow^{*} n$ if and only if $C, N \rightarrow^{*} n$

- Fact. If $[M] = [N]$ then $M \equiv N$. (no $\rho$ necessary in $[M]\rho$)

I.e., you can replace $M$ by $N$ (or conversely) in any program, and you will never see the difference (except for timing, e.g.)
Observational equivalence

Let $M \cong N$ if and only if $M$ and $N$ are (operationally) indistinguishable by any context of observable type.

Formally: $M \cong N$ (where $M$, $N$ are closed) iff for every context $C : \text{nat}$, for every $n \in \mathbb{N}$, $C, M \rightarrow^*_\_ , n$ if and only if $C, N \rightarrow^*_\_ , n$.

Fact. If $[M] = [N]$ then $M \cong N$.

Proof. If $C, M \rightarrow^*_\_ , n$ then $[C[M]] = n$ (soundness)
so $[C[N]] = n$ (denotational semantics is compositional)
so $C, N \rightarrow^*_\_ , n$ (adequacy) $\square$

I.e., you can replace $M$ by $N$ (or conversely) in any program, and you will never see the difference (except for timing, e.g.)
Full abstraction

Fact. If $\llbracket M \rrbracket = \llbracket N \rrbracket$ then $M \cong N$. 
Full abstraction

Fact. If \( [M] = [N] \) then \( M \cong N \).

Full abstraction is when this is an equivalence… and it fails in PCF [Sazonov76, Plotkin 77]:

some murky issue with parallel or
(a function that exists semantically but that not in the syntax)
... which I will ignore here
Probabilistic PCF
Probabilistic choice

- Add instruction $M \oplus N$: do $M$ or $N$, with probability $\frac{1}{2}$
Probabilistic choice

- Add instruction $M \oplus N$: do $M$ or $N$, with probability $\frac{1}{2}$
- Operational semantics with extra labeled transitions
Probabilistic choice

- Add instruction $M \oplus N$: do $M$ or $N$, with probability $\frac{1}{2}$
- Operational semantics with extra labeled transitions
- Denotational semantics is now (as a first approach) a (sub)probability distribution over possible outcomes
Probabilistic choice

- Add instruction $M \oplus N$: do $M$ or $N$, with probability $\frac{1}{2}$
- Operational semantics with extra labeled transitions $C, M \oplus N$ $\frac{1}{2}$ $C, M$ $\frac{1}{2}$ $C, N$
- Denotational semantics is now (as a first approach) a (sub)probability distribution over possible outcomes
- What our crossword puzzle generator did:
  (more or less)

(and, no, contrarily to what this may suggest, execution does not produce increasing values in the information ordering)
Probabilistic PCF

\[ M, N, P, \ldots ::= \ldots \]  
 \[ | M \oplus N \quad \text{probabilistic choice} \]  
 \[ | \text{ret } M \quad \text{monad unit} \]  
 \[ | \text{do } x_{\sigma} = M; N \quad \text{sequential composition} \]
Probabilistic PCF

\[ M, N, P, \ldots ::= \ldots \quad (\text{as in PCF}) \]

| \( M \oplus N \) | \text{probabilistic choice} |
|\|\|
| \text{ret } M | \text{monad unit} |
| \text{do } x_\sigma = M; N | \text{sequential composition} |

\[ \sigma, \tau, \ldots ::= \text{nat} | \text{unit} | \sigma \rightarrow \tau | T\tau \]

\( T\tau \): monadic types [Moggi 91]

\[ \begin{align*}
\frac{M : T\tau}{M \oplus N : T\tau} \\
\frac{M : \tau}{\text{ret } M : T\tau} \\
\frac{M : T\sigma}{\text{do } x_\sigma = M; N : T\tau}
\end{align*} \]

\( T\tau \) = type of \textbf{(first-class)} distributions
Probabilistic PCF

\[ M, N, P, \ldots ::= \ldots \]  
\[ | M \oplus N \] \quad \text{probabilistic choice}  
\[ | \text{ret } M \] \quad \text{monad unit}  
\[ | \text{do } x_\sigma = M; N \] \quad \text{sequential composition}  

\[ \text{Types: } \sigma, \tau, \ldots ::= \text{nat} \mid \text{unit} \mid \sigma \rightarrow \tau \mid T\tau \]  
\[ T\tau: \text{monadic types} \ [\text{Moggi 91}] \]  

\[ \text{New operational rules:} \]

Exploration rules
\[ C, \text{do } x = M; N \rightarrow C[\text{do } x = _\_; N], M \]  
\[ \_, \text{ret } M \rightarrow \text{ret } _, M \]

Computation rules
\[ C[\text{do } x = _\_; N], \text{ret } M \rightarrow C, N[x := M] \]
\[ C, M \oplus N \rightarrow^{1/2} M \]  
\[ C, M \oplus N \rightarrow^{1/2} N \]

\[ M : T\tau \quad N : T\tau \]
\[ \frac{M \oplus N : T\tau}{M \oplus N : T\tau} \]
\[ * : \text{unit} \]

\[ \frac{M : \tau}{\text{ret } M : T\tau} \quad \frac{M : T\sigma \quad N : T\tau}{\text{do } x_\sigma = M; N : T\tau} \]

\( T\tau \) = type of (\text{first-class}) distributions
An extended interpreter

<table>
<thead>
<tr>
<th>type term = V of string</th>
<th>(* variables *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A of term * term</td>
<td>(* applications *)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Flip of term * term</td>
<td>(* M⊕N *)</td>
</tr>
<tr>
<td>Ret of term</td>
<td>(* ret M *)</td>
</tr>
<tr>
<td>Do of string * term</td>
<td>(* do x=M;N *)</td>
</tr>
<tr>
<td>* term</td>
<td></td>
</tr>
</tbody>
</table>
An extended interpreter

type term = V of string (* variables *)
| A of term * term (* applications *)
| ...
| Flip of term * term (* M+N *)
| Ret of term (* ret M *)
| Do of string * term * term (* do x=M;N *)

type ctx = Ec (* _ *)
| Ac of ctx * term (* C[_N] *)
| ...
| Rc (* ret _ *)
| Dc of ctx * string * term (* C[do x=_;N] *)
An extended interpreter

type term = V of string  (* variables *)
| A of term * term  (* applications *)
| ...
| Flip of term * term  (* M⊕N *)
| Ret of term  (* ret M *)
| Do of string * term
  * term  (* do x=M;N *)

type ctx = Ec  (* _ *)
| Ac of ctx * term  (* C[_N] *)
| ...
| Rc  (* ret _ *)
| Dc of ctx * string
  * term  (* C[do x=_;N] *)

let rec exec (c:ctx) (t:term) =
match t with
  (* exploration *)
...
| Do(x,m,n) -> exec (Dc(c,x,n),m)
| Ret(m) when c==Ec
  -> exec (Rc,m)
  (* computation *)
| _ -> compute c t
An extended interpreter

**Type definitions**

```plaintext
type term = V of string (* variables *)
  | A of term * term (* applications *)
  | ... (* M ⊕ N *)
  | Flip of term * term (* M ⊕ N *)
  | Ret of term (* ret M *)
  | Do of string * term (* do x=M;N *)

| type ctx = Ec (* _)
  | Ac of ctx * term (* C[_.N] *)
  | ... (* C[do x=_.N] *)
  | Rc
  | Dc of ctx * string (* do x=_.N *)
```

**Implementation**

```plaintext
let rec exec (c:ctx) (t:term) =
  match t with
  (* exploration *)
  ... (* computation *)
  | Do(x,m,n) -> exec (Dc(c,x,n),m)
  | Ret(m) when c==Ec -> exec (Rc,m) (* computation *)
  | _ -> compute c t
```

```plaintext
and compute (c:ctx) (t:term) =
  match c, t with
  ... (* computation *)
  | Dc(c',x,n), Ret(m) -> exec c' (subst n x m)
  | _, Flip(m,n) -> if Random.bool() then exec c m else exec c n
  | Rc, result -> result (* finished! *)
  | Ec, result -> result (* finished! *)
  | _, _ -> failwith « stumped »
```

**Typing rules**

- $C, do x = M, N \rightarrow C[do x = _.N], M$
- $\_ ret M \rightarrow ret \_ M$
- $C, M \oplus N \rightarrow \frac{1}{2} M$
- $C, M \oplus N \rightarrow \frac{1}{2} N$
Let’s run it, on another example

```
goubault@macbook-pro-de-jean Topics2023 % gimml
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "prob_pcf.ml"
```
Let’s run it, on another example

```
goubault@macbook-pro-de-jean$ Topics2023 % gimml
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "prob_pcf.ml"
```

```
Rand3 ≜ \lambda r. (ret 0 ⊕ ret 1) ⊕ (ret 2 ⊕ r))
```
What does Rand3 do?

\[ Rand3 \triangleq \text{rec}(\lambda r. \left( \text{ret 0} \oplus \text{ret 1} \right) \oplus \left( \text{ret 2} \oplus r \right) ) \]
What does Rand3 do?

\[
\text{Rand3} \triangleq \text{rec}(\lambda r. (\text{ret } 0 \oplus \text{ret } 1) \oplus (\text{ret } 2 \oplus r))
\]
What does Rand3 do?

\[ \text{Rand3} \triangleq \text{rec}(\lambda r. (\text{ret } 0 \oplus \text{ret } 1) \oplus (\text{ret } 2 \oplus r)) \]

\[ \Pr[\_ . \text{Rand3} \downarrow n] = \frac{1}{3} \text{ for every } n \in \{0,1,2\} : \text{proof?} \]
What does Rand3 do?

\[
\text{Rand3} \triangleq \text{rec}(\lambda r \cdot (\text{ret } 0 \oplus \text{ret } 1) \oplus (\text{ret } 2 \oplus r))
\]

\[
\text{Pr}\left[\_ \cdot \text{Rand} \downarrow n\right] = \frac{1}{3} \quad \text{for every } n \in \{0, 1, 2\}: \text{proof?}
\]

\[
\text{Operationally: } \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \cdots = \frac{1}{3}
\]
What does Rand3 do?

Pr[_. Rand3 ↓ n] = \frac{1}{3} \quad \text{for every } n \in \{0,1,2\}: \text{proof?}

Operationally: \quad \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \cdots = \frac{1}{3}

Denotationally: \quad [Rand3]_\rho(\{n\}) \quad \text{(denotational measure of getting } n) = x

where \quad \frac{1}{4} + \frac{1}{4} \cdot x = x \quad \text{(rec defines a fixed point)}
Denotational semantics for probabilistic PCF

- Introduced in Claire Jones’ PhD thesis [Jones 90]
- $\llbracket T_\tau \rrbracket \doteq V_{\leq 1}(\llbracket \tau \rrbracket)$ dcpo of subprobability valuations on $\llbracket \tau \rrbracket$
  (~ think « subprobability measures »)
Denotational semantics for probabilistic PCF

- Introduced in Claire Jones’ PhD thesis [Jones 90]
- $\llbracket T\tau \rrbracket \triangleq V_{\leq 1}(\llbracket \tau \rrbracket)$, dcpo of subprobability valuations on $\llbracket \tau \rrbracket$
  
  (~ think « subprobability measures »)

- $\llbracket M \oplus N \rrbracket_\rho \triangleq \frac{1}{2}\llbracket M \rrbracket_\rho + \frac{1}{2}\llbracket N \rrbracket_\rho$

<table>
<thead>
<tr>
<th>$M : T\tau$</th>
<th>$N : T\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M \oplus N : T\tau$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$M : \tau$</th>
<th>$M : T\sigma$</th>
<th>$N : T\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ret } M : T\tau$</td>
<td>$\text{do } x_\sigma = M; N : T\tau$</td>
<td></td>
</tr>
</tbody>
</table>
Denotational semantics for probabilistic PCF

- Introduced in Claire Jones’ PhD thesis [Jones 90]
- $\llbracket \text{T}_\tau \rrbracket \hat{=} V_{\leq 1}(\llbracket \tau \rrbracket)$ dcpo of subprobability valuations on $\llbracket \tau \rrbracket$
  (~ think « subprobability measures »)
- $\llbracket \text{M} \oplus \text{N} \rrbracket_\rho \hat{=} \frac{1}{2} \llbracket \text{M} \rrbracket_\rho + \frac{1}{2} \llbracket \text{N} \rrbracket_\rho$
- $\llbracket \text{ret} \text{M} \rrbracket_\rho \hat{=} \delta_{\llbracket \text{M} \rrbracket_\rho}$
- $\llbracket \text{do } x_\sigma = \text{M}; \text{N} \rrbracket_\rho \hat{=} (U \mapsto \int_x \llbracket \text{N} \rrbracket_\rho(U) \, d\llbracket \text{M} \rrbracket_\rho)$
Denotational semantics for probabilistic PCF

- Introduced in Claire Jones’ PhD thesis [Jones 90]
- $\llbracket T_{\tau} \rrbracket \doteq V_{\leq 1}(\llbracket \tau \rrbracket)$
  - dcpo of subprobability valuations on $\llbracket \tau \rrbracket$
  - ($\sim$ think « subprobability measures »)

- $\llbracket M \oplus N \rrbracket_{\rho} \doteq \frac{1}{2} \llbracket M \rrbracket_{\rho} + \frac{1}{2} \llbracket N \rrbracket_{\rho}$
- $\llbracket \text{ret } M \rrbracket_{\rho} \doteq \delta_{\llbracket M \rrbracket_{\rho}}$
- $\llbracket \text{do } x_{\sigma} = M; N \rrbracket_{\rho} \doteq (U \mapsto \int_{x} \llbracket N \rrbracket_{\rho}(U) \, d[\llbracket M \rrbracket_{\rho}])$

