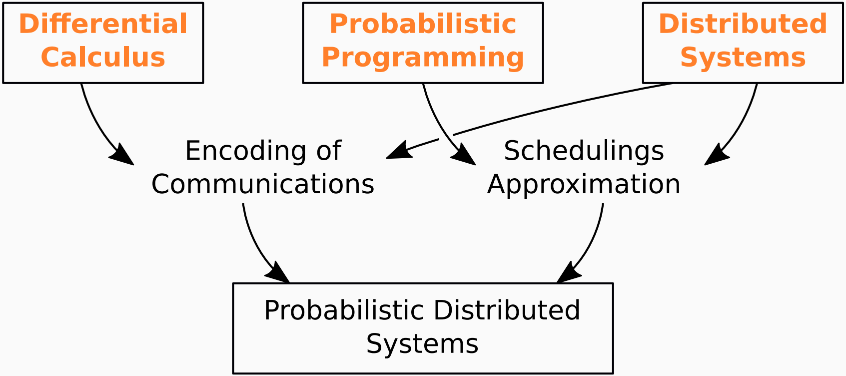
Sémantiques des Calculs Distribués, Différentiels et Probabilistes

Habilitation à diriger des recherches

Christine Tasson

23 novembre 2018


Differential Calculus
Probabilistic Programming
Distributed Systems
Encoding of Communications
Schedulings Approximation
Probabilistic Distributed Systems

Computer Science **Mathematics**

Programs → Interpretation (Denotational Semantics)

Computer Science $\circlearrowright$ Mathematics

```
import random:
def flip(p):
  if random.random()<p:
    return 0
  else:
    return 1
```

$$\begin{array}{ccc} [0,1] & \xrightarrow{f} & \mathcal{V}([0,1]) \\ 0.3 & \mapsto & 0.3\,\delta_0 + 0.7\,\delta_1 \end{array}$$

Programs → (Denotational Semantics) → Interpretation

Computer Science ↻ Mathematics

```
import random:
def flip(p):
  if random.random()<p:
    return 0
  else:
    return 1
```

$$[0,1] \xrightarrow{f} \mathcal{V}([0,1])$$
$$0.3 \mapsto 0.3\,\delta_0 + 0.7\,\delta_1$$

Denotational Semantics

Programs → Interpretation

Constructions ← Structures

Computational content

Computer Science ⟳ Mathematics

```
import random:
def flip(p):
  if random.random()<p:
    return 0
  else:
    return 1
```

⟶

$$[0,1] \xrightarrow{f} \mathcal{V}([0,1])$$
$$0.3 \mapsto 0.3\,\delta_0 + 0.7\,\delta_1$$

Programs $\longrightarrow$ Interpretation (Denotational Semantics)

Constructions $\longleftarrow$ Structures (Computational content)

Lambda $\longleftarrow$ Functions

```
def shift(n):
  return lambda s:s+n
```

Lambda-Calculus

Church

1930

1930: Church

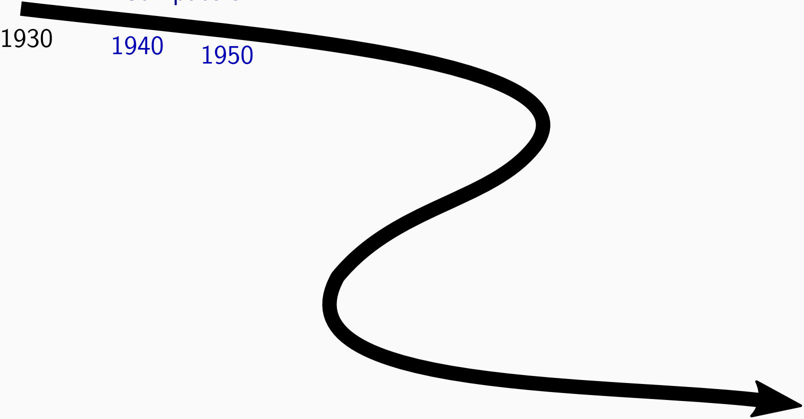
Lambda-terms represent computable functions.

	Programs	Functions	
	M, N	$f, g : \mathbb{N} \to \mathbb{N}$	
Variable	x	x	Variable
Abstraction	$\lambda x.M$	$f : x \mapsto f(x)$	Map
Application	$(\lambda x.M)N$	$f \circ g : x \mapsto f(\, g(x)\,)$	Composition

Lambda-Calculus

Church

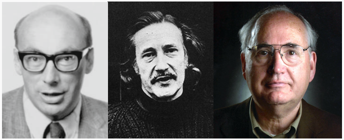
Computers



Church

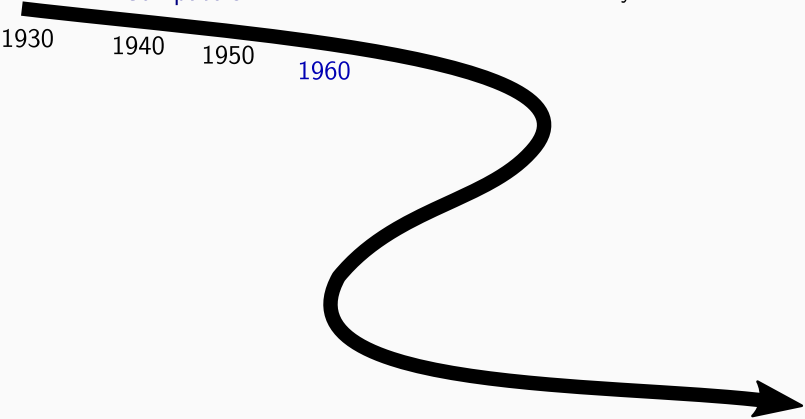
Lambda-Calculus

Operational and Denotational Semantics



Landin Strachey Scott

Computers



1960: From syntax to semantics

Syntax describes how to write programs,
Semantics describes how and what programs compute.

Operational semantics describes program execution as transition system. [Landin 1966]

For λ-calculus, **substitution** in contexts

$$(\lambda x.M)N \to M[N/x]$$

Denotational Semantics denotes programs as functions acting on *values* and on *memory state*. [Strachey 1960] [Scott 1969]
For pure λ-calculus, solving **equation**

$$D \stackrel{?}{=} Var + [D \to D] + \cdots$$

1960: From syntax to semantics

Syntax describes how to write programs,
Semantics describes how and what programs compute.

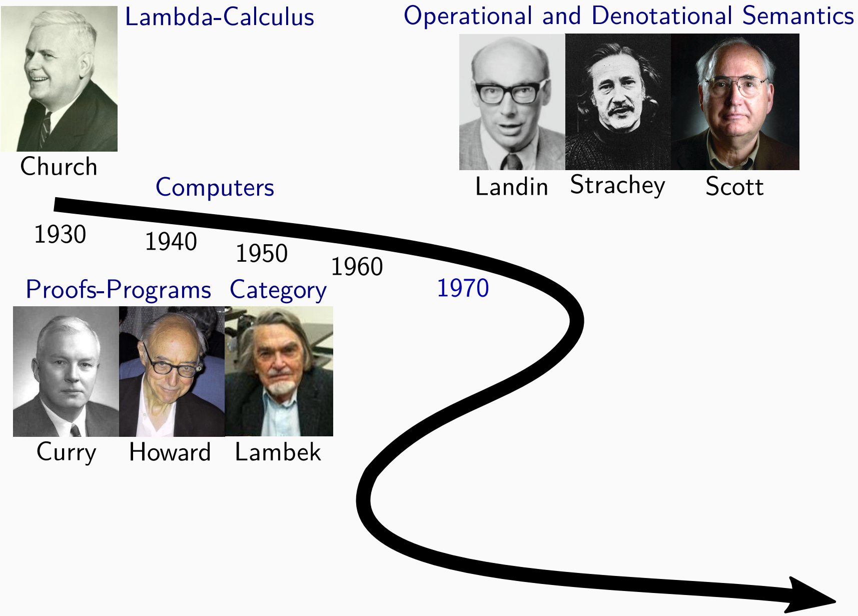
Operational semantics describes program execution as transition system. [Landin 1966]

