Models of Randomness

Part II: random types-as-closures

Fritz Obermeyer

Department of Mathematics
Carnegie-Mellon University

2008:04:15

Notation and concepts

SK = combinatory algebra
SKR = random extension ($R = \lambda x, y.x + y$)
  (note $(x + y) + z \neq x + (y + z)$; really $\frac{x+y}{2}$)
SKJ = parallel/ambiguous extension ($\mathbf{J} = \lambda x, y.x \mid y$)
  (some authors write join $x \mid y$ instead $x \sqcup y$)
SKJR = random+ambiguous extension

All four assume HP-complete observational equivalence:
  $M \sqsubseteq M'$ iff $\forall \underline{N}.\ M\ \underline{N}$ conv $\implies M'\ \underline{N}$ conv
where $\underline{N}$ ranges over traces $N_1, \ldots, N_k$ of arguments.

Definable types-as-closures builds on 2007 talk,
`http://www.math.cmu.edu/~fho/notes/`

Goal: a probability monad

In a curry type system in applied λ-calculi

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\text{Rand} \;:\; \text{type} \;\longrightarrow\; \text{type}$$

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\begin{aligned} \text{Rand} &: \text{type} \to \text{type} \\ \text{always} &: \forall\alpha.\ \alpha \to \text{Rand}\ \alpha \end{aligned}$$

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$
\begin{array}{lcl}
\mathsf{Rand} & : & \mathsf{type} \to \mathsf{type} \\
\mathsf{always} & : & \forall\alpha.\ \alpha \to \mathsf{Rand}\ \alpha \\
\mathsf{sample} & : & \forall\alpha,\beta.\ \mathsf{Rand}\ \alpha \to (\alpha \to \mathsf{Rand}\ \beta) \to \mathsf{Rand}\ \beta
\end{array}
$$

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\begin{array}{lcl} \text{Rand} & : & \text{type} \to \text{type} \\ \text{always} & : & \forall\alpha.\ \alpha \to \text{Rand}\ \alpha \\ \text{sample} & : & \forall\alpha,\beta.\ \text{Rand}\ \alpha \to (\alpha \to \text{Rand}\ \beta) \to \text{Rand}\ \beta \end{array}$$

In which applied λ-calculus should we look?

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\begin{aligned}
\text{Rand} &: \text{type} \to \text{type} \\
\text{always} &: \forall \alpha.\ \alpha \to \text{Rand}\ \alpha \\
\text{sample} &: \forall \alpha, \beta.\ \text{Rand}\ \alpha \to (\alpha \to \text{Rand}\ \beta) \to \text{Rand}\ \beta
\end{aligned}$$

In which applied λ-calculus should we look?
extend SK to randomized SKR.

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$
\begin{array}{lcl}
\text{Rand} & : & \text{type} \to \text{type} \\
\text{always} & : & \forall\alpha.\ \alpha \to \text{Rand}\ \alpha \\
\text{sample} & : & \forall\alpha,\beta.\ \text{Rand}\ \alpha \to (\alpha \to \text{Rand}\ \beta) \to \text{Rand}\ \beta
\end{array}
$$

In which applied λ-calculus should we look?
extend SK to randomized SKR.
Is Rand compatible with types-as-closures?

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\begin{array}{lcl} \text{Rand} & : & \text{type} \to \text{type} \\ \text{always} & : & \forall\alpha.\ \alpha \to \text{Rand}\ \alpha \\ \text{sample} & : & \forall\alpha,\beta.\ \text{Rand}\ \alpha \to (\alpha \to \text{Rand}\ \beta) \to \text{Rand}\ \beta \end{array}$$

In which applied λ-calculus should we look?
extend SK to randomized SKR.
Is Rand compatible with types-as-closures?
extend SK to random lattice SKJR.

Goal: a probability monad

In a curry type system in applied λ-calculi
(where types are properties of untyped terms),
we seek an instance of Moggi's computational monad

$$\begin{aligned} \text{Rand} &: \text{type} \to \text{type} \\ \text{always} &: \forall \alpha.\ \alpha \to \text{Rand}\ \alpha \\ \text{sample} &: \forall \alpha, \beta.\ \text{Rand}\ \alpha \to (\alpha \to \text{Rand}\ \beta) \to \text{Rand}\ \beta \end{aligned}$$

In which applied λ-calculus should we look?
extend SK to randomized SKR.

Is Rand compatible with types-as-closures?
extend SK to random lattice SKJR.
look for "small" definable subspaces.

Abstract probability algebras

Start with a dcpo with $\perp$.

Abstract probability algebras

Start with a dcpo with $\perp$.

Consider initial "R-algebra" with binary mixing $x + y$

Abstract probability algebras

Start with a dcpo with $\bot$.
Consider initial "R-algebra" with binary mixing $x + y$ subject to monotonicity and

$x + x = x$	*idempotence*
$x + y = y + x$	*commutativity*
$(w + x) + (y + z) = (w + z) + (y + x)$	*associativity*

Abstract probability algebras

Start with a dcpo with $\bot$.
Consider initial "R-algebra" with binary mixing $x + y$ subject to monotonicity and

$x + x = x$	*idempotence*
$x + y = y + x$	*commutativity*
$(w + x) + (y + z) = (w + z) + (y + x)$	*associativity*

- equivalent to arbitrary real mixing (in dcpo)

Abstract probability algebras

Start with a dcpo with $\perp$.
Consider initial "R-algebra" with binary mixing $x + y$ subject to monotonicity and

$x + x = x$	*idempotence*
$x + y = y + x$	*commutativity*
$(w + x) + (y + z) = (w + z) + (y + x)$	*associativity*

- equivalent to arbitrary real mixing (in dcpo)
- equivalent to valuations

Abstract probability algebras

Start with a dcpo with $\perp$.
Consider initial "R-algebra" with binary mixing $x + y$ subject to monotonicity and

$x + x = x$	*idempotence*
$x + y = y + x$	*commutativity*
$(w + x) + (y + z) = (w + z) + (y + x)$	*associativity*

- equivalent to arbitrary real mixing (in dcpo)
- equivalent to valuations

Compare with initial join-semilattice ("J-algebra")

$$x \mid x = x$$
$$x \mid y = y \mid x$$
$$x \mid (y \mid x) = (x \mid y) \mid z$$

Extending SK to SKR

Start with untyped combinatory algebra SK.