Let us define all that first!
Continuous valuations
... We come to some topology at last

- Measures give mass to **measurable** subsets
- Following a remark by [Smyth 83] that testable property=open, Jones remarks that it is more natural to measure **open** subsets
- Hence what are the **open** subsets of a dcpo? Let me introduce the **Scott topology**.
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
  - if $\operatorname{sup}^\uparrow_i x_i \in U$ then some $x_i \in U$ already
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
  - if $\sup_i x_i \in U$ then some $x_i \in U$ already
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
  - if $\sup_{i} x_i \in U$ then some $x_i \in U$ already
- Equivalently, iff its characteristic map $\chi_U : X \to \mathbb{S}$ is Scott-continuous, where $\mathbb{S} \triangleq \{0 < 1\}$
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
  - if $\sup_{i} x_{i} \in U$ then some $x_{i} \in U$ already
- Equivalently, iff its characteristic map $\chi_{U} : X \to \mathbb{S}$ is Scott-continuous, where $\mathbb{S} \doteq \{0 < 1\}$
- The collection of Scott-open sets forms a topology: the **Scott topology**
The Scott topology

- A subset $U$ of a dcpo $X$ is **Scott-open** iff:
  - $U$ is **upwards closed** ($x \in U, x \leq y \Rightarrow y \in U$)
  - if $\sup_{i} x_i \in U$ then some $x_i \in U$ already
- Equivalently, iff its characteristic map $\chi_U: X \to \mathbb{S}$ is Scott-continuous, where $\mathbb{S} \doteq \{0 < 1\}$
- The collection of Scott-open sets forms a topology: the **Scott topology**
- The **Scott-continuous** maps are exactly the **continuous** maps for the Scott topologies
Continuous valuations

- First studied by [SahebDjahromi 80]: gives mass to Scott-open subsets
- Makes sense on every topological space, not just dcpos with the Scott topology
- Let $\mathcal{O}_X$ denote the lattice of open subsets of a space $X$

**Definition.** A valuation $\nu$ on $X$ is a map $\nu: \mathcal{O}_X \to \mathbb{R}_+$ satisfying:
Continuous valuations

❖ First studied by [SahebDjahromi 80]: gives mass to **Scott-open** subsets
❖ Makes sense on every topological space, not just dcpos with the Scott topology
❖ Let $\mathcal{O}X$ denote the lattice of open subsets of a space $X$

**Definition.** A valuation $\nu$ on $X$ is a map $\nu : \mathcal{O}X \to \mathbb{R}_+$ satisfying:

❖ **strictness:** $\nu(\emptyset) = 0$
Continuous valuations

- First studied by [SahebDjahromi 80]: gives mass to **Scott-open** subsets
- Makes sense on *every* topological space, not just dcpos with the Scott topology
- Let $\mathcal{O}X$ denote the lattice of open subsets of a space $X$

**Definition.** A valuation $\nu$ on $X$ is a map $\nu: \mathcal{O}X \to \mathbb{R}_+$ satisfying:

- **strictness:** $\nu(\emptyset) = 0$
- **monotonicity:** $\nu(U) \leq \nu(V)$ if $U \subseteq V$
Continuous valuations

- First studied by [SahebDjahromi 80]: gives mass to **Scott-open** subsets
- Makes sense on **every topological space**, not just dcpos with the Scott topology
- Let \(\mathcal{O}X\) denote the lattice of open subsets of a space \(X\)

**Definition.** A **valuation** \(\nu\) on \(X\) is a map \(\nu: \mathcal{O}X \rightarrow \mathbb{R}_+\) satisfying:

- **strictness:** \(\nu(\emptyset) = 0\)
- **monotonicity:** \(\nu(U) \leq \nu(V)\) if \(U \subseteq V\)
- **modularity:** \(\nu(U \cup V) + \nu(U \cap V) = \nu(U) + \nu(V)\)
Continuous valuations

- First studied by [SahebDjahromi 80]: gives mass to Scott-open subsets
- Makes sense on every topological space, not just dcpos with the Scott topology
- Let $\mathcal{O}_X$ denote the lattice of open subsets of a space $X$

**Definition.** A valuation $\nu$ on $X$ is a map $\nu: \mathcal{O}_X \to \mathbb{R}_+$ satisfying:

- **strictness:** $\nu(\emptyset) = 0$
- **monotonicity:** $\nu(U) \leq \nu(V)$ if $U \subseteq V$
- **modularity:** $\nu(U \cup V) + \nu(U \cap V) = \nu(U) + \nu(V)$

A continuous valuation is not just monotonic, but Scott-continuous: $\nu(\sup_i U_i) = \sup_i \nu(U_i)$. 
Simple valuations

Definition. The Dirac valuation $\delta_x$:

$$\delta_x(U) \triangleq \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise} \end{cases}$$

is a continuous valuation.

If you draw at random with respect to $\delta_x$, you will get $x$ all the time.
Simple valuations

❖ **Definition.** The Dirac valuation $\delta_x$: 
\[
\delta_x(U) = \begin{cases} 
1 & \text{if } x \in U \\
0 & \text{otherwise}
\end{cases}
\]
is a continuous valuation.

❖ If you draw at random with respect to $\delta_x$, you will get $x$ **all the time**.

❖ **Definition.** A simple valuation is 
\[
\sum_{i=1}^{n} a_i \delta_{x_i}, \text{ where } a_i \in \mathbb{R}_+
\]
Simple valuations

Definition. The Dirac valuation $\delta_x$:

$$\delta_x(U) \triangleq \begin{cases} 
1 & \text{if } x \in U \\
0 & \text{otherwise}
\end{cases}$$

is a continuous valuation.

If you draw at random with respect to $\delta_x$, you will get $x$ all the time.

Definition. A simple valuation is

$$\sum_{i=1}^{n} a_i \delta_{x_i}$$

where $a_i \in \mathbb{R}_+$

... draws each $x_i$ with probability $a_i$ (assuming $x_i$ pairwise distinct)
Simple valuations

Definition. The Dirac valuation $\delta_x$: 

$$ \delta_x(U) \equiv \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise} \end{cases} $$

is a continuous valuation.

Definition. A simple valuation is 

$$ \sum_{i=1}^{n} a_i \delta_{x_i} $$

where $a_i \in \mathbb{R}_+$

… draws each $x_i$ with probability $a_i$ (assuming $x_i$ pairwise distinct)

always Scott-continuous
Other continuous valuations

❖ All (bounded) valuations are simple on a finite space

❖ Our dcpos $[\tau]$ will **not** be finite
Other continuous valuations

- All (bounded) valuations are simple on a finite space
- Our dcpos $\tau$ will not be finite
- In general, not all continuous valuations are simple.  
  E.g., Lebesgue measure $\lambda$ on $\mathbb{R}$ (restricted to its open subsets)  
  is a continuous valuation that is not simple.
Other continuous valuations

❖ All (bounded) valuations are simple on a finite space
❖ Our dcpo $[\tau]$ will not be finite
❖ In general, not all continuous valuations are simple. E.g., Lebesgue measure $\lambda$ on $\mathbb{R}$ (restricted to its open subsets) is a continuous valuation that is not simple
❖ Here is an example on a dcpo…
The dcpo $\mathbb{IR}$

- Elements of $\mathbb{IR}$: intervals $[a, b]$ with $a \leq b \in \mathbb{R}$
  We equate $[a, a]$ with $a \in \mathbb{R}$, so $\mathbb{R} \subseteq \mathbb{IR}$

- Ordered by reverse inclusion:
  $[a, b] \leq [c, d]$ iff $[a, b] \supseteq [c, d]$
The dcpo $\mathbb{IR}$

- **Elements of $\mathbb{IR}$:** intervals $[a, b]$ with $a \leq b \in \mathbb{R}$
  We equate $[a, a]$ with $a \in \mathbb{R}$, so $\mathbb{R} \subseteq \mathbb{IR}$

- **Ordered by reverse inclusion:**
  $[a, b] \leq [c, d]$ iff $[a, b] \supseteq [c, d]$

- **Fact.** The open subsets of $\mathbb{R}$ (with its usual metric topology)
  are exactly the sets $U \cap \mathbb{R}$,
  where $U$ ranges over the **Scott-open** subsets of $\mathbb{IR}$
The dcpo $\mathbb{R}$

- Elements of $\mathbb{R}$: intervals $[a, b]$ with $a \leq b \in \mathbb{R}$
  We equate $[a, a]$ with $a \in \mathbb{R}$, so $\mathbb{R} \subseteq \mathbb{R}$

- Ordered by **reverse** inclusion:
  $[a, b] \leq [c, d]$ iff $[a, b] \supseteq [c, d]$

- **Fact.** The open subsets of $\mathbb{R}$ (with its usual metric topology) are exactly the sets $U \cap \mathbb{R}$, where $U$ ranges over the **Scott-open** subsets of $\mathbb{IR}$

- Now let $\bar{\lambda}(U) \cong \lambda(U \cap \mathbb{R})$, for every $U \in \mathcal{O}(\mathbb{IR})$:
  $\bar{\lambda}$ is a continuous valuation on the dcpo $\mathbb{R}$
  that is **not a simple valuation**  
  (Note: $\bar{\lambda}$ is supported on $\mathbb{R}$)
Beyond simple valuations

❖ Maybe we only need *simple* valuations in computer science?
❖ No: simple valuations do not form a **dcpo**
Beyond simple valuations

- Maybe we only need **simple** valuations in computer science?
- No: simple valuations do not form a dcpo
- E.g., \( \bar{\lambda}_{[0,1]} = \sup_{n \in \mathbb{N}} \bar{\lambda}_n \) (uniform measure on \([0,1]\)), where \( \bar{\lambda}_n \triangleq \sum_{i=1}^{2^n} \frac{1}{2^n} \delta_{[\frac{i-1}{2^n}, \frac{i}{2^n}]} \)

(there is a similar formula for \( \bar{\lambda} \) itself, but I wish to show you a **probability** valuation)
Beyond simple valuations

- Maybe we only need **simple** valuations in computer science?
- No: simple valuations do not form a **dcpo**
- E.g., \( \tilde{\lambda}_{[0,1]} = \sup_{n \in \mathbb{N}} \tilde{\lambda}_n \) (uniform measure on \([0,1]\)),

\[
\tilde{\lambda}_n \triangleq \sum_{i=1}^{2^n} \frac{1}{2^n} \delta_{\left[ \frac{i-1}{2^n}, \frac{i}{2^n} \right]}
\]

(there is a similar formula for \( \tilde{\lambda} \) itself, but I wish to show you a **probability** valuation)
Beyond simple valuations

- Maybe we only need **simple** valuations in computer science?
- No: simple valuations do not form a **dcpo**
- E.g., \( \overline{\lambda}_{[0,1]} = \sup_{n \in \mathbb{N}} \overline{\lambda}_n \) (uniform measure on \([0,1]\)),
  where \( \overline{\lambda}_n \triangleq \sum_{i=1}^{2^n} \frac{1}{2^n} \delta_{\left[ \frac{i-1}{2^n}, \frac{i}{2^n} \right]} \)
  (there is a similar formula for \( \overline{\lambda} \) itself, but I wish to show you a **probability** valuation)
Beyond simple valuations

- Maybe we only need **simple** valuations in computer science?
- No: simple valuations do not form a **dcpo**
- E.g., \( \overline{\lambda}_{[0,1]} = \sup_{n \in \mathbb{N}} \overline{\lambda}_n \) (uniform measure on \([0,1]\)),
  
  where \( \overline{\lambda}_n \triangleq \sum_{i=1}^{2^n} \frac{1}{2^n} \delta_{\left[\frac{i-1}{2^n}, \frac{i}{2^n}\right]} \)
  
  (there is a similar formula for \( \overline{\lambda} \) itself, but I wish to show you a **probability** valuation)
Continuous valuations and measures

- **Theorem [Adamski 77].** Given any Borel measure on a hereditarily Lindelöf space, its restriction to open sets is a **continuous valuation.** (i.e., every Borel measure on such a space is \(\tau\)-smooth.)

- **Theorem [de Brecht, JGL, Jia, Lyu 2019].** Every **continuous valuation** on an LCS-complete space extends to a **Borel measure.** (a \(G_\delta\) subspace of a locally compact sober space)

- A sweet spot: de Brecht’s **quasi-Polish spaces** (=2nd countable LCS-complete)
  Those include all **\(\omega\)-continuous dcpos** + all **Polish** spaces
Continuous valuations and measures

- **Theorem [Adamski 77].** Given any Borel measure on a hereditarily Lindelöf space, its restriction to open sets is a continuous valuation (i.e., every Borel measure on such a space is \( \tau \)-smooth).

- **Theorem [de Brecht, JGL, Jia, Lyu 2019].** Every continuous valuation on an LCS-complete space (a \( G_\delta \) subspace of a locally compact sober space) extends to a Borel measure.

