université PARIS DIDEROT PARIS 7

Semantics of
Probabilistic and Differential Programming

Workshop on program transformations at NeurIPS

Christine Tasson (tasson@irif.fr)

December 2019

Institut de Recherche en Informatique Fondamentale

Every programmer can perform data analysis by describing models as programs and key operations (inference and gradient) computations are delegated to compiler.

Probabilistic programming languages

BUGS (Spiegelhalter et al. 1995), BLOG (Milch et al. 2005), Church (Goodman et al. 2008), WebPPL (Goodman et al. 2014), Venture (Mansinghka et al. 2014), Anglican (Wood et al. 2015), Stan (Stan Development Team 2014), Hakaru (Narayanan et al., 2016) BayesDB (Mansinghka et al. 2017), Edward (Tran et al.Tran et al. 2017), Birch (Murray et al. 2018), Turing (Ge et al. 2018), Gen (Cusumano-Towner et al. 2019), Pyro (Bingham et al. 2019), ...

Differential programming languages

Theano (Bergstra et al. 2010), Tensorflow 1.0 (Abadi et al. 2016, Yu et al.2018), Tangent (van Merrienboer et al. 2018), Autograd (Maclaurin et al. 2015), TensorFlow Eager Mode (Shankar and Dobson 2017), Chainer (Tokui 2018), PyTorch (PyTorch 2018), and JAX (Frostig et al. 2018), ...

Probabilistic Programming

Bayesian Inference

Sampling

Idea: How to model probability distributions by programs

```
1  def plinko(n):
2    if (n==0):
3      return 0
4    else:
5      if coin():
6        return plinko(n-1)+1
7      else:
8        return plinko(n-1)-1
```

By Matemateca (IME USP)

Sampling

Idea: How to model probability distributions by programs

```
sample(plinko(4))
> 2
```

```
1  def plinko(n):
2    if (n==0):
3      return 0
4    else:
5      if coin():
6        return plinko(n-1)+1
7      else:
8        return plinko(n-1)-1
```

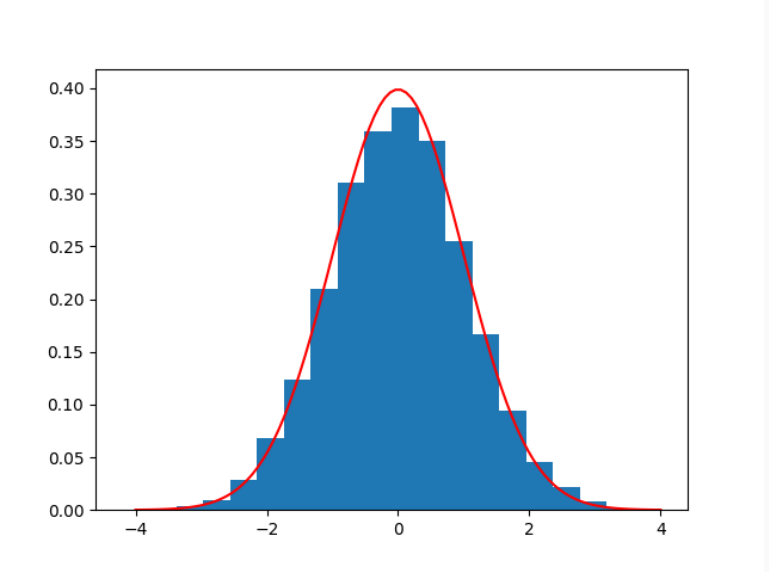
Sampling

Idea: How to model probability distributions by programs

```
1  def plinko(n):
2    if (n==0):
3      return 0
4    else:
5      if coin():
6        return plinko(n-1)+1
7      else:
8        return plinko(n-1)-1
```

```
sample(plinko(4))
> 2
```

```
nSample(plinko(4), 1000)
plot(gaussian(0,1))
```



What is Bayesian Inference

Gender Bias (Laplace): Paris, from 1745 to 1770

$f0 = 241\,945$ females out of $B0 = 493\,472$ births (49%).

What is Bayesian Inference

Gender Bias (Laplace): Paris, from 1745 to 1770

$f0 = 241\,945$ females out of $B0 = 493\,472$ births (49%).

What is the probability to be born female ?

- female births are independent and follow the same law with bias θ
- the probability to get f females out of B births is

$$P(f|\theta, B) = \binom{B}{f} \theta^f (1-\theta)^{B-f}$$

Novelty: the bias θ to be born female follows a probabilistic distribution.

What is Bayesian Inference

Gender Bias (Laplace): Paris, from 1745 to 1770
$f0 = 241\,945$ females out of $B0 = 493\,472$ births (49%).

What is the probability to be born female ?

- female births are independent and follow the same law with bias θ
- the probability to get f females out of B births is

$$P(f|\theta, B) = \binom{B}{f} \theta^f (1-\theta)^{B-f}$$

Novelty: the bias θ to be born female follows a probabilistic distribution.

Inference paradigm: what is the law of θ conditioned on f and B?