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(f + g)x = (f\ x) + (g\ x)$$

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(f + g)x = (f\ x) + (g\ x)$$

SKR is randomized Turing-complete.

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(\mathsf{f} + \mathsf{g})\mathsf{x} = (\mathsf{f}\ \ \mathsf{x}) + (\mathsf{g}\ \ \mathsf{x})$$

SKR is randomized Turing-complete.

Consider the curry type system

$$\frac{\mathsf{x} : \alpha \vdash \mathsf{M} : \beta}{\lambda \mathsf{x}.\mathsf{M} : \alpha \to \beta} \qquad \frac{\mathsf{M} : \alpha \to \beta \qquad \mathsf{N} : \alpha}{\mathsf{M}\ \ \mathsf{N} : \beta}$$

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(\mathsf{f} + \mathsf{g})\mathsf{x} = (\mathsf{f}\ \mathsf{x}) + (\mathsf{g}\ \mathsf{x})$$

SKR is randomized Turing-complete.

Consider the curry type system

$$\frac{\mathsf{x} : \alpha \vdash \mathsf{M} : \beta}{\lambda \mathsf{x}.\mathsf{M} : \alpha \to \beta} \qquad \frac{\mathsf{M} : \alpha \to \beta \qquad \mathsf{N} : \alpha}{\mathsf{M}\ \mathsf{N} : \beta} \qquad \frac{\mathsf{M}, \mathsf{N} : \mathsf{Rand}\ \alpha}{\mathsf{M} + \mathsf{N} : \mathsf{Rand}\ \alpha}$$

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(\mathsf{f} + \mathsf{g})\mathsf{x} = (\mathsf{f}\ \ \mathsf{x}) + (\mathsf{g}\ \ \mathsf{x})$$

SKR is randomized Turing-complete.

Consider the curry type system

$$\frac{\mathsf{x}:\alpha \vdash \mathsf{M}:\beta}{\lambda \mathsf{x}.\mathsf{M}:\alpha \to \beta} \qquad \frac{\mathsf{M}:\alpha \to \beta \qquad \mathsf{N}:\alpha}{\mathsf{M}\ \ \mathsf{N}:\beta} \qquad \frac{\mathsf{M},\mathsf{N}:\mathsf{Rand}\ \ \alpha}{\mathsf{M}+\mathsf{N}:\mathsf{Rand}\ \ \alpha}$$

and embed SK into SKR via always $= \lambda \mathsf{x}.\mathsf{x} : \forall \alpha.\alpha \to \mathsf{Rand}\ \ \alpha$.

Extending SK to SKR

Start with untyped combinatory algebra SK.
Extend to R-algebra with right-distributivity

$$(\mathsf{f} + \mathsf{g})\mathsf{x} = (\mathsf{f}\ \ \mathsf{x}) + (\mathsf{g}\ \ \mathsf{x})$$

SKR is randomized Turing-complete.

Consider the curry type system

$$\frac{\mathsf{x}:\alpha \vdash \mathsf{M}:\beta}{\lambda \mathsf{x}.\mathsf{M}:\alpha \to \beta} \qquad \frac{\mathsf{M}:\alpha \to \beta \qquad \mathsf{N}:\alpha}{\mathsf{M}\ \ \mathsf{N}:\beta} \qquad \frac{\mathsf{M},\mathsf{N}:\mathsf{Rand}\ \ \alpha}{\mathsf{M}+\mathsf{N}:\mathsf{Rand}\ \ \alpha}$$

and embed SK into SKR via always $= \lambda \mathsf{x}.\mathsf{x} : \forall \alpha.\alpha \to \mathsf{Rand}\ \ \alpha$.
How to sample, raising $\alpha \to \mathsf{Rand}\ \ \beta$ to $\mathsf{Rand}\ \ \alpha \to \mathsf{Rand}\ \ \beta$?

Sampling booleans

Introduce $\mathbf{K} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{x}$, $\mathbf{F} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{y}$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{\mathsf{M} : \mathsf{bool} \qquad \mathsf{N}, \mathsf{N}' : \alpha}{\mathsf{M}\ \ \mathsf{N}\ \ \mathsf{N}' : \alpha}$$

Sampling booleans

Introduce $\mathbf{K} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{x}$, $\mathbf{F} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{y}$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{\mathsf{M} : \mathsf{bool} \qquad \mathsf{N}, \mathsf{N}' : \alpha}{\mathsf{M}\ \ \mathsf{N}\ \ \mathsf{N}' : \alpha}$$

We can sample with

$$\mathsf{sample}_{\mathsf{bool}} = \lambda \mathsf{p}, \mathsf{f}.\ \ \mathsf{p}\ \ (\mathsf{f}\ \ \mathbf{K})\ \ (\mathsf{f}\ \ \mathbf{F})$$

Sampling booleans

Introduce $\mathbf{K} = \lambda x, y.x$, $\mathbf{F} = \lambda x, y.y$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{M : \mathsf{bool} \qquad N, N' : \alpha}{M \;\; N \;\; N' : \alpha}$$

We can sample with

$$\mathsf{sample}_{\mathsf{bool}} = \lambda p, f.\;\; p \;\; (f \;\; \mathbf{K}) \;\; (f \;\; \mathbf{F})$$

Letting, e.g.,

$$\mathsf{amb} \;\; := \;\; \lambda x.\;\; x \;\; (x \;\; \mathbf{K} \;\; \mathbf{F}) \;\; (x \;\; \mathbf{F} \;\; \mathbf{K}) \qquad \textit{ambiguity}$$

$$R \;\; := \;\; \mathbf{K} + \mathbf{F}, \qquad \textit{even coin toss}$$

note the difference between

amb R

Sampling booleans

Introduce $\mathbf{K} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{x}$, $\mathbf{F} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{y}$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{\mathsf{M} : \mathsf{bool} \qquad \mathsf{N}, \mathsf{N}' : \alpha}{\mathsf{M}\ \ \mathsf{N}\ \ \mathsf{N}' : \alpha}$$

We can sample with

$$\mathsf{sample}_{\mathsf{bool}} = \lambda \mathsf{p}, \mathsf{f}.\ \ \mathsf{p}\ \ (\mathsf{f}\ \ \mathbf{K})\ \ (\mathsf{f}\ \ \mathbf{F})$$

Letting, e.g.,

$\mathsf{amb} \ := \ \lambda \mathsf{x}.\ \mathsf{x}\ (\mathsf{x}\ \mathbf{K}\ \mathbf{F})\ (\mathsf{x}\ \mathbf{F}\ \mathbf{K})$ *ambiguity*