- A sweet spot: de Brecht’s quasi-Polish spaces (=2nd countable LCS-complete)
  Those include all \( \omega \)-continuous dcpos + all Polish spaces

Just remember that continuous valuations~measures, in most cases (not all, though)
Integration

- For every lower semicontinuous map \( f: X \rightarrow \bar{\mathbb{R}}_+ \) and every continuous valuation \( \mu \) on \( X \),
  
  there is an integral \( \int_X f(x) d\mu \) (or \( \int f \, d\mu \) for short)

[Jones 90, Tix 95]
**Integration**

- For every **lower semicontinuous** map $f: X \to \overline{\mathbb{R}}_+$ and every continuous valuation $\mu$ on $X$, there is an integral $\int_X f(x) d\mu$ (or $\int f \, d\mu$ for short)

[Jones 90, Tix 95]
Integration

❖ For every **lower semicontinuous** map $f: X \to \mathbb{R}_+$ and every continuous valuation $\mu$ on $X$, there is an integral $\int_x f(x) d\mu$ (or $\int f \, d\mu$ for short)

[Jones 90, Tix 95]

I.e., continuous from $X$ to $(\mathbb{R}_+, \text{Scott topology})$

If $X$ is a dcpo, this just means **Scott-continuous**, but the definition works for any topological space $X$
The probabilistic powerdomain

- The **probabilistic powerdomain** $V_{\leq 1}X$ of a space $X$ is the dcpo of all subprobability (continuous) valuations $\nu$ on $X$.
The probabilistic powerdomain $\mathbf{V}_{\leq 1} X$ of a space $X$ is the dcpo of all subprobability (continuous) valuations $\nu$ on $X$

i.e., $\nu(X) \leq 1$
The probabilistic powerdomain

The probabilistic powerdomain $\mathbf{V}_{\leq 1} X$ of a space $X$ is the dcpo of all subprobability (continuous) valuations $\nu$ on $X$

\[ \text{i.e., } \nu(X) \leq 1 \]

 ordered by $\mu \leq \nu$ iff for every $U \in \emptyset X$, $\mu(U) \leq \nu(U)$
The probabilistic powerdomain $\mathbf{V}_{\leq 1} X$ of a space $X$ is the dcpo of all subprobability (continuous) valuations $\nu$ on $X$

i.e., $\nu(X) \leq 1$

ordered by $\mu \leq \nu$ iff for every $U \in \emptyset X$, $\mu(U) \leq \nu(U)$

We can now define (as promised):
- $\llbracket T \tau \rrbracket \triangleq \mathbf{V}_{\leq 1}(\llbracket \tau \rrbracket)$ first-class subprobability distributions
- $\llbracket M \oplus N \rrbracket \rho \triangleq \frac{1}{2} \llbracket M \rrbracket \rho + \frac{1}{2} \llbracket N \rrbracket \rho$
- $\llbracket \text{ret } M \rrbracket \rho \triangleq \delta_{\llbracket M \rrbracket \rho}$
- $\llbracket \text{do } x_\sigma = M; N \rrbracket \rho \triangleq (U \in \emptyset(\llbracket \sigma \rrbracket) \mapsto \int_x \llbracket N \rrbracket \rho(U) \, d[\llbracket M \rrbracket \rho)$
The probabilistic powerdomain $V_{\leq 1}X$ of a space $X$ is the dcpo of all subprobability (continuous) valuations $\nu$ on $X$

ordered by $\mu \leq \nu$ iff for every $U \in \mathcal{O}X$, $\mu(U) \leq \nu(U)$

We can now define (as promised):

$\llbracket T_{\tau} \rrbracket \doteq V_{\leq 1}(\llbracket \tau \rrbracket)$ first-class subprobability distributions

$\llbracket M \oplus N \rrbracket_{\rho} \doteq \frac{1}{2} \llbracket M \rrbracket_{\rho} + \frac{1}{2} \llbracket N \rrbracket_{\rho}$

$\llbracket \text{ret } M \rrbracket_{\rho} \doteq \delta_{\llbracket M \rrbracket_{\rho}}$

$\llbracket \text{do } x_{\sigma} = M; N \rrbracket_{\rho} \doteq (U \in \mathcal{O}(\llbracket \sigma \rrbracket) \mapsto \int_{x} \llbracket N \rrbracket_{\rho}(U) d\llbracket M \rrbracket_{\rho})$

Really defines a strong monad on Dcpo
Soundness, adequacy

- Operationally, let $\Pr[C, M \downarrow n] \triangleq$ the (sub)probability that $C, M \rightarrow^* \text{ret } _, n$
  — sum over all executions $C, M \rightarrow^* \text{ret } _, n$
  — of their weights, defined as the product of their probability labels

- **Prop (soundness).** $\Pr[C, M \downarrow n] \leq \llbracket C[M] \rrbracket_\rho(\{n\})$ (at type **nat**)
Soundness, adequacy

- Operationally, let $\Pr[C, M \downarrow n] \triangleq$ the (sub)probability that $C, M \rightarrow^* \text{ret } _, n$
  - sum over all executions $C, M \rightarrow^* \text{ret } _, n$
  - of their weights, defined as the product of their probability labels

- Prop (soundness). $\Pr[C, M \downarrow n] \leq \llbracket C[M] \rrbracket \rho(\{n\})$ (at type $\text{nat}$)

- Theorem (adequacy). $\Pr[C, M \downarrow n] = \llbracket C[M] \rrbracket \rho(\{n\})$ (at type $\text{nat}$)
Soundness, adequacy

- Operationally, let $\text{Pr}[C, M \downarrow n] \triangleq$ the (sub)probability that $C, M \rightarrow^{*} \text{ret }_, n$
  — sum over all executions $C, M \rightarrow^{*} \text{ret }_, n$
  — of their weights, defined as the product of their probability labels

- **Prop (soundness).** $\text{Pr}[C, M \downarrow n] \leq \llbracket C[M] \rrbracket \rho(\{n\})$ (at type $\text{nat}$)

- **Theorem (adequacy).** $\text{Pr}[C, M \downarrow n] = \llbracket C[M] \rrbracket \rho(\{n\})$ (at type $\text{nat}$)

- Soundness is a bit harder than before: requires $\llbracket C \rrbracket$ linear

- Adequacy: by a suitable form of logical relations
No full abstraction (1/2)

- Fact. If $\llbracket M \rrbracket = \llbracket N \rrbracket$ then $M \cong N$. (as in PCF, consequence of adequacy)
- Just like PCF, probabilistic PCF is not fully abstract
No full abstraction (1/2)

- Fact. If $⟦M⟧=⟦N⟧$ then $M ≅ N$.  
  (as in PCF, consequence of adequacy)
- Just like PCF, probabilistic PCF is **not** fully abstract
- **Parallel or** is a problem, as before
Fact. If $\llbracket M \rrbracket = \llbracket N \rrbracket$ then $M \cong N$. (as in PCF, consequence of adequacy)

Just like PCF, probabilistic PCF is not fully abstract

Parallel or is a problem, as before

There is a second obstacle to full abstraction here: the statistical testers $[V > r] \in \llbracket T \tau \rightarrow T \text{unit} \rrbracket$
mapping $\nu \in \llbracket T \tau \rrbracket$ to $\delta_*$ if $\nu(V) > r$
to 0 otherwise

are not definable in probabilistic PCF

... leading to another cause of failure of full abstraction
No full abstraction (2/2)

- Fact. If $[M] = [N]$ then $M \cong N$.

- Even if we added primitive parallel or and statistical testers, we do not have the technology to prove the converse.

- The problem is that we need all dcpo $[\tau]$ to be at least continuous dcpo (a.k.a. domains) to hope to obtain a proof.
Continuous deposits, a.k.a. domains
Continuous dcpos, a.k.a. domains

- Motto: the continuous dcpos are the nice dcpos, where (almost) every property you wish for is true
Continuous dcpos, a.k.a. domains

❖ Motto: the continuous dcpos are the nice dcpos, where (almost) every property you wish for is true

❖ Let \( x \ll y \) (\( x \) way-below \( y \)) iff
  \[
y \leq \sup_i z_i \text{ implies } \exists i, y \leq z_i
  \]
Continuous dcpos, a.k.a. domains

- Motto: the continuous dcpos are the **nice** dcpos, where (almost) every property you wish for is true

- Let $x \ll y$ (x way-below y) iff
  
  $y \leq \sup_i z_i$ implies $\exists i, y \leq z_i$
Continuous dcpos, a.k.a. domains

- **Motto:** the continuous dcpos are the **nice** dcpos, where (almost) every property you wish for is true

- Let \( x \ll y \) (\( x \) way-below \( y \)) iff
  \[
  y \leq \sup_i z_i \text{ implies } \exists i, y \leq z_i
  \]
Continuous dcpos, a.k.a. domains

❖ Motto: the continuous dcpos are the nice dcpos, where (almost) every property you wish for is true

❖ Let $x \ll y$ (x way below y) iff
  \[ y \leq \sup_{i} z_{i} \text{ implies } \exists i, y \leq z_{i} \]
Continuous dcpos, a.k.a. domains

- Motto: the continuous dcpos are the **nice** dcpos, where (almost) every property you wish for is true

- Let \( x \ll y \) (\( x \) way-below \( y \)) iff
  \[ y \leq \sup_i z_i \iff \exists i, y \leq z_i \]

  In order to get at least as much information as \( y \), you must get at least as much information as \( x \) at some **finite stage**
Continuous dcpos, a.k.a. domains

❖ Motto: the continuous dcpos are the nice dcpos, where (almost) every property you wish for is true

❖ Let $x \ll y$ (x way-below y) iff $y \leq \sup_{i} z_{i}$ implies $\exists i, y \leq z_{i}$

❖ **Definition.** A dcpo is continuous iff every point $x$ is the supremum of some directed family of points way-below $x$.

in order to get at least as much information as $y$, you must get at least as much information as $x$ at some finite stage
Examples, counterexamples

❖ $\mathbb{N}_\perp$, the semantics of $\text{nat}$ in PCF is a continuous dcpo
Examples, counterexamples

- 😊 $\mathbb{N}_\bot$, the semantics of $\text{nat}$ in PCF is a continuous dcpo

- Theorem [Jones 90]. For every continuous dcpo $X$, $V_{\leq 1}X$ is a continuous dcpo

  Hence $[\tau]$ continuous dcpo $\Rightarrow [T\tau]$ continuous dcpo in probabilistic PCF
Examples, counterexamples

❖ $\mathbb{N}_\bot$, the semantics of `nat` in PCF is a continuous dcpo

❖ **Theorem [Jones 90]**. For every continuous dcpo $X$, $V_{\leq 1} X$ is a continuous dcpo

Hence $\llbracket \tau \rrbracket$ continuous dcpo $\Rightarrow$ $\llbracket T\tau \rrbracket$ continuous dcpo in probabilistic PCF

❖ **Fact**. In general, $[X \rightarrow Y]$ is not continuous, even if $X$ and $Y$ are.

(Consider $X \equiv Y \equiv \mathbb{Z}$.)
Examples, counterexamples

- 😊 $\mathbb{N}_\bot$, the semantics of nat in PCF is a continuous dcpo

- **Theorem [Jones 90].** For every continuous dcpo $X$, $V_{\leq 1}X$ is a continuous dcpo
  
  Hence $[\tau]$ continuous dcpo $\Rightarrow$ $[T\tau]$ continuous dcpo in probabilistic PCF

- 😞 **Fact.** In general, $[X \to Y]$ is not continuous, even if $X$ and $Y$ are.
  
  (Consider $X \triangleright Y \triangleleft Z^\bot$.)
Examples, counterexamples

- 😊 $\mathbb{N}_\bot$, the semantics of \texttt{nat} in PCF is a continuous dcpo

- **Theorem [Jones 90]**. For every continuous dcpo $X$, $V_{\leq 1} X$ is a continuous dcpo
  Hence $\llbracket \tau \rrbracket$ continuous dcpo $\Rightarrow \llbracket T\tau \rrbracket$ continuous dcpo in probabilistic PCF

- 😞 **Fact**. In general, $[X \to Y]$ is not continuous, even if $X$ and $Y$ are.
  (Consider $X \Dashdot Y \Dashdot \mathbb{Z}^-$.)
A **bc-domain** is a continuous dcpo that:
- is pointed (has a \( \perp \))
- any two elements \( x, y \) with an upper bound
  - have a **least** upper bound \( x \lor y \)
A bc-domain is a continuous dcpo that:
- is pointed (has a $\bot$)
- any two elements $x, y$ with an upper bound have a least upper bound $x \lor y$

$\mathbb{N}_{\bot}$ is a bc-domain
Cartesian-closed categories of domains 1: bc-domains

- A **bc-domain** is a continuous dcpo that:
  - is pointed (has a $\perp$)
  - any two elements $x, y$ with an upper bound have a **least** upper bound $x \lor y$

- $\mathbb{N}_\perp$ is a bc-domain
- not $\mathbb{IR}$ (no $\perp$)
Cartesian-closed categories of domains 1: bc-domains

❖ A **bc-domain** is a continuous dcpo that:
   — is pointed (has a $\bot$)
   — any two elements $x, y$ with an upper bound have a **least** upper bound $x \lor y$

❖ $\mathbb{N}_\bot$ is a bc-domain
❖ not $\mathbb{IR}$ (no $\bot$)
❖ $\mathbb{IR}_\bot$ is a bc-domain
A **bc-domain** is a continuous dcpo that:
- is pointed (has a \(\perp\))
- any two elements \(x, y\) with an upper bound have a **least** upper bound \(x \lor y\)

**Fact.** For any two bc-domains \(X, Y\), \([X \to Y]\) is a bc-domain.
Cartesian-closed categories of domains 1: bc-domains

- A **bc-domain** is a continuous dcpo that:
  - is pointed (has a ⊥)
  - any two elements $x, y$ with an upper bound have a **least** upper bound $x \lor y$

- **Fact.** For any two bc-domains $X, Y$, $[X \to Y]$ is a bc-domain.