For λ-calculus, **substitution** in contexts

$$(\lambda x.M)N \to M[N/x]$$

Denotational Semantics denotes programs as functions acting on *values* and on *memory state*. [Strachey 1960] [Scott 1969]
For pure λ-calculus, solving **equation**

$$D \stackrel{\checkmark}{=} Var + \underbrace{[D \to D]}_{\text{Continuous}} + \cdots$$


Lambda-Calculus
Church
Computers
Operational and Denotational Semantics
Landin
Strachey
Scott
1930
1940
1950
1960
1970
Proofs-Programs
Category
Curry
Howard
Lambek

1970: Computer Science - Logic - Category

Curry-Howard correspondence between *programs* and *proofs*

λ-**calculus**	**Logic**
Term : Type	Proof : Formula
$M : A \Rightarrow B$	$\dfrac{\pi}{A \Rightarrow B}$

1970: Computer Science - Logic - Category

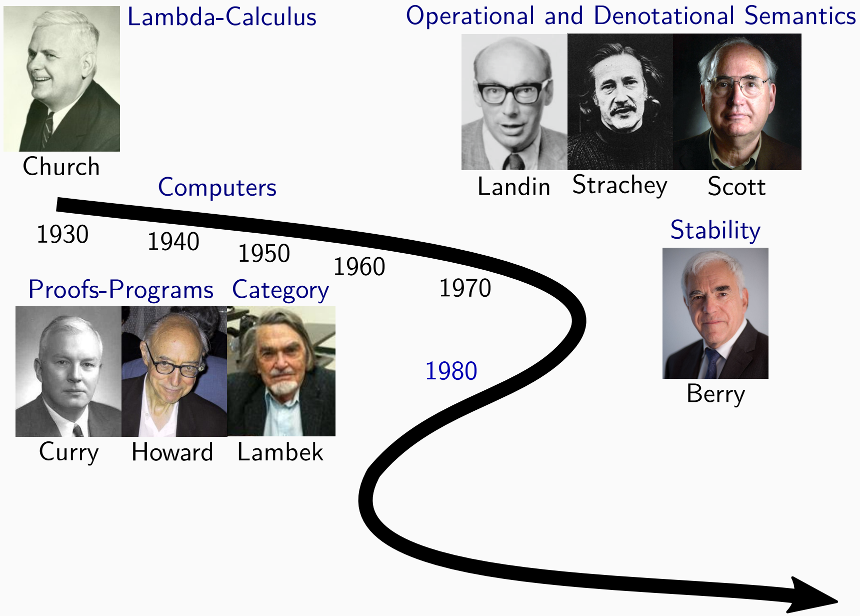
Curry-Howard correspondence between *programs* and *proofs*

λ-calculus	**Logic**
Term : Type	Proof : Formula
$M : A \Rightarrow B$	$\dfrac{\pi}{A \Rightarrow B}$

Lambek correspondence with **Cartesian Closed Categories**

Categories are made of objects and morphisms with $\circ$ *composition*,

$[A \to B]$ Object of Morphisms from A to B



Lambda-Calculus
Church
Operational and Denotational Semantics
Landin
Strachey
Scott
Computers
1930
1940
1950
1960
1970
1980
Stability
Berry
Proofs-Programs
Category
Curry
Howard
Lambek

1980: Sequential algorithms

PCF a typed functional languages such as `Haskell` or `ML`

$$M, N, P := \underbrace{x \mid \lambda x.M \mid (M)\, N}_{\lambda\text{-calculus}} \mid \underbrace{0 \mid \texttt{succ}\, M}_{\text{Integers}} \mid \underbrace{\texttt{if}\, M\, \texttt{then}\, N\, \texttt{else}\, P}_{\text{Conditional}} \mid \underbrace{\texttt{fix}\, M}_{\text{Recursion}}$$

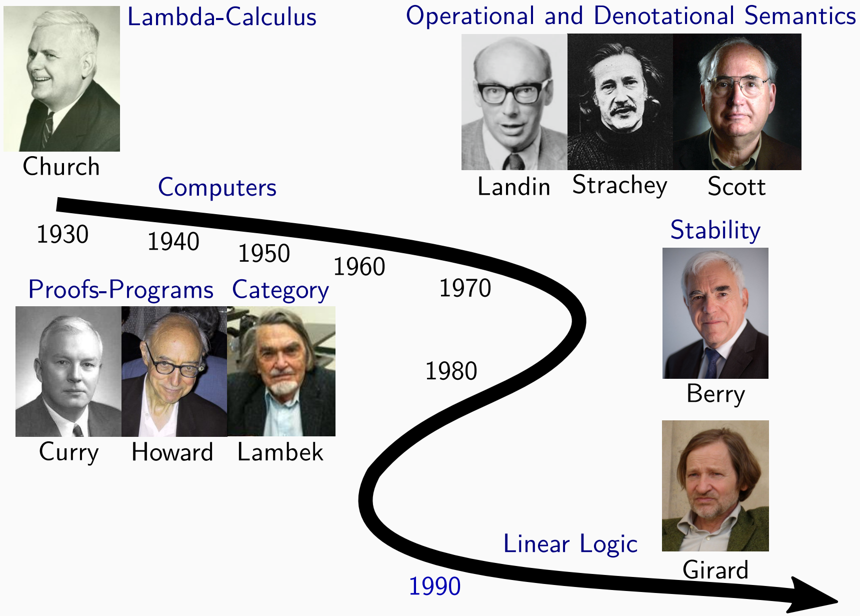
Denotational Semantics

Scott Domains contain non sequential functions such as Parallel-Or.

Stability gets rid of this example, but does not characterize *sequentiality*

Sequential algorithm model uses the language of category [Berry-Curien 1982]

The Full Abstraction quest generates new models *Hypercoherence* [Ehrhard 1993] and *Game semantics* [Abramsky-Jagadeesan-Malacaria 1994], [Hyland-Ong 1995]


Lambda-Calculus
Church
Operational and Denotational Semantics
Landin
Strachey
Scott
Computers
1930
1940
1950
1960
1970
1980
1990
Proofs-Programs
Category
Curry
Howard
Lambek
Stability
Berry
Linear Logic
Girard

1990: Linear Logic

Semantical observation: [Girard 1987]

$$A \overset{\textit{Stable}}{\Rightarrow} B \quad \simeq \quad !A \overset{\textit{Linear}}{\multimap} B$$

Girard introduced new models

- qualitative Coherent Spaces [Girard 1986]
- quantitative Normal Functors [Girard 1988] and Probabilistic Coherent Spaces [Girard 2004]

Categorical models

Linear		**Non-Linear**
$(\mathcal{L}, 1, \otimes, \multimap)$ SMCC	$\perp$	$(\mathcal{M}, \top, \times, \Rightarrow)$ CCC

Table of contents

Differential λ-Calculus

Differential Lambda Calculus

Semantical observation: in quantitative models of Linear Logic, programs are interpreted by smooth functions, hence differentiation.
[Ehrhard-Regnier 2003]