$\mathsf{R} \ := \ \mathbf{K} + \mathbf{F}$, *even coin toss*

note the difference between

$\mathsf{amb}\ \mathsf{R} \ = \ \mathbf{K} + \mathbf{F}$ *random*

Sampling booleans

Introduce $\mathbf{K} = \lambda x, y.x$, $\mathbf{F} = \lambda x, y.y$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{M : \mathsf{bool} \qquad N, N' : \alpha}{M\ N\ N' : \alpha}$$

We can sample with

$$\mathsf{sample}_{\mathsf{bool}} = \lambda p, f.\ p\ (f\ \mathbf{K})\ (f\ \mathbf{F})$$

Letting, e.g.,

$$\begin{aligned} \mathsf{amb} &:= \lambda x.\ x\ (x\ \mathbf{K}\ \mathbf{F})\ (x\ \mathbf{F}\ \mathbf{K}) \\ R &:= \mathbf{K} + \mathbf{F}, \end{aligned}$$

ambiguity
even coin toss

note the difference between

$$\mathsf{amb}\ R = \mathbf{K} + \mathbf{F}$$

random

$$\mathsf{sample}_{\mathsf{bool}}\ x \text{ from } R \text{ in } \mathsf{amb}\ x$$

...sugar for

Sampling booleans

Introduce $\mathbf{K} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{x}$, $\mathbf{F} = \lambda \mathsf{x}, \mathsf{y}.\mathsf{y}$ with typing

$$\frac{}{\mathbf{K} : \mathsf{bool}} \qquad \frac{}{\mathbf{F} : \mathsf{bool}} \qquad \frac{\mathsf{M} : \mathsf{bool} \qquad \mathsf{N}, \mathsf{N}' : \alpha}{\mathsf{M}\ \ \mathsf{N}\ \ \mathsf{N}' : \alpha}$$

We can sample with

$$\mathsf{sample}_{\mathsf{bool}} = \lambda \mathsf{p}, \mathsf{f}.\ \ \mathsf{p}\ \ (\mathsf{f}\ \ \mathbf{K})\ \ (\mathsf{f}\ \ \mathbf{F})$$

Letting, e.g.,

$$\mathsf{amb} \ := \ \lambda \mathsf{x}.\ \mathsf{x}\ (\mathsf{x}\ \mathbf{K}\ \mathbf{F})\ (\mathsf{x}\ \mathbf{F}\ \mathbf{K}) \qquad \textit{ambiguity}$$
$$\mathsf{R} \ := \ \mathbf{K} + \mathbf{F}, \qquad \textit{even coin toss}$$

note the difference between

$$\mathsf{amb}\ \mathsf{R} \ = \ \mathbf{K} + \mathbf{F} \qquad \textit{random}$$
$$\mathsf{sample}_{\mathsf{bool}}\ \mathsf{x}\ \mathsf{from}\ \mathsf{R}\ \mathsf{in}\ \mathsf{amb}\ \mathsf{x} \qquad \textit{...sugar for}$$
$$= \ \mathsf{sample}_{\mathsf{bool}}\ \mathsf{R}\ \lambda \mathsf{x}.\ \mathsf{amb}\ \mathsf{x}$$

Sampling booleans

Introduce $\mathbf{K} = \lambda x, y.x$, $\mathbf{F} = \lambda x, y.y$ with typing

$$\frac{}{\mathbf{K} : \text{bool}} \qquad \frac{}{\mathbf{F} : \text{bool}} \qquad \frac{M : \text{bool} \qquad N, N' : \alpha}{M\ N\ N' : \alpha}$$

We can sample with

$$\text{sample}_{\text{bool}} = \lambda p, f.\ \ p\ \ (f\ \ \mathbf{K})\ \ (f\ \ \mathbf{F})$$

Letting, e.g.,

$$\text{amb} \ := \ \lambda x.\ x\ (x\ \mathbf{K}\ \mathbf{F})\ (x\ \mathbf{F}\ \mathbf{K})$$ *ambiguity*

$$R \ := \ \mathbf{K} + \mathbf{F},$$ *even coin toss*

note the difference between

$$\text{amb } R \ = \ \mathbf{K} + \mathbf{F}$$ *random*

$$\text{sample}_{\text{bool}}\ x \text{ from } R \text{ in amb } x$$ *...sugar for*

$$= \ \text{sample}_{\text{bool}}\ R\ \lambda x.\ \text{amb}\ x$$

$$= \ \mathbf{K}$$ *deterministic*

Sampling natural numbers

Let $\mathsf{zero} = \lambda\_, \mathsf{x.x}$, $\mathsf{succ} = \lambda \mathsf{n}, \mathsf{f}, \mathsf{x}.\ \mathsf{f}(\mathsf{n}\ \ \mathsf{f}\ \ \mathsf{x})$ with typing

$$\frac{}{\mathsf{zero} : \mathsf{nat}} \qquad \frac{}{\mathsf{succ} : \mathsf{nat} \rightarrow \mathsf{nat}} \qquad \frac{\mathsf{n} : \mathsf{nat} \qquad \mathsf{s} : \alpha \rightarrow \alpha \qquad \mathsf{z} : \alpha}{\mathsf{n}\ \ \mathsf{s}\ \ \mathsf{z} : \alpha}$$

Sampling natural numbers

Let $\mathsf{zero} = \lambda\_, \mathsf{x}.\mathsf{x}$, $\mathsf{succ} = \lambda \mathsf{n}, \mathsf{f}, \mathsf{x}.\ \mathsf{f}(\mathsf{n}\ \ \mathsf{f}\ \ \mathsf{x})$ with typing

$$\frac{}{\mathsf{zero} : \mathsf{nat}} \qquad \frac{}{\mathsf{succ} : \mathsf{nat} \to \mathsf{nat}} \qquad \frac{\mathsf{n} : \mathsf{nat} \qquad \mathsf{s} : \alpha \to \alpha \qquad \mathsf{z} : \alpha}{\mathsf{n}\ \ \mathsf{s}\ \ \mathsf{z} : \alpha}$$

and sample with

$$\mathsf{sample}_{\mathsf{nat}} = \lambda \mathsf{p}, \mathsf{f}.\ \ \mathsf{p}\ \ (\mathsf{f} \circ \mathsf{succ})\ \ (\mathsf{f}\ \ \mathsf{zero})$$