- **Fact [Jones 90].** $\mathcal{V}_{\leq 1} X$ is not a bc-domain in general, even if $X$ is.
  
  (In $X \cong a$ $b$, $\frac{1}{2} \delta_a$ and $\frac{1}{2} \delta_b$ have **uncountably many** minimal upper bounds.)
Cartesian-closed categories of domains 1: bc-domains

- A **bc-domain** is a continuous dcpo that:
  - is pointed (has a \(\bot\))
  - any two elements \(x, y\) with an upper bound have a least upper bound \(x \lor y\)

- **Fact.** For any two bc-domains \(X, Y\), \([X \to Y]\) is a bc-domain.

- **Fact [Jones 90].** \(V_{\leq 1}X\) is not a bc-domain in general, even if \(X\) is.

\(\text{In } X \equiv a \quad \frac{1}{2}a\) and \(\frac{1}{2}b\) have uncountably many minimal upper bounds.

\(\Box\) is a bc-domain
\(\Box\) is a bc-domain
\(\Box\) is a bc-domain
\(\Box\) is a bc-domain

\(\Box\) is a bc-domain
\(\Box\) is a bc-domain

\(\Box\) is not \(\Box\) (no \(\bot\))
There are other Cartesian-closed categories of domains: L-domains, RB-domains, FS-domains, etc.
The state of the art

- There are other Cartesian-closed categories of domains: L-domains, RB-domains, FS-domains, etc.
- 😞 None is known to be closed under $V \leq 1$
There are other Cartesian-closed categories of domains: L-domains, RB-domains, FS-domains, etc.

None is known to be closed under $V_{\leq 1}$

As of 2023, the best results are still those of [Jung, Tix 98] apart from [JGL 12] ($V_{\leq 1}$ (QRB-domain) is a QRB-domain) or [Mislove 20] ($V_{\leq 1}$ (chain) is a continuous lattice) or [JGL 22] ($V_{\leq 1}$ (quasi-cont. dcpo) is quasi-continuous)
Statistical programming languages
Meanwhile, statisticians were inventing their own languages in order to describe probabilistic processes:

- Church [Goodman, Mansinghka, Roy, Bonawitz, Tenenbaum 08]
- Anglican [Wood, van de Meent, Mansinghka 14]
- WebPPL [Wood, Stuhlmüller 14]
- Venture [Mansinghka, Selsam, Perov 14]
- Hakaru [Shan, Ramsey 17]
- PPCF [Ehrhard, Pagani, Tasson 18]
- PCFSS [Dal Lago, Hoshino 19]
- etc.

### Dirichlet processes in Church

```
(define DP alpha proc)
  (let ((sticks (= (lambda x (beta 1.0 alpha))))
    (atoms (= (lambda x (proc)))))
    (lambda () (atoms (pick-a-stick sticks 1))))
```

### Hidden Markov models in Venture

```
(define (HMM alpha proc)
  (let ((dps (lambda argps (DP alpha (lambda () (apply proc argps)))))
    (lambda argps (apply dpe argps)))
  )
```

```scheme
(define initial-state-dist (list (/ 1 3) (/ 1 3) (/ 1 3)))
(define get-state-transition-dist (lambda (s)
  (cond ((<= s 0) (list .1 .1 .4)) ((<= s 1) (list .2 .2 .6))
    ((<= s 2) (list .15 .15 .7))))

(define transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state))))
(define get-state (lambda (idx)
  (if (<= idx 0) (discrete initial-state-dist)
    (transition (get-state (- index 1)))))

(define get-state-observation-mean (lambda (s)
  (cond ((<= s 0) -1) ((<= s 1) 1) ((<= s 2) 0))))

(define observe (normal get-state-obs-mean (get-state 1)) .9
(define observe (normal get-state-obs-mean (get-state 2)) .8)

(define predict (get-state 0)
  [predict (get-state 0)]
  ;
  [predict (get-state 16)]
  ```
Meanwhile, statisticians were inventing their own languages in order to describe **probabilistic processes**:

— Church [Goodman,Mansinghka,Roy,Bonawitz,Tenenbaum 08]
— Anglican [Wood,van de Meent,Mansinghka 14]
— WebPPL [Wood,Stuhlmüller 14]
— Venture [Mansinghka,Selsam,Perov 14]
— Hakaru [Shan, Ramsey 17]
— PPCF [Ehrhard, Pagani, Tasson 18]
— PCFSS [Dal Lago, Hoshino 19]
— etc.

Basically pure higher-order functional languages,

```scheme
(define (DP alpha proc)
  (let ((sticks (mem (lambda x (beta 1.0 alpha)))))
    (atom (mem (lambda y (proc))))))
  (lambda () (atom (pick-a-stick sticks 1)))))

(define (pick-a-stick sticks J)
  (if (< (random) (sticks J))
    J
    (pick-a-stick sticks (+ J 1))))

(define (DPsm alpha proc)
  (let ((dps (mem (lambda args
    (DP alpha (lambda () (apply proc args))))))
    (lambda argsm ((apply dpe argsm)))))

(define initial-state-dist (list (/ 1 3) (/ 1 3) (/ 1 3)))
(define get-state-transition-dist (lambda (s)
  (cond ((= s 0) (list .1 .5 .4))
        ((= s 1) (list .2 .2 .6))
        ((= s 2) (list .1 .5 .3))))

(define transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state))))
(define get-state (mem (lambda (index)
  (if (<= index 0) (discrete initial-state-dist)
    (transition (get-state (- index 1)))))))
(define get-state-transition-mean (lambda (s)
  (cond ((= s 0) -1) ((= s 1) 1) ((= s 2) 0))))
(define get-state-observation-mean (lambda (s)
  (cond ((= s 0) 1) ((= s 1) 1) ((= s 2) 0))))
(define get-state-observation-mean (lambda (s)
  (cond ((= s 0) 1) ((= s 1) 1) ((= s 2) 0))))
```

**Dirichlet processes in Church**

```
(define initial-state-dist (list (/ 1 3) (/ 1 3) (/ 1 3)))
(define get-state-transition-dist (lambda (s)
  (cond ((= s 0) (list .1 .5 .4))
        ((= s 1) (list .2 .2 .6))
        ((= s 2) (list .1 .5 .3))))

(define transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state))))
(define get-state (mem (lambda (index)
  (if (<= index 0) (discrete initial-state-dist)
    (transition (get-state (- index 1)))))))
(define get-state-transition-mean (lambda (s)
  (cond ((= s 0) -1) ((= s 1) 1) ((= s 2) 0))))
(define get-state-observation-mean (lambda (s)
  (cond ((= s 0) 1) ((= s 1) 1) ((= s 2) 0))))
```

**Hidden Markov models in Venture**
Meanwhile, statisticians were inventing their own languages in order to describe **probabilistic processes:**

- Church [Goodman,Mansinghka,Roy,Bonawitz,Tenenbaum 08]
- Anglican [Wood,van de Meent,Mansinghka 14]
- WebPPL [Wood,Stuhlmüller 14]
- Venture [Mansinghka,Selsam,Perov 14]
- Hakaru [Shan, Ramsey 17]
- PPCF [Ehrhard, Pagani, Tasson 18]
- PCFSS [Dal Lago, Hoshino 19]
- etc.

Basically pure higher-order functional languages, **plus computation over the reals**
Meanwhile, statisticians were inventing their own languages in order to describe **probabilistic processes:**

— Church [Goodman, Mansinghka, Roy, Bonawitz, Tenenbaum 08]
— Anglican [Wood, van de Meent, Mansinghka 14]
— WebPPL [Wood, Stuhlmüller 14]
— Venture [Mansinghka, Selsam, Perov 14]
— Hakaru [Shan, Ramsey 17]
— PPCF [Ehrhard, Pagani, Tasson 18]
— PCFSS [Dal Lago, Hoshino 19]
— etc.

Basically pure higher-order functional languages,

*plus computation over the reals*

*with a rich set of distributions over the reals*

```
(define (DP alpha proc)
  (let ((sticks (mem (lambda x (beta 1.0 alpha))))
        (atoms (mem (lambda x (proc))))
        (lambda () (atoms (pick-a-stick sticks 1))))))

(define (pick-a-stick sticks J)
  (if (< (random) (sticks J))
    (pick-a-stick sticks (+ J 1))))

(define (DP alpha proc)
  (let ((dps (mem (lambda args
            (DP alpha
            (lambda () (apply proc args)))))))
    (lambda argsin ((apply dpe argsin))))
```
Meanwhile, statisticians were inventing their own languages in order to describe **probabilistic processes:**

- Church [Goodman, Mansinghka, Roy, Bonawitz, Tenenbaum 08]
- Anglican [Wood, van de Meent, Mansinghka 14]
- WebPPL [Wood, Stuhlmüller 14]
- Venture [Mansinghka, Selsam, Perov 14]
- Hakaru [Shan, Ramsey 17]
- PPCF [Ehrhard, Pagani, Tasson 18]
- PPFSS [Dal Lago, Hoshino 19]
- etc.

**Statistical programming languages**

- Basically pure higher-order functional languages,
  - plus computation **over the reals**
  - with a rich set of distributions **over the reals**
  - plus computation of **conditional probabilities**

```
(define (DP alpha proc)
  (let ((sticks (mem (lambda x (beta 1.0 alpha))))
         (atoms (mem (lambda x (proc)))))
    (lambda () (atoms (pick-a-stick sticks 1))))

(define (pick-a-stick sticks J)
  (if (< (random) (sticks J))
      J
      (pick-a-stick sticks (+ J 1))))

(define (Dirichlet alpha proc)
  (let ((dps (mem (lambda args)
                   (DP alpha
                        (lambda () (apply proc args)))))
         (lambda argmin (apply dps argmin)))
    (lambda argmin (apply dps argmin))))
```

**Hidden Markov models in Venture**

```
[assume initial-state-dist (list 1 3) (1 3) (1 3)]
[assume get-state-transition-dist (lambda (s))
  (cond ((= s 0) (list .1 .5 .4)) ((= s 1) (list .2 .2 .6))
        ((= s 2) (list .15 .15 .75)))]
[assume transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state))))
[assume get-state (mem (lambda (index))
  (if (< index 0) (discrete initial-state-dist)
     (transition (get-state (- index 1))))))]
[assume get-state-observation-mean (lambda (s))
  (cond ((= s 0) -1) ((= s 1) 1) ((= s 2) 0))]
[observe (normal (get-state-obs-mean (get-state 1)) 1) .9]
[observe (normal (get-state-obs-mean (get-state 2)) 1) .8]
[observe (normal (get-state-obs-mean (get-state 16)) 1) .8]
[observe (normal (get-state-obs-mean (get-state 16)) 1) .9]
[predict (get-state 0)]
[predict (get-state 1)]
[predict (get-state 15)]
[predict (get-state 16)]
```

**Dirichlet processes in Church**

```
(let ((sticks (mem (lambda x (beta 1.0 alpha))))
         (atoms (mem (lambda x (proc)))))
    (lambda () (atoms (pick-a-stick sticks 1))))
```

**Computation over the reals**

```
(define (pick-a-stick sticks J)
  (if (< (random) (sticks J))
      J
      (pick-a-stick sticks (+ J 1))))
```

**Conditional probabilities**
Statistical programming languages

- Meanwhile, statisticians were inventing their own languages in order to describe **probabilistic processes:**
  - Church [Goodman,Mansinghka,Roy,Bonawitz,Tenenbaum 08]
  - Anglican [Wood,van de Meent,Mansinghka 14]
  - WebPPL [Wood,Stuhlmüller 14]
  - Venture [Mansinghka,Selsam,Perov 14]
  - Hakaru [Shan, Ramsey 17]
  - PPCF [Ehrhard, Pagani, Tasson 18]
  - PCFSS [Dal Lago, Hoshino 19]
  - etc.

- Basically pure higher-order functional languages,
  - plus computation **over the reals**
  - with a rich set of distributions **over the reals**
  - plus computation of **conditional probabilities**

Usually implemented as **floating-point**...
but we can do better (from a formal point of view)

Dirichlet processes in Church

```
(define (DP alpha proc)
  (let ((sticks (map lambda x (beta 1.0 alpha)))))
  (lambda (lambda (lambda x (proc))))
  (lambda () (lambda (pick-a-stick sticks))))
```

Hidden Markov models in Venture

```
(defun (hmms (em (lambda arg))
  (DF alpha
    (lambda () (apply proc arg)))))
```
Semantics

- The difficulty of handling **continuous** distributions and **real numbers** spurred the invention of new semantical domains:
  - quasi-Borel spaces/predomains
    [Heunen,Kammar,Staton,Yang 17; Vákár, Kammar, Staton 21]
  - measurable cones [Ehrhard,Pagani, Tasson 17]
  - measurable space/geometry of interaction [Dal Lago, Hoshino 19]
The difficulty of handling \textit{continuous} distributions and \textit{real numbers} spurred the invention of new semantical domains:

- quasi-Borel spaces/predomains
  [Heunen,Kammar,Staton,Yang 17; Vákár, Kammar, Staton 21]
- measurable cones [Ehrhard,Pagani, Tasson 17]
- measurable space/geometry of interaction [Dal Lago, Hoshino 19]

... especially since [Jung, Tix 98] was \textit{misinterpreted} as a failure of domain theory — we will give a dcpo semantics
Exact real arithmetic

- Exact real arithmetic can be implemented in many ways
- E.g., [Boehm 87] encodes a real number $x$ by a function $n \in \mathbb{N} \mapsto x_n \in \mathbb{Z}$ such that $\left| x - \frac{x_n}{4^n} \right| \leq \frac{1}{4^n}$
Exact real arithmetic

- **Exact real arithmetic** can be implemented in many ways.
- E.g., [Boehm 87] encodes a real number \( x \) by a function \( n \in \mathbb{N} \mapsto x_n \in \mathbb{Z} \) such that \( \left| x - \frac{x_n}{4^n} \right| \leq \frac{1}{4^n} \).
- \( x \) is really \( \sup_{n \in \mathbb{N}} \left[ \frac{x_n}{4^n}, \frac{x_n + 1}{4^n} \right] \) in \( \mathbb{R}_\perp \), as we now demonstrate...