	Programs	Functions	
	M, N	f, g	
Variable	x	x	Variable
Abstraction	$\lambda x.M$	$f : x \mapsto f(x)$	Map
Application	$(\lambda x.M)N$	$f \circ g : x \mapsto f(\,g(x)\,)$	Composition
Differentiation	$D\lambda x.M \cdot N$	$u, x \mapsto Df_x(u)$	Derivation

Categorical Model of Differential Lambda-Calculus

Definition 4.2 A *Cartesian (closed) differential category* is a Cartesian (closed) left-additive category having an operator $D(-)$ that maps a morphism $f : A \to B$ into a morphism $D(f) : A \times A \to B$ and satisfies the following axioms:

D1. $D(f+g) = D(f) + D(g)$ and $D(0) = 0$

D2. $D(f) \circ \langle h+k, v\rangle = D(f) \circ \langle h, v\rangle + D(f) \circ \langle k, v\rangle$ and $D(f) \circ \langle 0, v\rangle = 0$

D3. $D(\mathrm{Id}) = \pi_1$, $D(\pi_1) = \pi_1 \circ \pi_1$ and $D(\pi_2) = \pi_2 \circ \pi_1$

D4. $D(\langle f, g\rangle) = \langle D(f), D(g)\rangle$

D5. $D(f \circ g) = D(f) \circ \langle D(g), g \circ \pi_2\rangle$

D6. $D(D(f)) \circ \langle\langle g, 0\rangle, \langle h, k\rangle\rangle = D(f) \circ \langle g, k\rangle$

D7. $D(D(f)) \circ \langle\langle 0, h\rangle, \langle g, k\rangle\rangle = D(D(f)) \circ \langle\langle 0, g\rangle, \langle h, k\rangle\rangle$

[Blute-Cockett-Seely 2009] [Bucciarelli-Ehrhard-Manzonetto 2010]

A differential operator such that if $f : A \Rightarrow B$, then $Df : A \times A \Rightarrow B$ corresponds to $u, x \mapsto Df_x(u)$ with axioms for linearity in 1st coord.

Categorical Model of Differential Lambda-Calculus

Definition 4.2 A *Cartesian (closed) differential category* is a Cartesian (closed) left-additive category having an operator $D(-)$ that maps a morphism $f : A \to B$ into a morphism $D(f) : A \times A \to B$ and satisfies the following axioms:

D1. $D(f+g) = D(f) + D(g)$ and $D(0) = 0$

D2. $D(f) \circ \langle h+k, v\rangle = D(f) \circ \langle h, v\rangle + D(f) \circ \langle k, v\rangle$ and $D(f) \circ \langle 0, v\rangle = 0$

D3. $D(\mathrm{Id}) = \pi_1$, $D(\pi_1) = \pi_1 \circ \pi_1$ and $D(\pi_2) = \pi_2 \circ \pi_1$

D4. $D(\langle f, g\rangle) = \langle D(f), D(g)\rangle$

D5. $D(f \circ g) = D(f) \circ \langle D(g), g \circ \pi_2\rangle$

D6. $D(D(f)) \circ \langle\langle g, 0\rangle, \langle h, k\rangle\rangle = D(f) \circ \langle g, k\rangle$

D7. $D(D(f)) \circ \langle\langle 0, h\rangle, \langle g, k\rangle\rangle = D(D(f)) \circ \langle\langle 0, g\rangle, \langle h, k\rangle\rangle$

[Blute-Cockett-Seely 2009] [Bucciarelli-Ehrhard-Manzonetto 2010]

A differential operator such that if $f : A \Rightarrow B$, then $Df : A \times A \Rightarrow B$ corresponds to $u, x \mapsto Df_x(u)$ with axioms for linearity in 1st coord.

What setting for handling both linear and non-linear variables ?

using the substitution monoidal structure [Fiore-Plotkin-Turi 1999].

Linear Substitution

A profunctor $A \overset{F}{\nrightarrow} B$ is a functor $A \times B^{\mathrm{op}} \to \mathbf{Set}$,
it generalizes relations and matrices but with set coefficients.

Composition: $G \circ F(a, c) = \int^{b \in B} G(b, c) \times F(a, b)$

A generalised species is a profunctor $\mathcal{R} : \mathcal{L}A \nrightarrow A$ where
$\mathcal{L}$ computes the free *Symmetric Monoidal Category* over a category A.
$\mathcal{L}A$: sequences $\langle a_1, \ldots, a_n \rangle$ and bijections and sequence of morphisms.
[Fiore-Gambino-Hyland-Winskel 2007]

As for operads, substitution of generalised species is described by the composition in the Kleisli bicategory: $\mathcal{L}A \overset{\mathcal{R}}{\nrightarrow} A \quad \mathcal{L}A \overset{\mathcal{R}}{\nrightarrow} A$ gives a profuntor $\mathcal{L}A \overset{\mathcal{R} \circ \mathcal{R}}{\nrightarrow} A$ because $\mathcal{L}$ lifts to profunctors
[Fiore-Gambino-Hyland-Winskel 2016]

Resource Lambda Calculus

Semantical observation: in quantitative models of Linear Logic, programs are interpreted by series, hence Syntactic Taylor Expansion approximating programs by polynomials. [Ehrhard-Regnier 2006]

	Programs	Functions
	s, t	f, g
Variable	x	x
Abstraction	$\lambda x.s$	$f : x \mapsto \sum a_n x^n$
Linear App.	$\langle \lambda x.s \rangle [t_1, \ldots, t_n]$	$f \circ g : x \mapsto \sum a_n \underbrace{g(x) \cdot \cdots \cdot g(x)}_{n}$

Resource terms formalized as a generalised species $\mathcal{R} : \mathcal{L}A \nrightarrow A$
$\mathcal{R}(\langle a_1, \ldots, a_\ell \rangle, b)$ is the set of resource terms $x_1 : a_1, \ldots, x_\ell : a_\ell \vdash s : b$
[Ong-Tsukada 2017]

Non-Linear Substitution

A Cartesian generalised species a profunctor $\Lambda : \mathcal{M}A \nrightarrow A$ where $\mathcal{M}$ computes the free *Cartesian Category* over a category A.
$\mathcal{M}A$: sequences $\langle \overline{a}_1, \ldots, \overline{a}_n \rangle$ and functions and sequence of morphisms.
[Tanaka-Power 2004]

As for Lawvere theory, substitution is described by the composition in the Kleisli bicategory which is possible because $\mathcal{M}$ also lifts to profunctors.

Lambda terms can be formalized as a cartesian generalized species.
$\Lambda(\langle \overline{b}_1, \ldots, \overline{b}_n \rangle, \overline{b})$: the set of lambda terms $x_1 : \overline{b}_1, \ldots, x_n : \overline{b}_n \vdash M : \overline{b}$
[Hyland 2017]

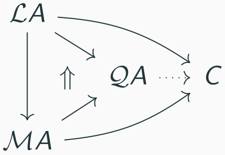
Mathematical Theory of Linear / non-Linear Substitution

What construction to combine into a 2-monad lifting to profunctors ?