Sampling natural numbers

Let $\text{zero} = \lambda\_, x.x$, $\text{succ} = \lambda n, f, x.\ f(n\ f\ x)$ with typing

$$\frac{}{\text{zero} : \text{nat}} \qquad \frac{}{\text{succ} : \text{nat} \to \text{nat}} \qquad \frac{n : \text{nat} \qquad s : \alpha \to \alpha \qquad z : \alpha}{n\ s\ z : \alpha}$$

and sample with

$$\text{sample}_{\text{nat}} = \lambda p, f.\ p\ \ (f \circ \text{succ})\ \ (f\ \ \text{zero})$$

then lift $\text{succ} : \text{nat} \to \text{nat}$ to $\text{succ}' : \text{Rand}\ \ \text{nat} \to \text{Rand}\ \ \text{nat}$

$$\text{succ}' = \lambda p.\ \text{sample}_{\text{nat}}\ \ p\ \ \text{succ}$$

Sampling natural numbers

Let zero $= \lambda\_, x.x$, succ $= \lambda n, f, x.\ f(n\ f\ x)$ with typing

$$\frac{}{\mathsf{zero} : \mathsf{nat}} \qquad \frac{}{\mathsf{succ} : \mathsf{nat} \to \mathsf{nat}} \qquad \frac{n : \mathsf{nat} \qquad s : \alpha \to \alpha \qquad z : \alpha}{n\ s\ z : \alpha}$$

and sample with

$$\mathsf{sample}_{\mathsf{nat}} = \lambda p, f.\ p\ (f \circ \mathsf{succ})\ (f\ \mathsf{zero})$$

then lift $\mathsf{succ} : \mathsf{nat} \to \mathsf{nat}$ to $\mathsf{succ}' : \mathsf{Rand\ nat} \to \mathsf{Rand\ nat}$

$$\mathsf{succ}' = \lambda p.\ \mathsf{sample}_{\mathsf{nat}}\ p\ \mathsf{succ}$$

Moral Church terms are already monadic

Sampling partial terms

General method for defining $\mathsf{sample}_\alpha\ \mathsf{p}\ \mathsf{f}$:

Sampling partial terms

General method for defining sample_α p f:

- enumerate domain α

Sampling partial terms

General method for defining sample_{α} p f:

- enumerate domain α
- apply f to each x:α

Sampling partial terms

General method for defining sample_α p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Sampling partial terms

General method for defining sample$_\alpha$ p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Method doesn't yet work for partial terms,

Sampling partial terms

General method for defining sample$_\alpha$ p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Method doesn't yet work for partial terms,
e.g. infinite domains, functions.

Sampling partial terms

General method for defining sample$_\alpha$ p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Method doesn't yet work for partial terms,
e.g. infinite domains, functions.

Method works if directed joins are definable,

Sampling partial terms

General method for defining sample$_\alpha$ p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Method doesn't yet work for partial terms,
e.g. infinite domains, functions.

Method works if directed joins are definable,
e.g. in lattice models with randomness...

Sampling partial terms

General method for defining sample$_\alpha$ p f:

- enumerate domain α
- apply f to each x:α
- case-branch on x based on p

Method doesn't yet work for partial terms,
e.g. infinite domains, functions.

Method works if directed joins are definable,
e.g. in lattice models with randomness...
...but first look at typing in lattice models.

Types-as-closures

Curry-types are properties of untyped terms.

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha$$ *extensive, idempotent*

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property $\mathsf{x} : \alpha$ iff $\mathsf{x} = \alpha\ \mathsf{x}$.

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property $\mathsf{x} : \alpha$ iff $\mathsf{x} = \alpha\ \mathsf{x}$.
Typechecking becomes an equational problem.

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join “SKJ”,

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property x : α iff x $= \alpha$ x.
Typechecking becomes an equational problem.
Typing can be enforced at the term level:
e.g. (bool x) will be “boolean” for any x.

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property x : α iff x $= \alpha$ x.
Typechecking becomes an equational problem.
Typing can be enforced at the term level:
e.g. (bool x) will be "boolean" for any x.
But we get extra "garbage" inhabitants,

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property x : α iff x $= \alpha$ x.
Typechecking becomes an equational problem.
Typing can be enforced at the term level:
e.g. (bool x) will be "boolean" for any x.
But we get extra "garbage" inhabitants, e.g.

$$\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Types-as-closures

Curry-types are properties of untyped terms.
Consider closures in combinatory algebra with join "SKJ",

$$\mathbf{I} \sqsubseteq \alpha = \alpha \circ \alpha \qquad \textit{extensive, idempotent}$$

and the fixedpoint property $x : \alpha$ iff $x = \alpha\ x$.
Typechecking becomes an equational problem.
Typing can be enforced at the term level:
e.g. (bool x) will be "boolean" for any x.
But we get extra "garbage" inhabitants, e.g.

$$\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

To minimize garbage, look for closures with small ranges.

Defining closures

How can we define closures?

Defining closures

How can we define closures?

General idea: join represents ambiguity.

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
unit, bool, nat,

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
unit, bool, nat, Prod, Sum, Exp,

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
unit, bool, nat, Prod, Sum, Exp, recursive,

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
unit, bool, nat, Prod, Sum, Exp, recursive,
polymorphic, dependent,

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
  (join over all ways of looking at x).
Only booleans remain unblurred,
  all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
  unit, bool, nat, Prod, Sum, Exp, recursive,
  polymorphic, dependent, subtypes, type-of-types

Defining closures

How can we define closures?

General idea: join represents ambiguity.
When expecting a bool x, squint
(join over all ways of looking at x).
Only booleans remain unblurred,
all else blurs to $\top$=complete ambiguity.

Specifically: argue about action on Bohm trees.