Eg., compute \( \pi \) by Machin’s formula

\[
\frac{\pi}{4} = 4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)
\]

Let us compute \( \frac{\pi}{7} \)...

What is \( \pi - 7 \times \left(\frac{\pi}{7}\right) \)?
Exact real arithmetic

- **Exact real arithmetic** can be implemented in many ways.

- E.g., [Boehm 87] encodes a real number $x$ by a function $n \in \mathbb{N} \mapsto x_n \in \mathbb{Z}$ such that $\left| x - \frac{x_n}{4^n} \right| \leq \frac{1}{4^n}$.

- $x$ is really $\sup_{n \in \mathbb{N}} \left[ \frac{x_n}{4^n}, \frac{x_n + 1}{4^n} \right]$ in $\mathbb{R}_\bot$, as we now demonstrate...

---

E.g., compute $\pi$ by Machin’s formula:
\[
\frac{\pi}{4} = 4 \arctan \left( \frac{1}{5} \right) - \arctan \left( \frac{1}{239} \right)
\]

Let us compute $\frac{\pi}{7}$...

What is $\pi - 7 \times \left( \frac{\pi}{7} \right)$?

No rounding error!
Lebesgue is implementable

- Remember $\lambda_{[0,1]}$?
- This is also implementable, as a **program** that draws exact real numbers at random in $[0,1]$...

Beyond simple valuations

- Maybe we only need **simple** valuations in computer science?
- No: simple valuations do not form a **dcpo**
- E.g., $\lambda_{[0,1]} = \sup_{n \in \mathbb{N}} \lambda_n$ (uniform measure on $[0,1]$), where $\lambda_n \triangleq \sum_{i=1}^{2^n} \frac{1}{2^n} \delta_{\frac{i-1}{2^n} \cdot \frac{1}{2^n}}$

  (there is a similar formula for $\lambda$ itself, but I wish to show you a probability valuation)
Lebesgue is implementable

- Remember $\lambda_{|[0,1]}$?
- This is also implementable, as a program that draws exact real numbers at random in $[0,1]$...

Beyond simple valuations

- Maybe we only need simple valuations in computer science?
- No: simple valuations do not form a depo
- E.g., $\lambda_{|[0,1]} = \sup_{\alpha \in \mathbb{R}} \lambda_{[\alpha]}$ (uniform measure on $[0,1]$), where $\lambda_{[\alpha]} = \sum_{j=1}^{\infty} \frac{1}{2^j} [2^{-j}\cdot \alpha, 2^{-j}\cdot(\alpha+1)]$

(there is a similar formula for $\lambda$ itself, but I wish to show you a probability valuation)
Lebesgue is implementable

- Remember $\overline{\lambda}_{[0,1]}$?
- This is also implementable, as a **program** that draws exact real numbers at random in $[0,1]$...

### Beyond simple valuations

- Maybe we only need simple valuations in computer science?
- No: simple valuations do not form a **depo**
- E.g., $\overline{\lambda}_{[0,1]} = \sup_{\text{depo}} \overline{\lambda}_P$ (uniform measure on $[0,1]$),

where $\overline{\lambda}_P = \sum_{j=1}^{\infty} \frac{1}{2^j} \delta_{2^j \cdot p}$

(There is a similar formula for $\overline{\lambda}$ itself, but I wish to show you a probability valuation)
Interval Statistical PCF (ISPCF), v1

\[ M, N, P, \ldots ::= \ldots \quad \text{(as in PCF)} \]
\[ \mid \text{ret } M \quad \text{monad unit} \]
\[ \mid \text{do } x_\sigma = M; N \quad \text{sequential composition} \]
\[ \mid M \oplus N \quad \text{probabilistic choice} \quad \text{(subsumed by sample\([0,1])} \]
\[ \mid \text{sample}[0,1] \quad (\bar{\lambda\}_{[0,1]}) \]
\[ \diamond \]
\[ \mid r \quad \text{(real constants, } r \in \mathbb{R} \text{)} \]
\[ \mid f(M_1, \ldots, M_n) \quad (f \in \{ +, -, >, \ldots \}) \]

\diamond \text{Types: } \sigma, \tau, \ldots ::= \text{nat} | \text{unit} | \text{real} | \sigma \rightarrow \tau | T\tau
Interval Statistical PCF (ISPCF), v1

\[ M, N, P, \ldots ::= \ldots \]  
\[ | \text{ret } M \]  
\[ | \text{do } x_\sigma = M; N \]  
\[ | M \oplus N \]  
\[ | \text{sample}[0,1] \]  
\[ | r \]  
\[ | f(M_1, \cdots, M_n) \]  
\[ \ast \text{Types: } \sigma, \tau, \ldots ::= \text{nat} | \text{unit} | \text{real} | \sigma \to \tau | T\tau \]  

\[(\text{as in PCF})\]

\[\text{ret } M \]  
\[\text{do } x_\sigma = M; N \]  
\[M \oplus N \]  
\[\text{sample}[0,1] \]  
\[r \]  
\[f(M_1, \cdots, M_n) \]

\[\text{monad unit}\]
\[\text{sequential composition}\]
\[\text{probabilistic choice}\]
[Distribution on exact reals]

\[(\text{subsumed by } \text{sample}[0,1])\]
Interval Statistical PCF (ISPCF), v1

\[ M, N, P, \ldots ::= \ldots \]  
\[ \text{as in PCF} \]

\[ | \text{ret } M \quad \text{ret } M \text{ monad unit} \]
\[ | \text{do } x_\sigma = M; N \quad \text{do } x_\sigma = M; N \text{ sequential composition} \]
\[ | M \oplus N \quad \text{M \oplus N probabilistic choice} \]
\[ | \text{sample}[0,1] \quad \text{sample}[0,1] \text{ (\( \tilde{\alpha}_{[0,1]} \))} \]
\[ \star \]
\[ | r \quad \text{r (real constants, } r \in \mathbb{R} \text{)} \]
\[ | f(M_1, \cdots, M_n) \quad (f \in \{ +, -, >, \cdots \}) \]

\[ \star \text{ Types: } \sigma, \tau, \ldots ::= \text{nat} | \text{unit} | \boxed{\text{real}} | \sigma \rightarrow \tau | T\tau \]

Distribution on exact reals

Exact real arithmetic

(subsumed by \text{sample}[0,1])
The denotational semantics of ISPCF v1

- $\llbracket \text{nat} \rrbracket \triangleq \mathbb{N}_\bot$, $\llbracket \text{unit} \rrbracket \triangleq \{ \bot, * \}$, $\llbracket \sigma \to \tau \rrbracket \triangleq [\llbracket \sigma \rrbracket] [\llbracket \tau \rrbracket]$, $\llbracket T \tau \rrbracket \triangleq V_{\leq 1}[\tau]$ as before

- $\llbracket \text{real} \rrbracket \triangleq \mathbb{IR}_\bot$

- $\llbracket \text{sample}[0,1] \rrbracket \rho \triangleq \tilde{\lambda}_{[0,1]}$

- $\llbracket r \rrbracket \rho \triangleq [r, r]$

- $\llbracket f(M_1, \ldots, M_n) \rrbracket \rho \triangleq \tilde{f}(\llbracket M_1 \rrbracket \rho, \ldots, \llbracket M_n \rrbracket \rho)$

where $\tilde{f} \triangleq$ largest Scott-continuous map $\leq f$...
The denotational semantics of ISPCF v1

\[
\begin{align*}
\lbrack \text{nat} \rbrack & \triangleq \mathbb{N}_\perp, \lbrack \text{unit} \rbrack \triangleq \{ \perp, * \}, \lbrack \sigma \to \tau \rbrack \triangleq [\lbrack \sigma \rbrack \to [\tau \rbrack], \\
[\mathbf{T}\tau] & \triangleq \mathbf{V}_{\leq 1} [\tau \rbrack \text{ as before }
\end{align*}
\]

\[
\begin{align*}
\lbrack \text{sample}[0,1] \rbrack \rho & \triangleq \lambda_{[0,1]} \\
[\mathbf{r}] \rho & \triangleq [r, r] \\
[\mathbf{f}(M_1, \ldots, M_n)] \rho & \triangleq \mathbf{\check{f}}(\lbrack M_1 \rbrack \rho, \ldots, [M_n] \rho) \\
\text{where } \mathbf{\check{f}} & \triangleq \text{largest Scott-continuous map } \leq f\ldots
\end{align*}
\]
The extension theorem for bc-domains

**Theorem.** For every (arbitrary) function $f : X \to Z$ from a topological space $X$ to a bc-domain $Z$, there is a **largest continuous** map $\tilde{f} : X \to Z$ such that $\forall x \in X, \tilde{f}(x) \leq f(x)$.

**Proof.** $\tilde{f}(x) = \sup \{z \in Z \mid x \in \text{int}(f^{-1}(\uparrow z))\} = \sup \{\inf_U f \mid U \in \mathcal{O}X \text{ s.t. } x \in U\}$. The sup exists because $Z$ is a bc-domain. $\square$
The extension theorem for bc-domains

**Theorem.** For every (arbitrary) function \( f: X \rightarrow Z \) from a topological space \( X \) to a bc-domain \( Z \), there is a **largest continuous** map \( \tilde{f}: X \rightarrow Z \) such that \( \forall x \in X, \tilde{f}(x) \leq f(x) \).

**Proof.** \( \tilde{f}(x) = \sup \{ z \in Z \mid x \in \text{int}(f^{-1}(\uparrow z)) \} \) (\( = \sup \{ \inf_U f \mid U \in \mathcal{O}X \text{ s.t. } x \in U \} \)).

The sup exists because \( Z \) is a bc-domain. \( \square \)
Theorem. For every (arbitrary) function $f: X \to Z$ from a topological space $X$ to a bc-domain $Z$, there is a largest continuous map $\hat{f}: X \to Z$ such that $\forall x \in X, \hat{f}(x) \leq f(x)$.

Proof. $\hat{f}(x) = \sup\{z \in Z \mid x \in \text{int}(f^{-1}( \uparrow z))\}$ ($= \sup\{\inf_U f \mid U \in \mathcal{O}X \text{ s.t. } x \in U\}$).

The sup exists because $Z$ is a bc-domain. □

$\hat{f}$ maps $([a, b], [c, d])$ to $([a + c, b + d])$.
The extension theorem for bc-domains

- **Theorem.** For every (arbitrary) function \( f: X \to Z \) from a topological space \( X \) to a bc-domain \( Z \), there is a largest continuous map \( \tilde{f}: X \to Z \) such that \( \forall x \in X, \tilde{f}(x) \leq f(x) \).

  \[ \tilde{f}(x) = \sup\{z \in Z \mid x \in \text{int}(f^{-1}( \uparrow z))\} = \sup\{\inf_U f | U \in \mathcal{O}X \text{ s.t. } x \in U\} \]

  The sup exists because \( Z \) is a bc-domain. \( \square \)

- \( \tilde{+} \) maps \([a, b], [c, d]\) to \([a + c, b + d]\)
- \( \tilde{\text{sgn}} \) maps \([a, b]\) to \(-1\) if \( b < 0 \), \(+1\) if \( a > 0 \), \( \bot \) otherwise
The extension theorem for bc-domains

Theorem. For every (arbitrary) function \( f : X \to Z \)
from a topological space \( X \) to a bc-domain \( Z \),
there is a largest continuous map \( \tilde{f} : X \to Z \)
such that \( \forall x \in X, \tilde{f}(x) \leq f(x) \).

Proof. \( \tilde{f}(x) = \sup \{ z \in Z \mid x \in \text{int}(f^{-1}(\uparrow z)) \} \)
(\( = \sup \{ \inf_U f \mid U \in \mathcal{O}X \text{ s.t. } x \in U \} \)).
The sup exists because \( Z \) is a bc-domain. \( \square \)

\( \tilde{+} \) maps \([a, b], [c, d]\) to \([a + c, b + d]\)
\( \tilde{\text{sgn}} \) maps \([a, b]\) to \(-1\) if \( b < 0 \), \(+1\) if \( a > 0 \), \( \bot \) otherwise

\( \bot \) here is unavoidable: \( \text{sgn} \) is best possible
(alternatively, \( \mathbb{R} \) is connected, so every non-trivial continuous predicate on \( \mathbb{R} \) is constant)
The extension theorem for bc-domains

**Theorem.** For every (arbitrary) function $f: X \to Z$ from a topological space $X$ to a bc-domain $Z$, there is a largest continuous map $\tilde{f}: X \to Z$ such that $\forall x \in X, \tilde{f}(x) \leq f(x)$.