- $\mathcal{L}$ free symmetric monoidal category 2-monad
  $\mathcal{L}A$: objects are sequences $\langle a_1, \ldots, a_\ell \rangle$
  morphisms are bijections and sequence of morphisms.
- $\mathcal{M}$ free cartesian cateogory 2-monad
  $\mathcal{M}A$: objects are sequences $\langle \overline{b}_1, \ldots, \overline{b}_n \rangle$
  morphisms are functions and sequence of morphisms.
- $\mathcal{Q}$ Mixed linear / non linear 2-monad [Power-Tanaka 2005][Fiore 2006]
  $\mathcal{Q}A$: objects are mixed sequences $\langle a_1, \ldots, a_\ell, \overline{b}_1, \ldots, \overline{b}_n \rangle$
  morphisms combine functions, bijections and sequence of morphisms.

Mixed Linear Non Linear Monad

Colimit in the 2-category of Symmetric Monoidal Categories.



$$\begin{array}{ccc} \mathcal{Q}A & \to & \mathcal{M}A \\ \langle a_1, \ldots, a_\ell, \overline{b}_1, \ldots, \overline{b}_n \rangle & \mapsto & \langle \overline{a}_1, \ldots, \overline{a}_\ell, \overline{b}_1, \ldots, \overline{b}_n \rangle \end{array}$$

Theorem (Hyland - Tasson)

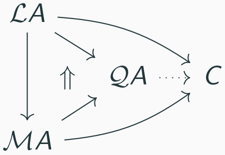
$\mathcal{Q}$ is a 2-monad on Symmetric Monoidal Categories.

Theorem (Hyland - Tasson)

A $\mathcal{Q}$-algebra is a Symmetric Monoidal Category that splits through a Cartesian Category with coherences.

Mixed Linear Non Linear Monad

Colimit in the 2-category of Symmetric Monoidal Categories.



$$\begin{array}{ccc} \mathcal{Q}A & \to & \mathcal{M}A \\ \langle a_1, \dots, a_\ell, \overline{b}_1, \dots, \overline{b}_n \rangle & \mapsto & \langle \overline{a}_1, \dots, \overline{a}_\ell, \overline{b}_1, \dots, \overline{b}_n \rangle \end{array}$$

We do not know if $\mathcal{Q}$ lifts to profunctors.

Theorem (Hyland - Tasson)

$\mathcal{Q}$ is a 2-monad on Symmetric Monoidal Categories.

Theorem (Hyland - Tasson)

A $\mathcal{Q}$-algebra is a Symmetric Monoidal Category that splits through a Cartesian Category with coherences.

Contribution

- The construction of the colimit of 2-monads
- The characterisation of its algebras

Next steps

- Lift $\mathcal{Q}$ to profunctors and describe the substitution monoidal structure of mixed linear/non linear variables.
- Combine the additive structure and encode differential operator

Perspectives

- Study other 2-monads appearing in semantics
- Exploit the bridge with combinatorics

Table of contents

Probabilistic Programming

Study the **implementation** of probabilistic algorithms with *formal methods*: correctness, termination, behavior in context,. . .

Operational Semantics describes **how** probabilistic programs compute.

Denotational Semantics describes **what** probabilistic programs compute

Study the **implementation** of probabilistic algorithms with *formal methods*: correctness, termination, behavior in context,...

Operational Semantics describes **how** probabilistic programs compute.

Prob(M, N) is the probability that M reduces to N

- In the discrete setting, **Prob**(M, N) is a stochastic *matrix*
- In the continuous setting, **Prob**(M, N) is a stochastic *kernel*

Denotational Semantics describes **what** probabilistic programs compute

Study the **implementation** of probabilistic algorithms with *formal methods*: correctness, termination, behavior in context,...

Operational Semantics describes **how** probabilistic programs compute.

$\mathbf{Prob}(M, N)$ is the probability that M reduces to N

- In the discrete setting, $\mathbf{Prob}(M, N)$ is a stochastic *matrix*
- In the continuous setting, $\mathbf{Prob}(M, N)$ is a stochastic *kernel*

Denotational Semantics describes **what** probabilistic programs compute

$[\![M]\!]$ is a probabilistic distribution, if M is a closed ground type program

- If $\vdash M : \mathtt{nat}$, then $[\![M]\!]$ a *discrete* distributions over integers
- If $\vdash M : \mathtt{real}$, then $[\![M]\!]$ a *continuous* distributions over reals

Syntax

Nat PPCF

Types: $A, B ::= \texttt{nat} \mid A \to B$

Terms: $M, N, L ::=$

$x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid$

$\underline{n} \mid \mathsf{succ}(M) \mid$

$\texttt{ifz}(L, M, N) \mid$

$\mathsf{coin} \mid \texttt{let}\, x{=}M\, \texttt{in}\, N$

Operational Semantics:

$\mathbf{Prob}(\mathsf{coin}, \underline{0}) = \frac{1}{2}$

If $\vdash M : \texttt{nat}$, $\mathbf{Prob}^{\infty}(M, \_)$ is the discrete distribution over $\mathbb{N}$ computed by M.

Syntax

Nat PPCF

Types: $A, B ::= \texttt{nat} \mid A \to B$

Terms: $M, N, L ::=$

$x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid$

$\underline{n} \mid \mathsf{succ}(M) \mid$

$\texttt{ifz}(L, M, N) \mid$

$\mathsf{coin} \mid \texttt{let}\, x{=}M \,\texttt{in}\, N$

Operational Semantics:

$\mathbf{Prob}(\mathsf{coin}, \underline{0}) = \frac{1}{2}$

If $\vdash M : \texttt{nat}$, $\mathbf{Prob}^{\infty}(M, \_)$ is the discrete distribution over $\mathbb{N}$ computed by M.

Real PPCF

Types: $A, B ::= \texttt{real} \mid A \to B$

Terms: $M, N, L ::=$

$x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid$

$\underline{r} \mid \underline{f}(M_1, \ldots, M_n) \mid$

$\texttt{ifz}(L, M, N) \mid$

$\texttt{sample} \mid \texttt{let}\, x{=}M \,\texttt{in}\, N$

Operational Semantics:

$\mathbf{Prob}(\texttt{sample}, U) = \lambda_{[0,1]}(U)$

If $\vdash M : \texttt{real}$, $\mathbf{Prob}^{\infty}(M, \_)$ is the continuous distribution over $\mathbb{R}$ computed by M.

Denotational Semantics - Discrete

	Domains Semantics	Quantitative Semantics
Types	Continuous **dcpos** $(X, \leq)$	**Proba. Coh. Spaces** $(\lvert X\rvert, \mathrm{P}(X) \subseteq \mathbb{R}_{\geq 0}^{\lvert X\rvert})$
Programs	**Scott Continuous**	**Analytic** Functions
Probability	Probabilistic **monad** $\mathcal{V}$	**Values** as proba. distrib.

Type:

$\mathbb{N}_\perp$ flat domain,

$\mathcal{V}(\mathbb{N}_\perp)$ proba. distr. over $\mathbb{N}_\perp$,

Prog: $[\![M]\!] : \mathbb{N}_\perp \to \mathcal{V}(\mathbb{N}_\perp)$,

$[\![\texttt{let n=x in M}]\!] : \mathcal{V}(\mathbb{N}_\perp) \to \mathcal{V}(\mathbb{N}_\perp)$

$$x \mapsto \left(\sum_n [\![M]\!]_{n,q} x_n \right)_q$$

[Jones-Plotkin 1989]

Type:

$\lvert\mathbf{Nat}\rvert = \mathbb{N}$

$\mathrm{P}(\mathbf{Nat})$ subproba. dist. over $\mathbb{N}$

Prog: $[\![M]\!] : \mathrm{P}(\mathbf{Nat}) \to \mathrm{P}(\mathbf{Nat})$

$$x \mapsto \left(\sum_{\mu = [n_1, \ldots, n_k]} [\![M]\!]_{\mu,q} \prod_{i=1}^{k} x_{n_i} \right)_q$$

[Danos-Ehrhard 2008]

Denotational Semantics - Continuous

Memory : **measurable** space and probabilistic

Programs: **kernels** encoding transformations of memory. [Kozen 1981]

The category **Kern** is **cartesian** but **not closed**. [Panangaden 1999]

Denotational Semantics - Continuous

Memory : **measurable** space and probabilistic
Programs: **kernels** encoding transformations of memory. [Kozen 1981]
The category **Kern** is **cartesian** but **not closed**. [Panangaden 1999]

Quasi-Borel spaces, a model of Real PPCF and recursive types based on domains and presheaves [Vakar-Kammar-Staton 2019].