What types are definable?
unit, bool, nat, Prod, Sum, Exp, recursive,
polymorphic, dependent, subtypes, type-of-types

(see 2007 talk on definable closures for details)

A lattice with randomness

Seeking a random monad in a lattice model, consider the free JR-algebra over SK, where application right-distributes over both

$$(\mathrm{f} \mid \mathrm{g})\mathrm{x} = \mathrm{f}\ \ \mathrm{x} \ \ \mid \ \ \mathrm{g}\ \ \mathrm{x}$$
$$(\mathrm{f} + \mathrm{g})\mathrm{x} = \mathrm{f}\ \ \mathrm{x} \ \ + \ \ \mathrm{g}\ \ \mathrm{x}$$

Call R-terms mixtures,

A lattice with randomness

Seeking a random monad in a lattice model, consider the free JR-algebra over SK, where application right-distributes over both

$$(f \mid g)x = f\ \ x\ \ \mid\ \ g\ \ x$$
$$(f + g)x = f\ \ x\ \ +\ \ g\ \ x$$

Call R-terms mixtures, J-terms joins,

A lattice with randomness

Seeking a random monad in a lattice model, consider the free JR-algebra over SK, where application right-distributes over both

$$(f \mid g)x = f\ x \ \mid \ g\ x$$
$$(f + g)x = f\ x \ + \ g\ x$$

Call R-terms mixtures, J-terms joins, RJ-terms slurries, e.g. $w \mid (x + (y \mid z))$

A lattice with randomness

Seeking a random monad in a lattice model, consider the free JR-algebra over SK, where application right-distributes over both

$$(f \mid g)x = f\ x \ \mid\ g\ x$$
$$(f + g)x = f\ x\ +\ g\ x$$

Call R-terms mixtures, J-terms joins, RJ-terms slurries, e.g. $w \mid (x + (y \mid z))$
Application right-distributes over slurries,

A lattice with randomness

Seeking a random monad in a lattice model, consider the free JR-algebra over SK, where application right-distributes over both

$$(f \mid g)x = f\ \ x\ \ \mid\ \ g\ \ x$$
$$(f + g)x = f\ \ x\ \ +\ \ g\ \ x$$

Call R-terms mixtures, J-terms joins, RJ-terms slurries, e.g. $w \mid (x + (y \mid z))$

Application right-distributes over slurries, but RJ distributivity fails,

A lattice with randomness

Seeking a random monad in a lattice model,
consider the free JR-algebra over SK, where
application right-distributes over both

$$(f \mid g)x = f\ x \ \mid \ g\ x$$
$$(f + g)x = f\ x \ + \ g\ x$$

Call R-terms mixtures, J-terms joins,
RJ-terms slurries, e.g. $w \mid (x + (y \mid z))$
Application right-distributes over slurries,
but RJ distributivity fails,
so slurries can be infinitely deep.

JR-Bohm-trees

JR-Bohm trees are defined by

$$\frac{x \ \ \text{var} \qquad M_1, \ldots, M_k \ \ \text{BT}}{x \ M_1 \ \ \ldots \ \ M_k \ \ \text{BT}} \qquad \frac{x \ \ \text{var} \qquad M \ \ \text{BT}}{\lambda x.M \ \ \text{BT}}$$

JR-Bohm-trees

JR-Bohm trees are defined by

$$\frac{\mathsf{x}\ \ \mathsf{var} \qquad \mathsf{M}_1, \ldots, \mathsf{M}_k\ \ \mathsf{BT}}{\mathsf{x}\ \ \mathsf{M}_1\ \ \ldots\ \ \mathsf{M}_k\ \ \mathsf{BT}} \qquad \frac{\mathsf{x}\ \ \mathsf{var} \qquad \mathsf{M}\ \ \mathsf{BT}}{\lambda \mathsf{x}.\mathsf{M}\ \ \mathsf{BT}}$$

$$\frac{\underline{\mathsf{M}}\ \ \mathsf{Set}(\mathsf{BT})}{\bigsqcup \underline{\mathsf{M}}\ \ \mathsf{BT}} \qquad \frac{\underline{\mathsf{M}}\ \ \mathsf{Prob}(\mathsf{BT})}{\int \underline{\mathsf{M}}\ \ \mathsf{BT}}$$

JR-Bohm-trees

JR-Bohm trees are defined by

$$\frac{x \ \ \mathrm{var} \qquad M_1, \ldots, M_k \ \ \mathrm{BT}}{x \ M_1 \ \ \ldots \ \ M_k \ \ \mathrm{BT}} \qquad \frac{x \ \ \mathrm{var} \qquad M \ \ \mathrm{BT}}{\lambda x.M \ \ \mathrm{BT}}$$

$$\frac{\underline{M} \ \ \mathrm{Set}(\mathrm{BT})}{\bigsqcup \underline{M} \ \ \mathrm{BT}} \qquad \frac{\underline{M} \ \ \mathrm{Prob}(\mathrm{BT})}{\int \underline{M} \ \ \mathrm{BT}}$$

Theorem

Every SKJR-term is equivalent to a slurry of BTs.

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(x + y) \mid z = (x \mid z) + (y \mid z)$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(x + y) \mid z = (x \mid z) + (y \mid z)$$

but distributivity fails in SKJR, e.g.

$$(\mathbf{I} \mid \bot + \top) \ (\bot + \top)$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x} + \mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned} &(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\ &\quad = \ \bot + \top \mid \bot + \top \end{aligned}$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x} + \mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned}(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\ = \ \bot + \top \mid \bot + \top\ = \bot + \top\end{aligned}$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x} + \mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned}(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\ = \ \bot + \top \mid \bot + \top \ = \bot + \top \\ ((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top)\end{aligned}$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x} + \mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned}
&(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\
&\quad = \ \bot + \top \mid \bot + \top \ = \bot + \top \\
&((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top) \\
&\quad = \ \mathbf{I}(\bot + \top) + \top(\bot + \top)
\end{aligned}$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x}+\mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned}
&(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\
&\qquad = \ \bot + \top \mid \bot + \top \ = \bot + \top \\
&((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top) \\
&\qquad = \ \mathbf{I}(\bot + \top) + \top(\bot + \top) \ = (\bot + \top) + \top
\end{aligned}$$

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x}+\mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned}
&(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\
&\qquad = \ \bot + \top \mid \bot + \top \ = \bot + \top \\
&((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top) \\
&\qquad = \ \mathbf{I}(\bot + \top) + \top(\bot + \top) \ = (\bot + \top) + \top
\end{aligned}$$

We can still sample, but never in parallel, inside joins.

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x}+\mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$\begin{aligned} &(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\ &\quad = \ \bot + \top \mid \bot + \top \ = \bot + \top \\ &((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top) \\ &\quad = \ \mathbf{I}(\bot + \top) + \top(\bot + \top) \ = (\bot + \top) + \top \end{aligned}$$

We can still sample, but never in parallel, inside joins.
So we'll require R-normal forms for "nice" inhabitants.