**Proof.** $\tilde{f}(x) = \sup\{z \in Z \mid x \in \text{int}(f^{-1}(\uparrow z))\} = \sup\{\inf_{U} f \mid U \in \mathcal{O}X \text{ s.t. } x \in U\}$. The sup exists because $Z$ is a bc-domain. □

- $\tilde{+}$ maps $([a, b], [c, d])$ to $([a + c, b + d])$
- $\tilde{\text{sgn}}$ maps $[a, b]$ to $-1$ if $b < 0$, $+1$ if $a > 0$, $\bot$ otherwise
- $\tilde{>}$ maps $([a, b], [c, d])$ to true (0) if $a > d$, false (1) if $b < c$, $\bot$ otherwise

A bc-domain is a continuous dcpo that:
- is pointed (has a $\bot$)
- any two elements $x, y$ with an upper bound have a least upper bound $x \lor y$

$\mathbb{N}_1$ is a bc-domain
- not $\mathbb{R}$ (no $\bot$)

$\mathbb{R}_1$ is a bc-domain

$\bot$ here is unavoidable: $\text{sgn}$ is best possible (alternatively, $\mathbb{R}$ is connected, so every non-trivial continuous predicate on $\mathbb{R}$ is constant)
The operational semantics of ISPCF v1

Exploration rules

\[ C, \text{do } x = M; N \rightarrow C[\text{do } x = _{\_}; N], M \]
\[ _{\_}, \text{ret } M \rightarrow \text{ret } _{\_}, M \]

Computation rules

\[ C[\text{do } x = _{\_}; N], \text{ret } M \rightarrow C, N[x := M] \]
\[ C[\text{sample}[0,1]] \rightarrow C[\text{ret } r] \]
\[ C[f(r_1, \ldots, r_n)] \rightarrow C[f(r_1, \ldots, r_n)] \quad (r \leftarrow \text{rand } \lambda_{[0,1]}) \]

❖ Theorem (soundness, adequacy).
\[ \Pr[C, M \downarrow n] = \llbracket C[M] \rrbracket_{\rho} \]
(at type \textbf{nat})
The operational semantics of ISPCF v1

Exploration rules

\[ C, \text{do } x = M; N \rightarrow C[\text{do } x = _\_; N], M \]
\[ _, \text{ret } M \rightarrow \text{ret } _, M \]

Computation rules

\[ C[\text{do } x = _\_; N], \text{ret } M \rightarrow C, N[x := M] \]
\[ C[\text{sample}[0,1]] \rightarrow C[\text{ret } r] \]
\[ C[f(r_1, \ldots, r_n)] \rightarrow C[f(r_1, \ldots, r_n)] \]

\( (r \leftarrow \text{rand } \lambda_{[0,1]} ) \)

❖ Theorem (soundness, adequacy).

Pr\[C, M \downarrow n] = \llbracket C[M] \rrbracket_\rho \]
(at type \text{nat})

Transition relation is now really a Markov kernel on a topological space of configurations:

\[ \text{Pr}[C[\text{sample}[0,1]] \rightarrow _\_ \in U] = \lambda(\{r \in [0,1] \mid C[\text{ret } r] \in U\}) \]
\[ \text{Pr}[C[f(r_1, \ldots, r_n)] \rightarrow _\_ \in U] = \delta_{C[f(r_1, \ldots, r_n)]}(U) \]
if \( f(r_1, \ldots, r_n) \) defined
\[ = 0 \quad \text{otherwise} \]
The operational semantics of ISPCF v1

Exploration rules

\[ C, \text{do } x = M; N \rightarrow C[\text{do } x = _-; N], M \]
\[ _, \text{ret } M \rightarrow \text{ret } _, M \]

Computation rules

\[ C[\text{do } x = _-; N], \text{ret } M \rightarrow C, N[x := M] \]
\[ C[\text{sample}[0,1]] \rightarrow C[\text{ret } r] \]
\[ C[f(r_1, \ldots, r_n)] \rightarrow C[f(r_1, \ldots, r_n)] \]

\( r \leftarrow \text{rand } \lambda_{[0,1]} \)

Transition relation is now really a Markov kernel on a topological space of configurations:

\[ \Pr[C, M \downarrow n] = \llbracket C[M] \rrbracket \rho \]

(at type nat)

Theorem (soundness, adequacy).

Pr[\( C, M \downarrow n \)] = \llbracket C[M] \rrbracket \rho

Transition relation is now really a Markov kernel on a topological space of configurations:

\[ \Pr[C[\text{sample}[0,1]] \rightarrow _- \in U] = \lambda(\{ r \in [0,1] \mid C[\text{ret } r] \in U \}) \]
\[ \Pr[C[f(r_1, \ldots, r_n)] \rightarrow _- \in U] = \delta_{C[f(r_1, \ldots, r_n)]}(U) \]
\[ = 0 \quad \text{if } f(r_1, \ldots, r_n) \text{ defined} \]
\[ = \text{otherwise} \]

with actual real numbers (in \( \mathbb{R} \))
not simply exact real numbers (in \( \mathbb{R} \))
What does this compute?

```ocaml
goubleal@macbook-pro-de-jean Topics2023 % gmm
GimML comes with ABSOLUTELY NO WARRANTY (See file COPYRIGHT).
GimML for Darwin 20.5.0, by Jean Goubault-Larrecq (c) 2021.
> use "ispcf1.ml"
> 
```

```

longest_decreasing_run \triangleq \text{rec}(\lambda f_{\text{real}} \to \text{int} \cdot \lambda x_{\text{real}} \cdot \lambda n_{\text{int}} \cdot \\
\hspace{1cm} \text{do } u = \text{sample}[0,1]; \\
\hspace{1cm} \text{if } u > x \text{ then } \text{ret } n \\
\hspace{1.0cm} \text{else } u \ (s(n)))
```

```ocaml
Von_Neumann \triangleq \text{do } x = \text{sample}[0,1]; \\
\hspace{1cm} \text{rec}(\lambda f_{\text{real}} \to \text{T real} \cdot \lambda \ell_{\text{real}} \cdot \\
\hspace{2.0cm} \text{do } n = \text{longest_decreasing_run } x \ 0; \\
\hspace{2.5cm} \text{if } \text{odd } n \text{ then } f(\ell + 1.0) \\
\hspace{3.0cm} \text{else } \text{ret } \ell) \ x
```
What does this compute?

**longest_decreasing_run** \(_0 = \) length \(k\) of longest decreasing prefix
\[ x = x_0 > x_1 > \cdots > x_k \]
where \(x_1, \ldots, x_k, \ldots\) are drawn in \([0,1]\)
i.i.d. uniformly

The probability that \([\text{odd } n]\) is true here is
\[
\sum_{n \text{ odd}} \left( \frac{x^{n-1}}{(n-1)!} - \frac{x^n}{n!} \right) = \exp(-x)
\]
What does this compute?

longest_decreasing_run x 0 =
length k of longest decreasing prefix
\[ x = x_0 > x_1 > \cdots > x_k \]
where \( x_1, \ldots, x_k, \ldots \) are drawn in \([0,1]\)
i.i.d. uniformly

The probability that \([\text{odd } n]\) is true here
is
\[ \sum_{n \text{ odd}} \left( \frac{x^{n-1}}{(n-1)!} - \frac{x^n}{n!} \right) = \exp(-x) \]

\[ \star \] the exponential distribution \( \exp(-x) \) \( d\lambda \)
[von Neumann 49]
What does this compute?

```
longest_decreasing_run x 0 = length k of longest decreasing prefix
  x = x₀ > x₁ > ⋯ > xₖ
where x₁, …, xₖ, … are drawn in [0,1]
i.i.d. uniformly
```

The probability that [odd n] is true here

\[
\sum_{n \text{ odd}} \left( \frac{x^{n-1}}{(n-1)!} - \frac{x^n}{n!} \right) = \exp(-x)
\]

… the exponential distribution \(\exp(-x) \, d\lambda\)

[von Neumann 49]
What does this compute?

**longest_decreasing_run** $x \in \mathbb{N}$ = length $k$ of longest decreasing prefix $x = x_0 > x_1 > \cdots > x_k$ where $x_1, \ldots, x_k, \ldots$ are drawn in $[0, 1]$ i.i.d. uniformly

The probability that $[\text{odd } n]$ is true here is

$$\sum_{n \text{ odd}} \left( \frac{x^{n-1}}{(n-1)!} - \frac{x^n}{n!} \right) = \exp(-x)$$

.. the exponential distribution $\exp(-x) \, d\lambda$

[von Neumann 49]
Soft conditioning

- A new primitive: \texttt{score} : real \rightarrow T \texttt{unit}

- \texttt{score }_r « \text{multiplies probability of current execution branch by } r »
Soft conditioning

- A new primitive: \texttt{score} : \texttt{real} → \texttt{T unit}
- \texttt{score} \_ \_ « multiplies probability of current execution branch by \_ \_ »

requires going beyond subprobability measures e.g., consider \texttt{score 3.0}
Soft conditioning

- A new primitive: \texttt{score : real} \rightarrow \texttt{T unit}
- \texttt{score }_r \ « \text{ multiplies probability of current execution branch by } r \ »

\[\text{requires going beyond subprobability measures e.g., consider } \texttt{score 3.0}\]

One should then \textbf{normalize} the result. Smallish issue: normalization (= conditioning) is \textbf{not} computable [Ackermann, Freer, Roy 11]

\[\text{(just do not normalize)}\]
Soft conditioning

- A new primitive: \texttt{score : real → \texttt{T unit}}
- \texttt{score r} « multiplies probability of current execution branch by \texttt{r} »
- Conditioning on a property \( P \):
  \[ \text{if } \neg P \text{ then score 0.0} \]

requires going beyond subprobability measures e.g., consider \texttt{score 3.0}

One should then \textbf{normalize} the result
Smallish issue: normalization (= conditioning) is not computable [Ackermann, Freer, Roy 11]
(just do not normalize)
Soft conditioning

- A new primitive: **score**: \( \text{real} \to \text{T unit} \)
- **score** \( r \) « multiplies probability of current execution branch by \( r \) »
- Conditioning on a property \( P \):
  - if \( \neg P \) then score \( 0.0 \)
- **Soft** conditioning [Vákár, Kammar, Staton 21]: argument to **score** is a **density**
  (here, some Gaussian noise around the [formally incompatible] conditions \( a = 1.1, 2a = 1.9, 3a = 2.7 \))

requires going beyond subprobability measures e.g., consider **score** \( 3.0 \)

One should then **normalize** the result
Smallish issue: normalization (= conditioning) is **not** computable [Ackermann, Freer, Roy 11] (just do not normalize)

---

**Fig. 1.** Bayesian linear regression as (a) a first-order probabilistic program, (b) an informal specification, and (c) a plot of the prior and posterior distributions. Here \( N(\mu, \sigma) \) is the normal (Gaussian) distribution with density \( \text{normal-pdf}(x \mid \mu, \sigma) = (2\pi\sigma^2)^{-1/2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \) and random seed \( \text{normal-rng}(\mu, \sigma) \).
The semantics of score

- score : real → T unit
- \[[\text{score } M]\] ≜ |\([M]\)| ⋅ \(\delta_\star\),
  where |\(_\_\)| is largest continuous map : \(\mathbb{IR} \to \mathbb{R}_+\)
  below absolute value : \(\mathbb{R} \to \mathbb{R}_+\) on \(\mathbb{R}\)
  |\([a, b]\)| ≜ \(a\) if \(a > 0\), \((-b)\) if \(b < 0\), \(0\) if \(a \leq 0 \leq b\)
The semantics of \texttt{score}

- \texttt{score : real → T unit}

- $[\texttt{score } M] \triangleq |[M]| \cdot \delta_*$, where $|\_|$ is the largest continuous map : $\mathbb{IR} \rightarrow \overline{\mathbb{R}}_+$ below absolute value : $\mathbb{R} \rightarrow \overline{\mathbb{R}}_+$ on $\mathbb{R}$
  
  \[ |[a, b]| \triangleq a \text{ if } a > 0, -b \text{ if } b < 0, 0 \text{ if } a \leq 0 \leq b \]

- We must also redefine $[\texttt{T } \tau]$ as
  \[ V([\tau]) = \text{dcpo of all continuous valuations on } [\tau] \]
  (not just the subprobability valuations)
The semantics of \textbf{score}

\begin{itemize}
  \item \textbf{score} : real $\rightarrow$ T unit
  \item $\llbracket \text{score } M \rrbracket \doteq |\llbracket M \rrbracket| \cdot \delta_*$,
    where $|\_|$ is largest continuous map : $\mathbb{I}\mathbb{R} \rightarrow \mathbb{R}^+$
    below absolute value : $\mathbb{R} \rightarrow \mathbb{R}^+$ on $\mathbb{R}$
    $|[a, b]| \doteq a$ if $a > 0$, $-b$ if $b < 0$, $0$ if $a \leq 0 \leq b$
  \item We must also redefine $\llbracket T \tau \rrbracket$ as
    \begin{center}
      $V(\llbracket \tau \rrbracket) = \text{dcpo of all continuous valuations on } \llbracket \tau \rrbracket$
    \end{center}
    (not just the subprobability valuations)
  \item Soundness, adequacy… as usual [JGL, Jia 23]
    \begin{center}
      ... but operational semantics uses \textbf{unbounded} distributions
    \end{center}
\end{itemize}
Commutativity
The catch

- I once told you that we needed at least **continuous** dcpos if we aim to prove a form of full abstraction.
- ... but that leads to difficult problems.
- Then I showed you a semantics of ISPCF in (**non**-continuous) dcpos that had all sorts of nice properties.
- The catch is that...
Commutativity

- The (rather silly) catch is that we cannot prove that

\[
[	ext{do } x = M; \text{do } y = N; P] = [	ext{do } y = N; \text{do } x = M; P]
\]