Denotational Semantics - Continuous

Memory : **measurable** space and probabilistic
Programs: **kernels** encoding transformations of memory. [Kozen 1981]
The category **Kern** is **cartesian** but **not closed**. [Panangaden 1999]

Quasi-Borel spaces, a model of Real PPCF and recursive types based on domains and presheaves [Vakar-Kammar-Staton 2019].

A CCC with measurability ! [Ehrhard-Pagani-Tasson 2018]

1. **Complete cones** and Scott continuous functions
   However, the category is cartesian but not closed.
2. Complete cones and **Stable functions** is cartesian closed.
   However, not every stable function is measurable.
3. **Measurable Cones** (complete cones with **measurable tests**). Measurable paths pass measurable tests and **Measurable Stable functions** preserve measurable paths.

Interpretation of programs

Discrete

- For $\vdash \underline{n} : \mathbb{N}$,
  $[\![\underline{n}]\!]_p = \delta_{p,n}$
- For $\vdash \text{coin} : \mathbb{N}$,
  $[\![\text{coin}]\!]_p = \frac{1}{2}\delta_{0,p} + \frac{1}{2}\delta_{1,p}$
- For $\vdash N : \mathbb{N}$, $\vdash P : A$, $\vdash Q : A$,
  $[\![\texttt{ifz}(N, P, Q)]\!]_a =$
  $[\![N]\!]_0 [\![P]\!]_a + \sum_{n \neq 0} [\![N]\!]_{n+1} [\![Q]\!]_a$

  $[\![\texttt{let}\, x = N \,\texttt{in}\, P]\!]_a =$
  $$\sum_{n=0}^{\infty} [\![N]\!]_n \widehat{[\![P]\!]}(n)_a$$

Interpretation of programs

Discrete

- For $\vdash \underline{n} : \mathbb{N}$,
  $[\![\underline{n}]\!]_p = \delta_{p,n}$
- For $\vdash \text{coin} : \mathbb{N}$,
  $[\![\text{coin}]\!]_p = \frac{1}{2}\delta_{0,p} + \frac{1}{2}\delta_{1,p}$
- For $\vdash N : \mathbb{N}$, $\vdash P : A$, $\vdash Q : A$,
  $[\![\texttt{ifz}(N, P, Q)]\!]_a =$
  $[\![N]\!]_0 [\![P]\!]_a + \sum_{n \neq 0} [\![N]\!]_{n+1} [\![Q]\!]_a$

  $[\![\texttt{let}\, x{=}N\, \texttt{in}\, P]\!]_a =$
  $$\sum_{n=0}^{\infty} [\![N]\!]_n \widehat{[\![P]\!]}(n)_a$$

Continuous

- For $\vdash \underline{r} : \texttt{real}$,
  $[\![\underline{r}]\!](U) = \delta_r(U)$
- For $\vdash \texttt{sample} : \texttt{real}$,
  $[\![\texttt{sample}]\!] = \lambda_{[0,1]}(U)$
- For $\vdash R : \texttt{real}$, $\vdash P, Q : A$,
  $[\![\texttt{ifz}(R, P, Q)]\!](U) =$
  $[\![R]\!](\{0\})[\![P]\!](U) + [\![R]\!](\mathbb{R} \setminus \{0\})[\![Q]\!](U)$

  $[\![\texttt{let}\, x{=}R\, \texttt{in}\, P]\!](U) =$
  $$\int [\![R]\!](dr) [\![P]\!](\delta_r)(U)$$

Results

Invariance of semantics

- (Discrete) $[\![M]\!] = \sum_N \mathbf{Prob}(M, N)[\![N]\!]$
- (Continuous) $[\![M]\!] = \int \mathbf{Prob}(M, dt)[\![t]\!]$

Results

Invariance of semantics

- (Discrete) $[\![M]\!] = \sum_N \mathbf{Prob}(M, N)[\![N]\!]$
- (Continuous) $[\![M]\!] = \int \mathbf{Prob}(M, dt)[\![t]\!]$

Adequacy Lemma

- (Discrete) If $\vdash M : \texttt{nat}$, then $[\![M]\!]_n = \mathbf{Prob}^{\infty}(M, \underline{n})$
- (Continous) If $\vdash M : \texttt{real}$, then $[\![M]\!](U) = \mathbf{Prob}^{\infty}(M, \underline{U})$

Results

Invariance of semantics

- (Discrete) $[\![M]\!] = \sum_N \mathbf{Prob}(M, N)[\![N]\!]$
- (Continuous) $[\![M]\!] = \int \mathbf{Prob}(M, dt)[\![t]\!]$

Adequacy Lemma

- (Discrete) If $\vdash M : \texttt{nat}$, then $[\![M]\!]_n = \mathbf{Prob}^\infty(M, \underline{n})$
- (Continous) If $\vdash M : \texttt{real}$, then $[\![M]\!](U) = \mathbf{Prob}^\infty(M, \underline{U})$

Adequacy:

If $[\![P]\!] = [\![Q]\!]$ then $P \simeq Q$ ($\mathbf{Prob}^\infty(C[P], \cdot) = \mathbf{Prob}^\infty(C[Q], \cdot)$)

- (Discrete) **Pcoh** is adequate for Nat PPCF. [Danos-Ehrhard 2008]
- (Continuous):

Theorem (Ehrhard-Pagani-Tasson 2018)

Measurable cones and **Stable** measurable functions are adequate for Real PPCF.

Results

Full Abstraction: $[\![P]\!] = [\![Q]\!]$ iff $P \simeq Q$

- (Discrete ✓) **Pcoh** is adequate for Nat PPCF. [Danos-Ehrhard 2008]

Theorems (Ehrhard-Pagani-Tasson 2018)

Probabilistic Coherent Spaces are Fully Abstract for Nat $\mathrm{PPCF}$and for probabilistic Call-By-Push-Value.

Key tool: programs are interpreted as series thanks to quantitative semantics of LL

- (Continuous **?**) We do not know if Full Abstraction holds for Measurable cones and Stable measurable functions.
  The continuous case is a conservative extension of the discrete case [Crubille 2018]

From Theory to Application

Denotational semantics is a first step towards certification.

By applying **Operational Semantics, Invariance** of the denotational semantics, **Adequacy** we can prove properties of the implementation

- (Discrete) Rejection Sampling Algorithm
- (Continuous) Metropolis Hasting Algorithm

Rejection Sampling Algorithm

Input: A $\underline{0}/\underline{1}$ array of length $n \geq 2$ s.t. $\frac{1}{2}$ cells are $\underline{0}$.