Ambiguity vs. parallelism

Historical operational semantics for join is parallelism.
Operational semantics for randomness is sampling.
Combining these requires distributivity

$$(\mathsf{x} + \mathsf{y}) \mid \mathsf{z} = (\mathsf{x} \mid \mathsf{z}) + (\mathsf{y} \mid \mathsf{z})$$

but distributivity fails in SKJR, e.g.

$$
\begin{aligned}
&(\mathbf{I} \mid \bot + \top)\ (\bot + \top) \\
&\qquad = \ \bot + \top \mid \bot + \top \ \ = \bot + \top \\
&((\mathbf{I} \mid \bot) + (\mathbf{I} \mid \top))\ (\bot + \top) \\
&\qquad = \ \mathbf{I}(\bot + \top) + \top(\bot + \top) \ \ = (\bot + \top) + \top
\end{aligned}
$$

We can still sample, but never in parallel, inside joins.
So we'll require R-normal forms for "nice" inhabitants.
For example...

Random convergence/divergence

The smallest nontrivial closure is

$$\mathsf{div} \;:=\; \lambda \mathsf{x}.\; \mathsf{x} \mid \mathsf{x}\ \top \mid \mathsf{x}\ \top\ \top \mid \mathsf{x}\ \top\ \top\ \top \mid \ldots$$

Random convergence/divergence

The smallest nontrivial closure is

$$\mathsf{div} \;:=\; \lambda\mathsf{x}.\ \mathsf{x} \mid \mathsf{x}\ \top \mid \mathsf{x}\ \top\ \top \mid \mathsf{x}\ \top\ \top\ \top \mid \ldots$$

Theorem

In SKJ, $\mathrm{inhab}(\mathsf{div}) = \{\bot, \top\}$.

Random convergence/divergence

The smallest nontrivial closure is

$$\mathsf{div} \;:=\; \lambda \mathsf{x}.\ \mathsf{x} \mid \mathsf{x}\ \top \mid \mathsf{x}\ \top\ \top \mid \mathsf{x}\ \top\ \top\ \top \mid \ldots$$

Theorem

In SKJ, $\mathrm{inhab}(\mathsf{div}) = \{\bot, \top\}$.

Since $\{\bot, \top\}$ is the two-point lattice,
we get R-normal forms for div,

Random convergence/divergence

The smallest nontrivial closure is

$$\mathsf{div} \;:=\; \lambda \mathsf{x}.\ \mathsf{x} \mid \mathsf{x}\ \top \mid \mathsf{x}\ \top\ \top \mid \mathsf{x}\ \top\ \top\ \top \mid \ldots$$

Theorem

In SKJ, $\mathrm{inhab}(\mathsf{div}) = \{\bot, \top\}$.

Since $\{\bot, \top\}$ is the two-point lattice,
we get R-normal forms for div, and thus

Theorem

In SKJR, $\mathrm{inhab}(\mathsf{div}) = \{\bot@\mathsf{t} + \top@(1-\mathsf{t}) \mid \mathsf{t} \in [0,1]\}$.

Random convergence/divergence

The smallest nontrivial closure is

$$\text{div} \;:=\; \lambda x.\; x \mid x\ \top \mid x\ \top\ \top \mid x\ \top\ \top\ \top \mid \ldots$$

Theorem

In SKJ, $\mathrm{inhab}(\text{div}) = \{\bot, \top\}$.

Since $\{\bot, \top\}$ is the two-point lattice,
we get R-normal forms for div, and thus

Theorem

In SKJR, $\mathrm{inhab}(\text{div}) = \{\bot @ t + \top @ (1-t) \;\mid\; t \;\in\; [0,1]\}$.

Convergence in SKJR is probabilistic;

Random convergence/divergence

The smallest nontrivial closure is

$$\text{div} := \lambda x.\ x \mid x\ \top \mid x\ \top\ \top \mid x\ \top\ \top\ \top \mid \ldots$$

Theorem

In SKJ, $\text{inhab}(\text{div}) = \{\bot, \top\}$.

Since $\{\bot, \top\}$ is the two-point lattice,
we get R-normal forms for div, and thus

Theorem

In SKJR, $\text{inhab}(\text{div}) = \{\bot @ t + \top @ (1-t) \mid t \in [0,1]\}$.

Convergence in SKJR is probabilistic;
observational information ordering is defined by
$M \sqsubseteq M'$ iff $\forall$trace $\underline{N}$. $\text{conv}(M\ \underline{N}) \leq \text{conv}(M'\ \underline{N})$.

The range property, spectra

(recall we want small ranges, few inhabitants)

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $|\ \text{rng}(\top)\ | = 1$,

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $|\,\mathrm{rng}(\top)\,| = 1$, $|\,\mathrm{rng}(\mathrm{div})\,| = 2$

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $\mid \text{rng}(\top) \mid = 1$, $\mid \text{rng}(\text{div}) \mid = 2$
$\mid \text{rng}(\text{semi}) \mid = 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $|$ rng$(\top)$ $|= 1$, $|$ rng(div) $|= 2$
$|$ rng(semi) $|= 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,
but some are finite dimensional,

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $\mid \text{rng}(\top) \mid = 1$, $\mid \text{rng}(\text{div}) \mid = 2$
$\mid \text{rng}(\text{semi}) \mid = 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,
but some are finite dimensional,
e.g., $\dim(\text{rng}(\top)) = 0$,

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $|\text{rng}(\top)| = 1$, $|\text{rng}(\text{div})| = 2$
$|\text{rng}(\text{semi})| = 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,
but some are finite dimensional,
e.g., $\dim(\text{rng}(\top)) = 0$, $\dim(\text{rng}(\text{div})) = 1$.

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $\mid \text{rng}(\top) \mid = 1$, $\mid \text{rng}(\text{div}) \mid = 2$
$\mid \text{rng}(\text{semi}) \mid = 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,
but some are finite dimensional,
e.g., $\dim(\text{rng}(\top)) = 0$, $\dim(\text{rng}(\text{div})) = 1$.

Question What is the spectrum of dimensions?

The range property, spectra

(recall we want small ranges, few inhabitants)
In SK, all nontrivial ranges are infinite.

In SKJ, some nontrivial ranges are finite,
e.g., $|\operatorname{rng}(\top)| = 1$, $|\operatorname{rng}(\text{div})| = 2$
$|\operatorname{rng}(\text{semi})| = 3, \ldots$ ΛPAUSE In

SKJR, all nontrivial ranges are infinite,
but some are finite dimensional,
e.g., $\dim(\operatorname{rng}(\top)) = 0$, $\dim(\operatorname{rng}(\text{div})) = 1$.