(x not free in \(N\), \(y\) not free in \(M\))
Commutativity

- The (rather silly) catch is that we cannot prove that
  \[ \text{⟦do } x = M; \text{do } y = N; P \text{⟧} = \text{⟦do } y = N; \text{do } x = M; P \text{⟧} \]
  (x not free in N, y not free in M)

- although \textbf{do } x = M; \textbf{do } y = N; P \equiv \textbf{do } y = N; \textbf{do } x = M; P
  (this will be a consequence of what we will do next)

i.e., that you can draw \(x\) and \(y\) independently in any order
Commutativity

- The (rather silly) catch is that we cannot prove that
  \[
  \llbracket \text{do } x = M; \text{do } y = N; P \rrbracket = \llbracket \text{do } y = N; \text{do } x = M; P \rrbracket
  \]
  (\(x\) not free in \(N\), \(y\) not free in \(M\))

- although \(\text{do } x = M; \text{do } y = N; P \cong \text{do } y = N; \text{do } x = M; P\)
  (this will be a consequence of what we will do next)

- I.e., the \(V\) monad on \(\mathbf{Dcpo}\) is not known to be commutative
  (but it is on \(\mathbf{Top}\) and on \(\mathbf{Cont}\))
Commutativity

- The (rather silly) catch is that we cannot prove that
  \[ \boxed{\text{do } x = M; \text{do } y = N; P} = \boxed{\text{do } y = N; \text{do } x = M; P} \]
  (\(x\) not free in \(N\), \(y\) not free in \(M\))

- although \(\text{do } x = M; \text{do } y = N; P \cong \text{do } y = N; \text{do } x = M; P\)
  (this will be a consequence of what we will do next)

- I.e., the \(V\) monad on \(\text{Dcpo}\) is not known to be commutative
  (but it is on \(\text{Top}\) and on \(\text{Cont}\))

- I.e., the integral interchange (\text{Fubini-Tonelli}) theorem
  is not known to hold for continuous valuations on \(\text{Dcpo}\)

i.e., that you can draw \(x\) and \(y\) independently in any order
Commutativity

❖ The (rather silly) catch is that we cannot prove that
\[
\llbracket \text{do } x = M; \text{do } y = N; P \rrbracket = \llbracket \text{do } y = N; \text{do } x = M; P \rrbracket
\]
(x not free in \(N\), y not free in \(M\))

❖ although \(\text{do } x = M; \text{do } y = N; P \cong \text{do } y = N; \text{do } x = M; P\)
(this will be a consequence of what we will do next)

❖ I.e., the \(V\) monad on \(\text{Dcpo}\) is not known to be commutative
(but it is on \(\text{Top}\) and on \(\text{Cont}\))

❖ I.e., the integral interchange (Fubini-Tonelli) theorem
is not known to hold for continuous valuations on \(\text{Dcpo}\)

❖ I.e., that you can draw \(x\) and \(y\) independently in any order

There is a very subtle issue here … as Fubini-Tonelli holds on the larger category \(\text{Top}\) 😞
Theorem [folklore; JGL, Xia 23]. Let $X, Y$ be arbitrary topological spaces. For every lower semicontinuous map $h : X \times Y \to \overline{\mathbb{R}}_+$,

$$
\int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu
$$

for every lower semicontinuous map $h : X \times Y \to \overline{\mathbb{R}}_+$. 
Fubini-Tonelli for continuous valuations

**Theorem [folklore; JGL, Xia 23].** Let $X, Y$ be arbitrary topological spaces. 

\[
\int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu
\]

for every lower semicontinuous map $h : X \times Y \to \mathbb{R}_+$. 

Remember that lower semicontinuous maps from a dcpo to $\mathbb{R}_+$ = Scott-continuous
Theorem [folklore; JGL, Xia 23]. Let $X, Y$ be arbitrary topological spaces.

$$
\int_x \left( \int_y h(x, y) d\nu \right) d\mu = \int_y \left( \int_x h(x, y) d\mu \right) d\nu
$$

for every lower semicontinuous map $h : X \times Y \to \overline{\mathbb{R}_+}$.

Unknown. Let $X, Y$ be arbitrary dcpos.

$$
\int_x \left( \int_y h(x, y) d\nu \right) d\mu = \int_y \left( \int_x h(x, y) d\mu \right) d\nu
$$

for every Scott-continuous map $h : X \times Y \to \overline{\mathbb{R}_+}$.
Fubini-Tonelli for continuous valuations

- **Theorem [folklore; JGL, Xia 23]**. Let $X, Y$ be arbitrary topological spaces.

\[ \int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu \]

for every lower semicontinuous map $h : X \times Y \to \mathbb{R}_+$

- **Unknown**. Let $X, Y$ be arbitrary dcpos.

\[ \int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = \text{?} \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu \]

for every Scott-continuous map $h : X \times Y \to \mathbb{R}_+$

Remember that lower semicontinuous maps from a dcpo to $\mathbb{R}_+$ are Scott-continuous.

The issue: products differ in **Top** and in **Dcpo** (the $\times$ symbol is ambiguous).
Fubini-Tonelli for continuous valuations

- **Theorem [folklore; JGL, Xia 23].** Let $X$, $Y$ be arbitrary topological spaces.
  \[ \int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu \]
  for every lower semicontinuous map $h : X \times Y \to \mathbb{R}_+$

- **Unknown.** Let $X$, $Y$ be arbitrary dcpos.
  \[ \int_x \left( \int_y h(x, y) \, d\nu \right) \, d\mu = ? \int_y \left( \int_x h(x, y) \, d\mu \right) \, d\nu \]
  for every Scott-continuous map $h : X \times Y \to \mathbb{R}_+$

Remember that lower semicontinuous maps from a dcpo to $\mathbb{R}_+$ are Scott-continuous.

The issue: products differ in **Top** and in **Dcpo** (the $\times$ symbol is ambiguous).
Theorem [folklore; JGL, Xia 23]. Let $X, Y$ be arbitrary topological spaces. 
\[
\int_x \left( \int_y h(x, y) d\nu \right) d\mu = \int_y \left( \int_x h(x, y) d\mu \right) d\nu
\]
for every lower semicontinuous map $h : X \times Y \to \mathbb{R}_+$

Unknown. Let $X, Y$ be arbitrary dcpos. 
\[
\int_x \left( \int_y h(x, y) d\nu \right) d\mu = \int_y \left( \int_x h(x, y) d\mu \right) d\nu
\]
for every Scott-continuous map $h : X \times Y \to \mathbb{R}_+$

Remember that lower semicontinuous maps from a dcpo to $\mathbb{R}_+$ are Scott-continuous.

The issue: products differ in Top and in Dcpo (the $\times$ symbol is ambiguous).
Products in Top, products in Dcpo

- **In Top**: $X \times Y$ has the **product topology**, open sets = unions of open rectangles $U \times V$, $U \in \mathcal{O}X$, $V \in \mathcal{O}Y$

- **In Dcpo**: $X \times Y$ has the **Scott topology** of $\leq \times \leq$

Diagram:

- dcpos $X, Y$ \rightarrow \text{spaces} $X_\sigma, Y_\sigma$
- form product in Dcpo $X \times Y$ \rightarrow dcpo $X_\sigma \times Y_\sigma$
- form product in Top $(X \times Y)_\sigma$
- take Scott topology  \rightarrow \text{space} $(X \times Y)_\sigma$
- take Scott topologies \rightarrow \text{spaces} $X_\sigma, Y_\sigma$
- form product in Dcpo $X, Y$ \rightarrow dcpos $X, Y$
Products in *Top*, products in *Dcpo*

- In *Top*: \( X \times Y \) has the **product topology**, open sets = unions of **open rectangles** \( U \times V, U \in \mathcal{O}X, V \in \mathcal{O}Y \)

- In *Dcpo*: \( X \times Y \) has the **Scott topology** of \( \leq \times \leq \)

\[
\begin{align*}
\text{dcpos} & \quad \text{topologies} & \quad \text{spaces} \\
X, Y & \quad \longrightarrow & \quad X_\sigma, Y_\sigma \\
\text{form product in *Dcpo*} & \quad \quad & \quad \text{form product in *Top*} \\
dcpo & \quad \quad & \quad \quad \\
X \times Y & \quad \quad & \quad X_\sigma \times Y_\sigma \\
take Scott topology & \quad \quad & \quad \quad \\
& \quad \quad & \quad (X \times Y)_\sigma
\end{align*}
\]

Note: \( X_\sigma \times Y_\sigma = (X \times Y)_\sigma \) if \( X \) or \( Y \) continuous *dcpo*
Products in **Top**, products in **Dcpo**

- **In Top**: $X \times Y$ has the **product topology**, open sets = unions of **open rectangles** $U \times V$, $U \in \mathcal{O}X$, $V \in \mathcal{O}Y$
- **In Dcpo**: $X \times Y$ has the **Scott topology** of $\leq \times \leq$

![Diagram showing the relationship between topological and dcpo products](image)

Note: $X_\sigma \times Y_\sigma = (X \times Y)_\sigma$ if $X$ or $Y$ continuous dcpos
Products in Top, products in Dcpo

- In Top: \(X \times Y\) has the **product topology**, open sets = unions of open rectangles \(U \times V, U \in \mathcal{O}X, V \in \mathcal{O}Y\)

- In Dcpo: \(X \times Y\) has the **Scott topology** of \(\leq \times \leq\)

Note: \(X_\sigma \times Y_\sigma = (X \times Y)_\sigma\) if \(X\) or \(Y\) continuous dcpo

More generally:
- if \(X_\sigma\) or \(Y_\sigma\) is core-compact [Gierz,Hofmann,Keimel,Lawson,Mislove 03]
- if \(X_\sigma\) and \(Y_\sigma\) are first-countable [de Brecht, priv. comm., 19]

Diagram:

- Form product in Dcpo
- Dcpos \(X, Y\) take Scott topologies \(X_\sigma, Y_\sigma\)
- Form product \(X_\sigma \times Y_\sigma\)
- Form product in Top
- \((X \times Y)_\sigma\)

Note:

\[\begin{align*}
\text{Fubini-Tonelli holds on } \mathcal{C} \text{ont}\end{align*}\]
Minimal valuations

- Let $V_{\text{fin}}X$ be the subset of $VX$ consisting of simple valuations.
- The smallest sub-dcpo $M_X$ of $VX$ containing $V_{\text{fin}}X$ is the dcpo of minimal valuations.
Minimal valuations

- Let $V_{\text{fin}}X$ be the subset of $VX$ consisting of simple valuations
- The smallest subdcpo $MX$ of $VX$ containing $V_{\text{fin}}X$ is the dcpo of \textit{minimal valuations}
- Explicitly, a minimal valuation is a directed supremum of directed suprema of ... of simple valuations (iterated transfinitely)
Minimal valuations

- Let $V_{\text{fin}}X$ be the subset of $VX$ consisting of simple valuations.
- The smallest subdcpo $MX$ of $VX$ containing $V_{\text{fin}}X$ is the dcpo of minimal valuations.
- Explicitly, a minimal valuation is a directed supremum of directed suprema of ... of simple valuations (iterated transfinitely).
- Prop [Jia, Lindenhovius, Mislove, Zamdzhiiev 21; JGL, Jia 23]. Fubini-Tonelli holds on Dcpo if one of the valuations is minimal.
- Proof sketch: Integration commutes with directed suprema. This reduces the question to the case of simple valuations, where commutation is easy.
ISPCF, v2

- $[\text{nat}] \triangleq \mathbb{N}_\bot$, $[\text{unit}] \triangleq \{ \bot, * \}$, $[\sigma \rightarrow \tau] \triangleq [[\sigma] \rightarrow [[\tau]]$, $[\text{real}] \triangleq \mathbb{R}_\bot$
- $[[T\tau]] \triangleq M[[\tau]]$

- $[\text{sample}[0,1]] \rho \triangleq \overline{\lambda}_{[0,1]}$
- $[[r]] \rho \triangleq [r, r]$
- $[[f(M_1, \ldots, M_n)] \rho \triangleq \check{f}([[M_1] \rho, \ldots, [[M_n]] \rho)]$

Instead of $V[\tau]$
(this is the only change!)
ISPCF, v2

- \([\text{nät}] \equiv \mathbb{N}_{\bot}, [\text{unit}] \equiv \{ \bot, * \}, [\sigma \rightarrow \tau] \equiv [[\sigma] \rightarrow [[\tau]], [\text{real}] \equiv \mathbb{IR}_{\bot}\]
  \([T\tau] \equiv M[\tau]\]

- \([\text{sample}[0,1]]\rho \equiv \overline{\lambda}_{[0,1]}\]
  \([r]\rho \equiv [r, r]\]
  \([f(M_1, \cdots, M_n)]\rho \equiv f([M_1]\rho, \cdots, [M_n]\rho)\]

Instead of \(V[\tau]\)
(this is the only change!)

That is a minimal valuation
ISPCF, v2

- $[\text{nat}] \triangleq \mathbb{N}_\bot$, $[\text{unit}] \triangleq \{ \bot, * \}$, $[\sigma \to \tau] \triangleq [[\sigma] \to [[\tau]]]$, $[\text{real}] \triangleq \mathbb{IR}_\bot$

- $[\text{sample}[0,1]]\rho \triangleq \lambda_{[0,1]}$

- $[r]\rho \triangleq [r, r]$

- $[[f(M_1, \ldots, M_n)]\rho \triangleq f([[M_1]\rho, \ldots, [[M_n]\rho])$

- Theorem (soundness, adequacy).

\[ \Pr[C, M \downarrow n] = [[C[M]]]\rho \]

(at type $\text{nat}$)

- Operational semantics is unchanged

- Instead of $V[\tau]$

  (this is the only change!)