0	1	2	3	4	5
$\underline{0}$	$\underline{1}$	$\underline{0}$	$\underline{1}$	$\underline{1}$	$\underline{0}$

$$f: \begin{array}{lcl} 0,2,5 & \mapsto & \underline{0} \\ 1,3,4 & \mapsto & \underline{1} \end{array}$$

Output: Find the index of a cell containing $\underline{0}$ ($\texttt{Success}$.

Implementation:

$$\texttt{LV} = \textbf{fix}\,(\lambda\texttt{LasVegas}^{\texttt{nat}}\,.\ \texttt{let k = rand n in}$$
$$\texttt{ifz (f k) then k else LasVegas})$$

Wanted: prove that $\textbf{Prob}^{\infty}(\texttt{LV}, \texttt{Success}) = 1$

Rejection Sampling Algorithm

Implementation: $\texttt{LV} \;=\; \textbf{fix}\,(\lambda\texttt{LasVegas}^{\texttt{nat}}.\ \texttt{let k = rand n in}$
$\texttt{ifz (f k) then k else LasVegas})$

Rejection Sampling Algorithm

Implementation: $\texttt{LV} = \textbf{fix}\,(\lambda\texttt{LasVegas}^{\texttt{nat}}.\ \texttt{let k = rand n in}$
$\texttt{ifz (f k) then k else LasVegas})$

Operational sem.: $\texttt{LV} \xrightarrow{1} \texttt{let}\,\texttt{k} = \texttt{rand}\,\texttt{n}\,\texttt{in}\,\texttt{ifz}\,(\texttt{f}\,\texttt{k})\,\texttt{then}\,\underline{\texttt{k}}\,\texttt{else}\,\texttt{LV}$

Rejection Sampling Algorithm

Implementation: $\texttt{LV} \;=\; \textbf{fix}\,(\lambda\texttt{LasVegas}^{\texttt{nat}}\,.\ \texttt{let k = rand n in}$
$\qquad\qquad\texttt{ifz (f k) then k else LasVegas})$

Operational sem.: $\texttt{LV} \xrightarrow{1} \texttt{let}\ \texttt{k} = \texttt{rand}\ \texttt{n}\ \texttt{in}\ \texttt{ifz}\ (\texttt{f}\ \texttt{k})\ \texttt{then}\ \underline{\texttt{k}}\ \texttt{else}\ \texttt{LV}$

Invariance of the semantics and interpretation of `let` and `ifz`:

$$\begin{aligned} [\![\texttt{LV}]\!]_p &= \sum_{k=0}^{\infty} [\![\texttt{rand n}]\!]_k [\![\texttt{ifz}\,(\texttt{f}\,\texttt{k})\,\texttt{then}\,\underline{\texttt{k}}\,\texttt{else}\,\texttt{LV}]\!]_p \\ &= \frac{1}{n}\cdot\Big(\sum_{f(k)=0 k<n} [\![\underline{k}]\!]_p + \sum_{f(k)\neq 0 k<n} [\![\texttt{LV}]\!]_p\Big) \end{aligned}$$

If $p < n$ & $f(p) = 0$, then $[\![\texttt{LV}]\!]_p = \frac{1}{n} + \frac{1}{n}\cdot\frac{n}{2}\cdot[\![\texttt{LV}]\!]_p$, so $[\![\texttt{LV}]\!]_p = \frac{2}{n}$.

If $p \geq n$ or $f(p) \neq 0$, then $[\![\texttt{LV}]\!]_p = \frac{1}{n}\cdot\frac{n}{2}\cdot[\![\texttt{LV}]\!]_p$, so $[\![\texttt{LV}]\!]_p = 0$.

Rejection Sampling Algorithm

Implementation: $\texttt{LV} = \textbf{fix}(\lambda\texttt{LasVegas}^{\texttt{nat}}.\ \texttt{let k = rand n in}$
$\texttt{ifz (f k) then k else LasVegas})$

Operational sem.: $\texttt{LV} \xrightarrow{1} \texttt{let}\ \texttt{k} = \texttt{rand}\ \texttt{n}\ \texttt{in}\ \texttt{ifz}\ (\texttt{f}\ \texttt{k})\ \texttt{then}\ \underline{\texttt{k}}\ \texttt{else}\ \texttt{LV}$

Invariance of the semantics and interpretation of let and ifz:

$$\begin{aligned} [\![\texttt{LV}]\!]_p &= \sum_{k=0}^{\infty} [\![\texttt{rand n}]\!]_k [\![\texttt{ifz}\,(\texttt{f}\,\texttt{k})\,\texttt{then}\,\underline{\texttt{k}}\,\texttt{else}\,\texttt{LV}]\!]_p \\ &= \frac{1}{n} \cdot \Big(\sum_{f(k)=0 k<n} [\![\underline{k}]\!]_p + \sum_{f(k)\neq 0 k<n} [\![\texttt{LV}]\!]_p \Big) \end{aligned}$$

If $p < n$ & $f(p) = 0$, then $[\![\texttt{LV}]\!]_p = \frac{1}{n} + \frac{1}{n} \cdot \frac{n}{2} \cdot [\![\texttt{LV}]\!]_p$, so $[\![\texttt{LV}]\!]_p = \frac{2}{n}$.

If $p \geq n$ or $f(p) \neq 0$, then $[\![\texttt{LV}]\!]_p = \frac{1}{n} \cdot \frac{n}{2} \cdot [\![\texttt{LV}]\!]_p$, so $[\![\texttt{LV}]\!]_p = 0$.

Adequacy Lemma, the probability that LV converges:

$$\begin{aligned} \textbf{Prob}^{\infty}(\texttt{LV}, \texttt{Success}) &= \sum_p \textbf{Prob}^{\infty}(\texttt{LV}, \underline{p}) = \sum_p [\![\texttt{LV}]\!]_p \\ &= \sum_{f(p)=0; p<n} \frac{2}{n} = \frac{n}{2} \cdot \frac{2}{n} = 1 \end{aligned}$$

Metropolis-Hasting Algorithm

Input: μ a distribution on $\mathbb{R}$ with density π:
$\mu(U) = \int_U \pi(x)dx$, but we only know $\gamma\pi$.

Output: Markov Chain x_n converging to
a random variable x with law μ

Metropolis-Hasting Algorithm

Input: μ a distribution on $\mathbb{R}$ with density π:
$\mu(U) = \int_U \pi(x)dx$, but we only know $\gamma\pi$.

Output: Markov Chain x_n converging to
a random variable x with law μ

Program:

```
MH = fix (λMetHast^(nat→nat).λn^nat. if n=0 then x_0 else
        let x = MetHast (n-1) in
            let y = gauss x in
                let z = bernouilli(α(x,y)) in
                    if z = 0 then x else y)
```

Wanted: $\mathtt{MH}(\underline{n})$ is a Markov Chain converging to a random var. of law μ.

Metropolis-Hasting Algorithm

Input: μ a distribution on $\mathbb{R}$ with density π:
$\mu(U) = \int_U \pi(x)dx$, but we only know $\gamma\pi$.

Output: Markov Chain x_n converging to
a random variable x with law μ

Program:
$$\texttt{MH} = \textbf{fix}\,(\lambda \texttt{MetHast}^{\texttt{nat}\to\texttt{nat}}.\lambda \texttt{n}^{\texttt{nat}}.\ \texttt{if n=0 then } \texttt{x}_0 \texttt{ else}$$
$$\texttt{let x = MetHast (n-1) in}$$
$$\texttt{let y = gauss x in}$$
$$\texttt{let z = bernouilli}(\alpha\texttt{(x,y)) in}$$
$$\texttt{if } z = 0 \texttt{ then x else y})$$

Wanted: $\texttt{MH}(\underline{n})$ is a Markov Chain converging to a random var. of law μ.