Question What is the spectrum of dimensions?

Known At least $\{0, 1, 2^{\aleph_0}\}$.

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\mathrm{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\mathrm{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\mathrm{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\mathrm{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\mathrm{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate $\lambda \mathsf{x}.\mathsf{x}(\mathsf{x}\ \mathbf{K}\ \top)(\mathsf{x}\ \top\ \mathbf{F}) : \mathsf{boool} \to \mathsf{bool}$,

$$\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\text{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\text{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate $\lambda \mathsf{x}.\mathsf{x}(\mathsf{x}\ \mathbf{K}\ \top)(\mathsf{x}\ \top\ \mathbf{F}) : \mathsf{boool} \to \mathsf{bool}$,

$$\text{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\text{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\text{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate $\lambda \mathsf{x}.\mathsf{x}(\mathsf{x}\ \mathbf{K}\ \top)(\mathsf{x}\ \top\ \mathbf{F}) : \mathsf{boool} \to \mathsf{bool}$,

$$\text{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

The Slurry_α, sample_α, always_α monad exists for many types: bool, nat, Prod, Sum, Exp, recursive. (not obvious)

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\mathrm{inhab}(\textsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\mathrm{inhab}(\textsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate λx.x(x $\mathbf{K}$ $\top$)(x $\top$ $\mathbf{F}$) : boool $\rightarrow$ bool,

$$\mathrm{inhab}(\textsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

The Slurry_α, sample_α, always_α monad exists for many types: bool, nat, Prod, Sum, Exp, recursive. (not obvious)

But slurries aren't "nice":

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\text{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\text{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate $\lambda x.x(x\ \mathbf{K}\ \top)(x\ \top\ \mathbf{F}) : \mathsf{boool} \to \mathsf{bool}$,

$$\text{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

The Slurry_α, sample_α, always_α monad exists for many types:
bool, nat, Prod, Sum, Exp, recursive. (not obvious)

But slurries aren't "nice":
no R-normal forms,

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\text{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\text{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate $\lambda \mathsf{x}.\mathsf{x}(\mathsf{x}\ \mathbf{K}\ \top)(\mathsf{x}\ \top\ \mathbf{F}) : \mathsf{boool} \to \mathsf{bool}$,

$$\text{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

The Slurry_α, sample_α, always_α monad exists for many types:
bool, nat, Prod, Sum, Exp, recursive. (not obvious)

But slurries aren't "nice":
no R-normal forms, no sampling semantics.

Are there less trivial types?

In SKJ we can define types boool and semi with

$$\mathrm{inhab}(\text{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$$
$$\mathrm{inhab}(\text{semi}) = \{\bot, \mathbf{I}, \top\}$$

and disambiguate λx.x(x $\mathbf{K}$ $\top$)(x $\top$ $\mathbf{F}$) : boool $\to$ bool,

$$\mathrm{inhab}(\text{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$$

Similarly in SKJR we can define semi, bool with inhabitants slurries of $\{\bot, \mathbf{I}, \top\}$ and $\{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

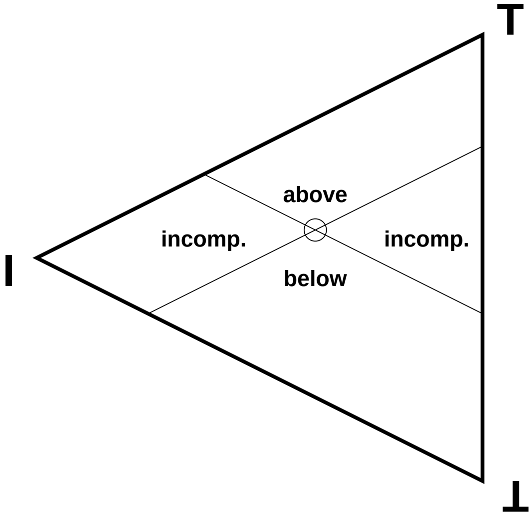
The Slurry$_\alpha$, sample$_\alpha$, always$_\alpha$ monad exists for many types: bool, nat, Prod, Sum, Exp, recursive. (not obvious)

But slurries aren't "nice": no R-normal forms, no sampling semantics.

Can we disambiguate to finite dimensional mixtures, without determinizing? (unknown)

Mixtures of semibooleans

The space of $\bot, \mathbf{I}, \top$-mixtures is 2-dimensional:



Slurries of semibooleans

Theorem

$\bot, \mathbf{I}, \top$-slurries are completely characterized by their action on the unit interval $[\bot, \top]$.

Slurries of semibooleans

Theorem

$\perp, \mathbf{I}, \top$*-slurries are completely characterized by their action on the unit interval* $[\perp, \top]$.

We thus have JR-isomorphism with the monotone convex functions $: [0, 1] \to [0, 1]$.

Slurries of semibooleans

Theorem

$\bot, \mathbf{I}, \top$*-slurries are completely characterized by their action on the unit interval* $[\bot, \top]$*.*

We thus have JR-isomorphism with the monotone convex functions : $[0, 1] \to [0, 1]$.

This space has dimension $2^{\aleph_0}$.

Slurries of semibooleans

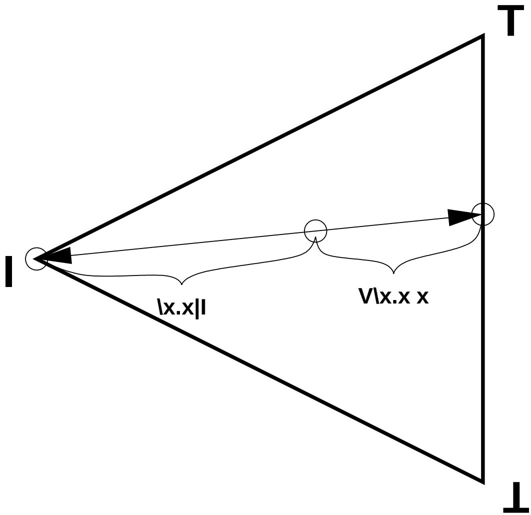
Theorem

$\bot, \mathbf{I}, \top$*-slurries are completely characterized by their action on the unit interval* $[\bot, \top]$*.*

We thus have JR-isomorphism with the monotone convex functions : $[0, 1] \to [0, 1]$.
This space has dimension $2^{\aleph_0}$.
Can we raise to a finite-dimensional subspace?