- That is a minimal valuation

- Beyond simple valuations

  * Maybe we only need simple valuations in computer science?
  * No: simple valuations do not form a set
  * E.g., $E_{[0,1]} = \sup_{\mu\in\mathcal{P}[0,1]} \mathcal{L}_\mu$ (uniform measure on [0,1]),
  where $\mathcal{L}_\mu \triangleq \sum_{i=0}^{\mu} \mathcal{L}_{\mu_i}^{[0,\lambda]}$ (there is a similar formula for $E_{[1]}$),

  but I wish to show you a probability valuation
ISPCF, v2

- $[\text{nat}] \triangleq \mathbb{N}_\bot$, $[\text{unit}] \triangleq \{ \bot, * \}$, $[\sigma \to \tau] \triangleq \{[\sigma] \to [\tau]\}$, $[\text{real}] \triangleq \mathbb{R}_\bot$.
- $[T\tau] \triangleq M[T\tau]$,
- $[\text{sample}[0,1]]\rho \triangleq \bar{\lambda}_{[0,1]}$,
- $[[r]]\rho \triangleq [r, r]$, $[[f(M_1, \ldots, M_n)]\rho] \triangleq f([[M_1]\rho, \ldots, [M_n]\rho])$.

### Theorem (soundness, adequacy).
Pr$[C, M \downarrow n] = [[C[M]]]\rho$
(at type $\text{nat}$)

- $[[\text{do } x = M; \text{do } y = N; P]]$
  $= [[\text{do } y = N; \text{do } x = M; P]]$
  (x not free in N, y not free in M)

Operational semantics is unchanged.
ISPCF, v2

- \([\text{nat}] \doteq \mathbb{N}_{\bot} \), \([\text{unit}] \doteq \{ \bot, * \} \), \([\sigma \to \tau] \doteq [[\sigma] \to [[\tau]]] \), \([\text{real}] \doteq \mathbb{R}_\bot\]
  - \([^T\tau] \doteq M[[\tau]]\]
- \([\text{sample}[0,1]]_\rho \doteq \overline{\lambda}_{[0,1]} \]
  - \([r]_\rho \doteq [r, r] \]
  - \([f(M_1, \ldots, M_n)]_\rho \doteq f([M_1]_\rho, \ldots, [M_n]_\rho)\]

- Theorem (soundness, adequacy).
  - \(\Pr[C, M \downarrow n] = [[C[M]]]_\rho\)
    - (at type \text{nat})

- Operational semantics is unchanged

- Instead of \(V[\tau]\)
  - (this is the only change!)

- That is a minimal valuation

- That is a minimal valuation

- \([[\text{do } x = M; \text{do } y = N; P}]\]
  - \(\doteq [[\text{do } y = N; \text{do } x = M; P}]\)
    - (x not free in N, y not free in M)

- Problem solved.
  - In particular, \(\text{do } x = M; \text{do } y = N; P\)
    - \(\cong \text{ do } y = N; \text{ do } x = M; P\)
Hierarchies of continuous valuations

Simple valuations $\sum_{i=1}^{n} a_i \delta_{x_i}$ ⊆ Minimal valuations dcpo-closure of simple valuations ⊆ Point-continuous valuations sobrification of simple valuations [Heckmann 97] ⊆ Central valuations center of the $\mathbf{V}$ monad on Dcpo [Jia, Mislove, Zamzhev 21] ⊆ Continuous valuations [SahebDjahromi 80, Jones 90]

All those satisfy Fubini-Tonelli on Dcpo (they form commutative monads on Dcpo)
Hierarchies of continuous valuations

- Simple valuations
  \[ \sum_{i=1}^{n} a_i \delta_{x_i} \]

- Minimal valuations: dcpo-closure of simple valuations
- Point-continuous valuations: sobrification of simple valuations
  [Heckmann 97]
- Central valuations: center of the \( V \) monad on \( \text{Dcpo} \)
  [Jia, Mislove, Zamzhev 21]
- Continuous valuations
  [SahebDjahromi 80, Jones 90]

All those satisfy Fubini-Tonelli on \( \text{Dcpo} \)

- Is any of those inclusions strict?
- If every continuous valuation on a dcpo is minimal / point-continuous / central, then we would obtain Fubini-Tonelli on \( \text{Dcpo} \).
Hierarchies of continuous valuations

Simple valuations
\[ \sum_{i=1}^{n} a_i \delta_{x_i} \]  \( \subseteq \)  \[ \text{dcpo-closure of simple valuations} \]

Minimal valuations

Point-continuous valuations
sobrification of simple valuations
[Heckmann 97]

Central valuations
center of the \( \mathbf{V} \) monad
on \( \mathbf{Dcpo} \)
[Jia, Mislove, Zamdzhiev 21]

Continuous valuations
[ShahbDjahromi 80, Jones 90]

Beyond simple valuations

* Maybe we only need simple valuations in computer science?
* No: simple valuations do not form a dcpo
* E.g., \( \tilde{x}_{[0,1]} = \sup_{x \in [0,1]} \tilde{x} \) (uniform measure on \([0,1]\)),
where \( \tilde{x} = \sum_{\alpha \in \mathcal{A}} \frac{1}{\alpha} \lambda_{\alpha}^{\mathcal{A}} \)
(there is a similar formula for \( \mathcal{N} \))

[This slide was modified from the original to better fit the context.]
Hierarchies of continuous valuations

Simple valuations
\[ \sum_{i=1}^{n} a_i \delta_{x_i} \]

Minimal valuations \( \subseteq \) Point-continuous valuations \( \subseteq \) Central valuations \( \subseteq \) Continuous valuations

\[ \sum_{i=1}^{n} a_i \delta_{x_i} \]

dcpo-closure of simple valuations

Proof: pretty elaborate, uses that every bounded discrete valuation extends to a measure [Alvarez-Manilla, Edalat, Saheb-Djahromi 00], plus a detailed analysis of valuations of the form \( \theta + r \cdot \mu \) with \( \theta \) bounded discrete. \( \square \)

\[ \sum_{i=1}^{n} a_i \delta_{x_i} \]

Theorem [JGL, Jia 21]. On Johnstone’s dcpo \( J \),
— every continuous valuation is point-continuous
— there is a continuous valuation \( \mu \) that is **not minimal**.

\[ \sum_{i=1}^{n} a_i \delta_{x_i} \]

The Johnstone dcpo \( J \)
- Johnstone’s dcpo \( J \) (1981): a well-known non-sober dcpo
  - Points = pairs \((m, n)\) in \( \mathbb{N} \times (\mathbb{N} \cup \{\omega\}) \)
  - \((m, n) \leq (m', n')\) iff
    - \( m = m' \) and \( n \leq n' \)
    - or \( n \leq m' \) and \( n' = \omega \)

A funny valuation on \( J \)
- On Johnstone’s dcpo \( J \), there is a continuous valuation \( \mu \) defined by:
  \[ \mu(U) = 1 \] for every non-empty Scott-open set \( U \)
  \[ \mu(\emptyset) = 0 \]
- Modularity \( \mu(U \cup V) + \mu(U \cap V) = \mu(U) + \mu(V) \)
  comes from the fact that \( J \) is **hyperconnected**:
  any two non-empty open sets intersect.
  (Check it! Observe that every non-empty open set contains all points \((m, n)\) for \( m \) large enough.)
- We will show that \( \mu \) is not minimal.
Hierarchies of continuous valuations

The Sorgenfrey line $\mathbb{R}_\ell$

- A famous counterexample in topology:
  - Sorgenfrey topology on $\mathbb{R}$ generated by basic open sets $[a, b[$ (topology of convergence from the right)

<table>
<thead>
<tr>
<th>Nice</th>
<th>Not so nice</th>
</tr>
</thead>
<tbody>
<tr>
<td>paracompact, T_4</td>
<td>product with itself not T_4</td>
</tr>
<tr>
<td>zero-dimensional</td>
<td>not locally compact</td>
</tr>
<tr>
<td>Choquet-complete, hence Baire</td>
<td>not convex</td>
</tr>
<tr>
<td>first-countable, with countable dense subset (Q)</td>
<td>not second-countable</td>
</tr>
<tr>
<td>hereditarily Lindelöf</td>
<td>completely quasi-metrizable</td>
</tr>
</tbody>
</table>

The Smyth powerdomain of $\mathbb{R}_\ell$

- Proposition. $\mathcal{G}\mathbb{R}_\ell$ is a dcpo model of $\mathbb{R}_\ell$:
  - through $x \mapsto \{x\}$, $\mathbb{R}_\ell$ embeds as a topological subspace of $\mathcal{G}\mathbb{R}_\ell$
  - $\mathbb{R}_\ell$ is the intersection $\mathcal{U} \cap \mathbb{R}_\ell$
  - where $\mathcal{U}$ is Scott-open in $\mathcal{G}\mathbb{R}_\ell$

- The Lebesgue valuation $\lambda$ on $\mathbb{R}_\ell$ induces a continuous valuation $\bar{\lambda}$ on $\mathcal{G}\mathbb{R}_\ell$ by
  \[ \bar{\lambda}(\mathcal{U}) = \lambda(\mathcal{U} \cap \mathbb{R}_\ell) \]

Point-continuous valuations

- sobrification of simple valuations [Heckmann 97]
- $\subseteq$ Central valuations [SahebDjahromi 80, Jones 90]
- $\subseteq$ Continuous valuations

Center of the $\mathcal{V}$ monad on Dcpo

Theorem [JGL, Jia 21]. On the dcpo $\mathcal{Q}(\mathbb{R}_\ell)$, $\bar{\lambda}$ is not point-continuous.

Proof: pretty involved, uses Adamski’s theorem, results by [He, Li, Xi, Zhao 19] on the Smyth powerdomain of first-countable spaces, and a complete characterization of the compact subsets of the Sorgenfrey line $\mathbb{R}_\ell$. □
Concluding remarks
Conclusion, past, and future

- A short tour of **semantics** of probabilistic programming languages
- This is just the tip of the iceberg…
  in fact I have barely touched the subject of **topology**
  — and only through domain theory
- Let me quickly tell you some of the other things…
Full abstraction

- Probabilistic PCF + (angelic) **non-determinism** + statistical termination testers \([V > r]\)
  is **fully abstract** \([JGL 15]\)

- Denotationally, redefine \([T\tau]\) as \(P_A[T\tau]\), domain of **Hoare previsions** on \([\tau]\) \([JGL 07]\)

- The key to full abstraction is theorems on **coincidence of topologies** \([JGL 15]\)

\[
M \preceq_n N \iff [M]_S \leq [N]_S, \text{ at all types.}
\]

**Definition**
A continuous *upper prevision* \(F\) is a functional from \((X \to \mathbb{R}^+)\)
to \(\mathbb{R}^+\) such that:

- \(F\) is (Soc\(^{\text{Hoare}}\))-continuous: \(F(\sup_{h \in H} h) = \sup_{h \in H} F(h)\);
- \(F\) is positively homogeneous: \(F(ah) = aF(h) (a \in \mathbb{R}^+)\);
- \(F\) is convex: \(F(h + h') \leq F(h) + F(h')\).

This models **probabilistic + angelic non-determinism**
(Exercise: Check that sup is indeed upper.)

**Lemma**
For every type \(\tau\), \([\tau]_S\) is a bc-domain.

(One of the nice CCCs of continuous domains.)

**Proposition (Key result — coincidence of topologies)**
If \(X\) and \(Y\) are bc-domains, then:

- **Scott topology on** \([X \to Y]\) = pointwise convergence
  Subbasis: \([a \in V]\) = \(\{f \mid f(a) \in V\}\), \(a \in X\), \(V\) open in \(Y\)
- **Scott topology on previsions on** \(X\) = weak topology
  Subbasis: \([h > r]\) = \(\{F \mid F(h) > r\}\), \(h \in [X \to I]\), \(r \in \mathbb{Q}\)
Previsions

- For a continuous valuation $\nu, h \mapsto \int_X h \, d\nu$ is continuous and linear.
  This is in fact an isomorphism $\nabla X \cong \left[ [X \to \overline{\mathbb{R}}_+] \to \text{lin} \overline{\mathbb{R}}_+ \right]$.

- Previsions relax the linearity requirement.
  E.g., Hoare previsions are sublinear: $F(h_1 + h_2) \leq F(h_1) + F(h_2)$ (and $F(r \cdot h) = r \cdot F(h)$).

- Isomorphisms between spaces of previsions on $X$ and convex hyperspaces over $\nabla X$,
  (as used by [Mislove 00], [McIver, Morgan 01], [Tix 99; Tix, Keimel, Plotkin 09])
  [JGL 08]: on continuous dcpos
  [JGL 17]: on large families of topological spaces
A few open questions

💫 Full abstraction for ISPCF+angelic non-determinism+statistical testers? [Yes]
   — any connection between statistical testers and conditioning?

💫 Fubini-Tonelli on $\mathbb{Dcpo}$?
   — equivalently, is every continuous valuation central on $\mathbb{Dcpo}$?

💫 Can the Jung-Tix dream be realized?
   — a Cartesian-closed category of continuous dcpos, closed under $\mathbf{V}$?

💫 Can we trust pseudo-random number generators
   instead of true random number generators
   in the implementation of $\text{sample}[0,1]$?