Operational Semantics:

$\texttt{MH}(\underline{0}) \to \texttt{x}_0$ thus, $\textbf{Prob}(\texttt{MH}(\underline{0}), U) = \delta_{x_0}(U)$

$$\texttt{MH}(\underline{n+1}) \to M = \texttt{let}\, x{=}\texttt{MH}(\underline{n})\,\texttt{in}\,\texttt{let}\, y{=}\texttt{gauss}\, x\, \texttt{in}$$
$$\texttt{let}\, z{=}\texttt{bernoulli}(\underline{\alpha}(x,y))\,\texttt{in}\,\texttt{ifz}(z,x,y)$$

Metropolis-Hasting Algorithm

$$\texttt{MH}(\underline{n+1}) \to M = \texttt{let}\, x{=}\texttt{MH}(\underline{n})\,\texttt{in}\,\texttt{let}\, y{=}\texttt{gauss}\, x\,\texttt{in}$$
$$\texttt{let}\, z{=}\texttt{bernoulli}(\underline{\alpha}(x,y))\,\texttt{in}\,\texttt{ifz}(z,x,y)$$

Adequacy/Invariance/Interpretation:

$$\textbf{Prob}(\texttt{MH}(\underline{n+1}), U) = [\![\texttt{MH}(\underline{n+1})]\!](U) = [\![M]\!](U)$$
$$= \int_{\mathbb{R}} [\![N]\!](\delta_r)(U)\, [\![\texttt{MH}(\underline{n})]\!](dr) = \int_{\mathbb{R}} P_{\texttt{MH}}(r,U)\, \textbf{Prob}(\texttt{MH}(\underline{n}), dr)$$
$$P_{\texttt{MH}}(r,U) = \delta_r(U)\left(1 - \int_{\mathbb{R}} \alpha(r,t)g(t,r)\lambda(dt)\right) + \int_U \alpha(r,t)g(t,r)\lambda(dt).$$

Thus it is a Markov-Chain whose law is defined with respect to the kernel $P_{\texttt{MH}}(r,U)$. It is standard to prove that μ is its invariant measure.

Example

Operational Sem., Invariance and Adequacy imply Correctness

Contributions

- The study of semantics of discrete and continuous probabilistic programming
- Full Abstraction for Probabilistic Coherent Spaces and `Nat` PPCF
- Adequacy for Measurable Cones and Measurable Stable functions and `Real` PPCF
- Use of quantitative approach of LL: $[\![M]\!] = \sum [\![M]\!]_\mu x^\mu$

Next steps

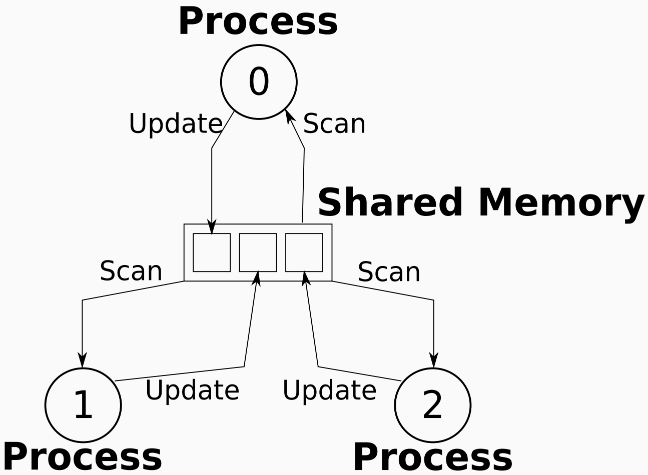
- Compare with Quasi Borel Spaces
- Extract model of Linear Logic from Measurable Cones and Measurable Stable Functions

Perspectives

- Combine differentiation and probability
- Certification in proof assistant

Table of contents

Distributed Systems


Process
0
Update
Scan
Shared Memory
Scan
Scan
1
Update
Update
2
Process
Process

Asynchronous computations

Distributed System

A fixed family of $n+1$ processes communicate by **Update** and **Scan** of their **local** memory into a shared **global** memory.

Asynchronous

- For each process, the kth Scan follows the kth Update
- Update and Scan are **mutually exclusive**
- no delay or order restriction

Interleaving Trace

Each execution of a protocol is given by an **interleaving trace** $T \in \{U_i, S_i \mid i \in [n] = \{0 \cdots n\}\}^*$ well-bracketed.

Example for 3 processes, 2 rounds: $U_1\ U_2\ S_1\ U_0\ S_0\ S_2\ U_1\ U_0\ S_1\ U_2\ S_2\ S_0$

Operational Semantics

Consider a program with $n+1$ processes and $(r_i)_{i\in[n]}$ rounds.

State a pair $s = (\ell, m)$ where

- $\ell = (\ell_i)_{i\in[n]}$ **local** memories (one register per process)
- $m = (m_i)_{i\in[n]}$ **global** memory (one register per process)

Initial state s_0*:* $\ell_i = i$ and $m_i = \bot$

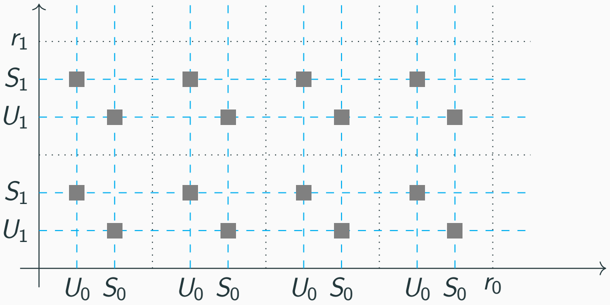
Operational Equivalence

Two interleaving traces T, T' are operationaly equivalent when

$$s_0 \xrightarrow{T}{}^{*} s \qquad \text{iff} \qquad s_0 \xrightarrow{T}{}^{*} s$$

Directed Algebraic Topology

Pospace $\mathbb{X}_n = \prod_{i \in [n]} [0, r_i] \setminus \bigcup_{\substack{i,j \in [n] \\ k \in [r_i],\ l \in [r_j]}} U_i^k \cap S_j^l$

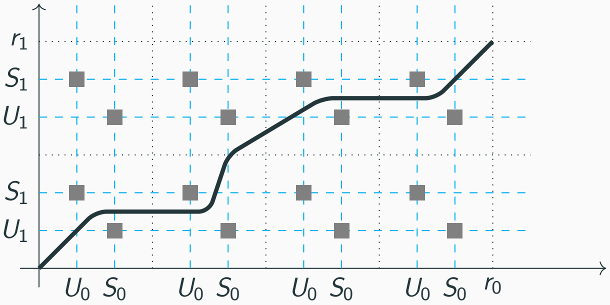


[Fajstrup-Goubault-Haucourt-Raussen 2016]

Directed Algebraic Topology

Pospace $\mathbb{X}_n = \prod_{i\in[n]} [0, r_i] \setminus \bigcup_{\substack{i,j\in[n] \\ k\in[r_i],\ l\in[r_j]}} U_i^k \cap S_j^l$

Dipath $\alpha : [0,1] \to \mathbb{X}_n$ continuous and non decreasing



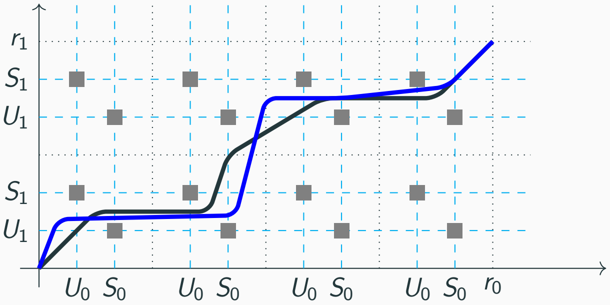
[Fajstrup-Goubault-Haucourt-Raussen 2016]