A 1-dimensional subspace

Try raising with Vλx.x x and then Vλx.x | $\mathbf{I}$:



A 1-dimensional subspace

Try raising with Vλx.x x and then Vλx.x $\mid \mathbf{I}$:

Range is 1-dimensional,

A 1-dimensional subspace

Try raising with Vλx.x x and then Vλx.x | $\mathbf{I}$:

Range is 1-dimensional,
almost has R-n.f. (binary join of mixtures)

A 1-dimensional subspace

Try raising with Vλx.x x and then Vλx.x $\mid$ $\mathbf{I}$:

Range is 1-dimensional,
almost has R-n.f. (binary join of mixtures)
but arguably useless...

A 1-dimensional subspace

Try raising with Vλx.x x and then Vλx.x $|$ $\mathbf{I}$:

Range is 1-dimensional,
almost has R-n.f. (binary join of mixtures)
but arguably useless...

The closure Vλx.x x is more useful
but has infinite dimensional range.

Mixtures of booleans

In SKJ, $\mathrm{inhab}(\mathsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;

Mixtures of booleans

In SKJ, $\mathrm{inhab}(\mathsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

Mixtures of booleans

In SKJ, $\text{inhab}(\textsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\text{inhab}(\textsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

In SKJR, disambiguation interferes with randomness,

Mixtures of booleans

In SKJ, $\mathrm{inhab}(\mathsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.
In SKJR, disambiguation interferes with randomness,
and we're stuck with $\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures.

Mixtures of booleans

In SKJ, $\text{inhab}(\textsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\text{inhab}(\textsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.
In SKJR, disambiguation interferes with randomness,
and we're stuck with $\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures.

$\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures are 5-dimensional.

Mixtures of booleans

In SKJ, $\text{inhab}(\text{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\text{inhab}(\text{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.
In SKJR, disambiguation interferes with randomness,
and we're stuck with $\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures.

$\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures are 5-dimensional.
Cumulative distribution functions yield a model:

Mixtures of booleans

In SKJ, $\mathrm{inhab}(\mathsf{boool}) = \{\bot,\ \mathbf{K},\ \mathbf{F},\ \mathbf{J} = \mathbf{K} \mid \mathbf{F},\ \top\}$;
a "disambiguation" trick then raises $\mathbf{J}$ to $\top$
yielding $\mathrm{inhab}(\mathsf{bool}) = \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.
In SKJR, disambiguation interferes with randomness,
and we're stuck with $\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures.

$\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top$-mixtures are 5-dimensional.
Cumulative distribution functions yield a model:

$$\{\ (\mathsf{b}, \mathsf{k}, \mathsf{f}, \mathsf{j}, \mathsf{t}) \in [0,1]^5 \\ \mid\ \mathsf{k} \geq \mathsf{b},\ \ \mathsf{f} \geq \mathsf{b},\ \ \mathsf{j} + \mathsf{b} \geq \mathsf{k} + \mathsf{f},\ \ \mathsf{t} \geq \mathsf{j}\ \}$$

partially ordered componentwise.

Slurries of booleans

Theorem

$\perp, \mathbf{K}, \mathbf{F}, \top$-*slurries are completely characterized by their action on two arguments in unit interval* $[\perp, \top]$.

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$-slurries are completely characterized by their action on two arguments in unit interval $[\bot, \top]$.

We thus have JR-isomorphism with the monotone convex functions : $[0, 1]^2 \to [0, 1]$.

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$-slurries are completely characterized by their action on two arguments in unit interval $[\bot, \top]$.

We thus have JR-isomorphism with the
monotone convex functions : $[0, 1]^2 \to [0, 1]$.
Can we raise to a finite-dimensional subspace?

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$-slurries are completely characterized by their action on two arguments in unit interval $[\bot, \top]$.

We thus have JR-isomorphism with the
monotone convex functions : $[0, 1]^2 \to [0, 1]$.
Can we raise to a finite-dimensional subspace?
Want to raise to a linear upper bound.

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$-slurries are completely characterized by their action on two arguments in unit interval $[\bot, \top]$.

We thus have JR-isomorphism with the
monotone convex functions $: [0,1]^2 \rightarrow [0,1]$.
Can we raise to a finite-dimensional subspace?
Want to raise to a linear upper bound.

We can raise $\bot, \mathbf{K}, \mathbf{F}, \top$-slurries with λx.x x x.

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$*-slurries are completely characterized by their action on two arguments in unit interval* $[\bot, \top]$*.*

We thus have JR-isomorphism with the
monotone convex functions $: [0,1]^2 \rightarrow [0,1]$.
Can we raise to a finite-dimensional subspace?
Want to raise to a linear upper bound.

We can raise $\bot, \mathbf{K}, \mathbf{F}, \top$-slurries with λx.x x x.
and partially disambiguate with λx.x(x $\mathbf{K}$ $\top$)(x $\top$ $\mathbf{F}$).

Slurries of booleans

Theorem

$\bot, \mathbf{K}, \mathbf{F}, \top$*-slurries are completely characterized by their action on two arguments in unit interval* $[\bot, \top]$*.*

We thus have JR-isomorphism with the
monotone convex functions: $[0, 1]^2 \rightarrow [0, 1]$.
Can we raise to a finite-dimensional subspace?
Want to raise to a linear upper bound.

We can raise $\bot, \mathbf{K}, \mathbf{F}, \top$-slurries with λx.x x x.
and partially disambiguate with λx.x(x $\mathbf{K}$ $\top$)(x $\top$ $\mathbf{F}$).
???

Problems with SKJR

Problem Disambiguation fails:

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,
but not enforced at the term level.

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,
but not enforced at the term level.

Problem Lower-bounding fails:

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,
but not enforced at the term level.

Problem Lower-bounding fails:
without distributivity, we can't raise e.g.
$\bot + \top$ to $\mathbf{I} + \top$.

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,
but not enforced at the term level.

Problem Lower-bounding fails:
without distributivity, we can't raise e.g.
$\bot + \top$ to $\mathbf{I} + \top$.

Adding parallelism allows lower-bounding,

Problems with SKJR

Problem Disambiguation fails:
even if we can raise slurries to mixtures,
random samples may be ambiguous.

Unambiguity can be checked equationally,
but not enforced at the term level.

Problem Lower-bounding fails:
without distributivity, we can't raise e.g.
$\bot + \top$ to $\mathbf{I} + \top$.

Adding parallelism allows lower-bounding,
but then requires sequentialization...