Directed Algebraic Topology

Pospace $\mathbb{X}_n = \prod_{i \in [n]} [0, r_i] \setminus \bigcup_{\substack{i,j \in [n] \\ k \in [r_i],\ l \in [r_j]}} U_i^k \cap S_j^l$

Dipath $\alpha : [0,1] \to \mathbb{X}_n$ continuous and non decreasing

Dihomotopy $h : \overrightarrow{[0,1]} \times [0,1] \to \mathbb{X}_n$ continuous non decreasing



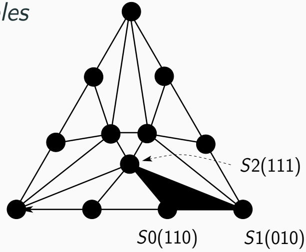
[Fajstrup-Goubault-Haucourt-Raussen 2016]

Consider a program with $n+1$ processes and $(r_i)_{i \in [n]}$ rounds.

Protocol Complex [Herlihy-Kozlov-Rasjbaum 2013]

- **Vertex:** (process, local memory)
- **Maximal Simplex:** $\{(0, \ell_0), \ldots, (n, \ell_n)\}$ where ℓ_i is the local view by process i of the global execution.

Examples



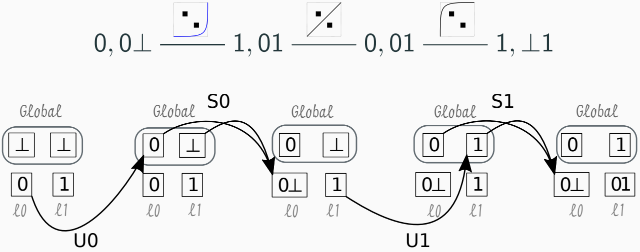
$U_1 \; S_1 \; U_0 \; S_0 \; U_2 \; S_2$

Consider a program with $n+1$ processes and $(r_i)_{i\in[n]}$ rounds.

Protocol Complex [Herlihy-Kozlov-Rasjbaum 2013]

- **Vertex:** (process, local memory)
- **Maximal Simplex:** $\{(0,\ell_0),\dots,(n,\ell_n)\}$ where ℓ_i is the local view by process i of the global execution.

Examples

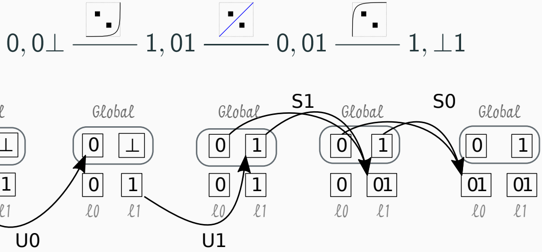


Consider a program with $n+1$ processes and $(r_i)_{i\in[n]}$ rounds.

Protocol Complex [Herlihy-Kozlov-Rasjbaum 2013]

- **Vertex:** (process, local memory)
- **Maximal Simplex:** $\{(0,\ell_0),\ldots,(n,\ell_n)\}$ where ℓ_i is the local view by process i of the global execution.

Examples



Theorem [Herlihy-Shavit 1999]

If the Protocol Complex is **contractible** then, the consensus is impossible.

Proof sketch

Assume there is an algorithm δ solving the task, for any execution.

$$0, 0\bot \;\text{———}\; 1, 01 \;\text{———}\; 0, 01 \;\text{———}\; 1, \bot 1 \qquad \xrightarrow{\delta} \qquad \bullet 0 \qquad \bullet 1$$

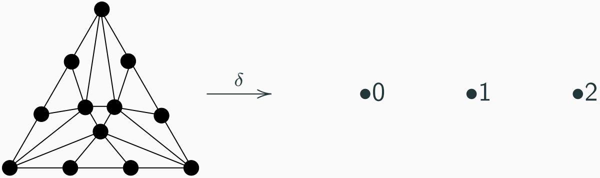
Impossibility Results

Theorem [Herlihy-Shavit 1999]

If the Protocol Complex is **contractible** then, the consensus is impossible.

Proof sketch

Assume there is an algorithm δ solving the task, for any execution.



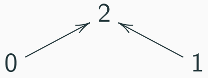
Geometrical Interpretation of Asynchronous Computability

Theorem (Goubault-Mimram-Tasson 2015)

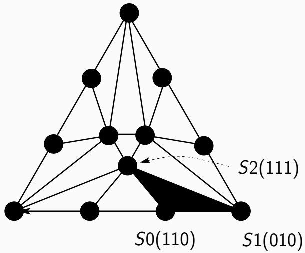
Equivalence between Simplexes, Interval Orders, Dipath, Traces.

$[U_1\ U_0\ S_1\ S_0\ U_2\ S_2]$

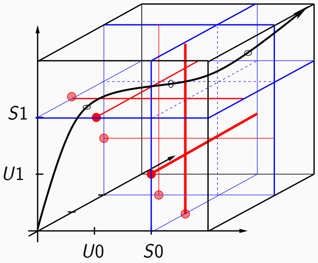
Interleaving Trace$_{/\approx}$



Interval Order



Simplex



Dipath$_{/\leftrightsquigarrow}$

Contributions

- The operational semantics of execution traces
- The equivalence between two geometric semantics

Next steps

- Generalise this equivalence to other communication primitives and failures
- Use this equivalence to transfer properties from one model to the other

Perspectives

- Combine differentiation and distributed calculus
- Describe a denotational semantics of distributed systems

Table of contents

Perspectives

Differential λ-calculus:

Contribution: A monad for mixed linear non linear variables.

Perspectives: Toward mixed subsitution and theory of derivation.

Probabilistic Programming:

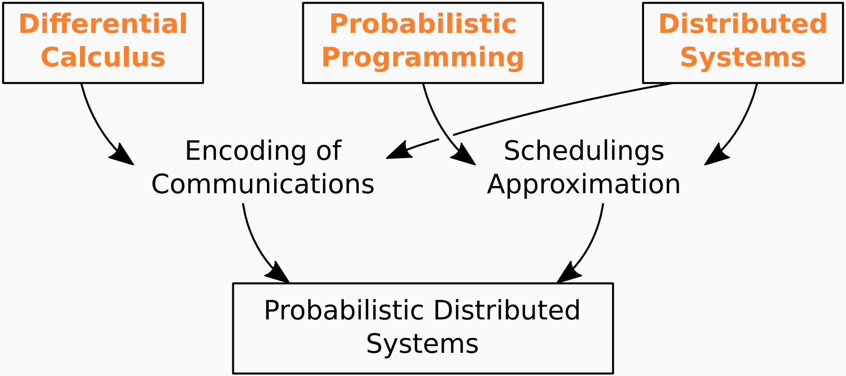
Contribution: Discrete and Continuous semantics.

Perspectives: Comparison with other models, Full Abstraction, Linear Logic, Recursive types,...

Distributed Computing:

Contribution: Equivalence between geometric semantics.

Perspectives: Generalise to different communication primitives and systematic method to produce protocol complexes


Differential Calculus
Probabilistic Programming
Distributed Systems
Encoding of Communications
Schedulings Approximation
Probabilistic Distributed Systems