- Sample θ from a postulated distribution π (prior)
- Simulate data f from the outcome θ (likelihood)
- Infer the distribution of θ (posterior) by **Bayes Law**

$$P(\theta \mid f, B) = \frac{P(f \mid \theta, B)\ \pi(\theta)}{\int_\theta P(f \mid \theta, B)\ \pi(\theta)} = \alpha \cdot P(f \mid \theta, B)\ \pi(\theta)$$

Conditioning and inference

```
# model
def fBirth(theta, B):
    if (B == 0):
        return 0
    else:
        f = flip(theta)
        return f + fBirth(theta, B-1)

# parameter (prior)
theta = uniform(0,1)

# data 1747 - 1783
f0 = 241 945
B0 = 493 472

# inference (posterior)
infer(fBirth, theta, f0, B0)
```

Idea: adjust `theta` distribution by comparison to data by simulation.

Inference by rejection sampling

```
1  # prior: Unit -> S
2  def guesser() :
3      sample(uniform(0,1))
4
5  # predicate: int x int -> (S -> Boolean)
6  def checker(f0, B0) :
7      lambda theta: gBirth(theta, B0) == f0
8
9  # infer: (Unit -> S) -> (S -> Boolean) -> S
10 def rejection(guesser, checker(f0, B0)):
11     theta = guesser()
12     if checker(f0, B0)(theta):
13         return theta
14     else:
15         rejection(guesser, checker(f0, B0))
```

Problem: inefficient, hence other approximated methods

Inference by Metropolis-Hasting

Infer θ by **Bayes Law**: $P(\theta \mid f, B) = \alpha \cdot P(f \mid \theta, B)\ \pi(\theta)$

```
1  # proportion: S x S -> float
2  def proportion(x, y):
3      return P(f | x, B0) / P(f | y, B0)
4
5  # Metropolis-Hasting: int * int * int -> S
6  def metropolis(n, f0, B0):
7      if (n=0):
8          return f0/B0
9      else:
10         x = metropolis(n-1, f0, B0)
11         y = gaussian(x, 1)
12         z = bernouilli(proportion(x, y))
13         if (z == 0):
14             return x
15         else:
16             return y
```

Probabilistic Programming

Semantics

Problems in semantics

- Prove formally the correspondence between algorithms, **implementations** and mathematics.
- Prove that two programs have **equivalent behavior**

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

Denotational Semantics describes **what** probabilistic programs compute

Problems in semantics

- Prove formally the correspondence between algorithms, **implementations** and mathematics.
- Prove that two programs have **equivalent behavior**

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

Proba(M, N) is the probability p that M reduces to N in one step, $M \xrightarrow{p} N$ defined by induction on the structure of M:

- $(\lambda x.M)N \xrightarrow{1} M[N/x]$
- $\mathrm{coin} \xrightarrow{1/2} \underline{0}$
- $\mathrm{coin} \xrightarrow{1/2} \underline{1}$
- ...

Denotational Semantics describes **what** probabilistic programs compute

Problems in semantics

- Prove formally the correspondence between algorithms, **implementations** and mathematics.
- Prove that two programs have **equivalent behavior**

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

Proba(M, N) is the probability p that M reduces to N in one step, $M \xrightarrow{p} N$ defined by induction on the structure of M:

- $(\lambda x.M)N \xrightarrow{1} M[N/x]$
- $\mathsf{coin} \xrightarrow{1/2} \underline{0}$
- $\mathsf{coin} \xrightarrow{1/2} \underline{1}$
- ...

Denotational Semantics describes **what** probabilistic programs compute

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

- If M has type `nat`, then $[\![M]\!]$ a *discrete* distribution over integers
- If M has type `real`, then $[\![M]\!]$ a *continuous* distribution over reals

Operational Semantics on an example

(Borgström-Dal Lago-Gordon-Szymczak ICFP'16)

```
def addCoins():
  a = coin
  b = coin
  c = coin
  return (a + b + c)
```

- $(\lambda x.M)N \xrightarrow{1} M[N/x]$
- $\mathsf{coin} \xrightarrow{1/2} \underline{0}$
- $\mathsf{coin} \xrightarrow{1/2} \underline{1} \quad \ldots$

$$\mathrm{addCoins}() \xrightarrow{1} \begin{array}{l}\mathtt{a = coin}\\ \mathtt{b = coin}\\ \mathtt{c = coin}\\ \mathtt{(a + b + c)}\end{array} \xrightarrow{1/2} \begin{array}{l}\mathtt{a = 0}\\ \mathtt{b = coin}\\ \mathtt{c = coin}\\ \mathtt{(a + b + c)}\end{array} \xrightarrow{1/2} \begin{array}{l}\mathtt{a = 0}\\ \mathtt{b = 1}\\ \mathtt{c = coin}\\ \mathtt{(a + b + c)}\end{array} \xrightarrow{1/2} \begin{array}{l}\mathtt{a = 0}\\ \mathtt{b = 1}\\ \mathtt{c = 1}\\ \mathtt{(a + b + c)}\end{array}$$

$$\xrightarrow{1} \begin{array}{l}\mathtt{b = 1}\\ \mathtt{c = 1}\\ \mathtt{(0 + b + c)}\end{array} \xrightarrow{1} \begin{array}{l}\mathtt{c = 1}\\ \mathtt{(0 + 1 + c)}\end{array} \xrightarrow{1} \mathtt{(0 + 1 + 1)} \xrightarrow{1} \mathtt{2}$$

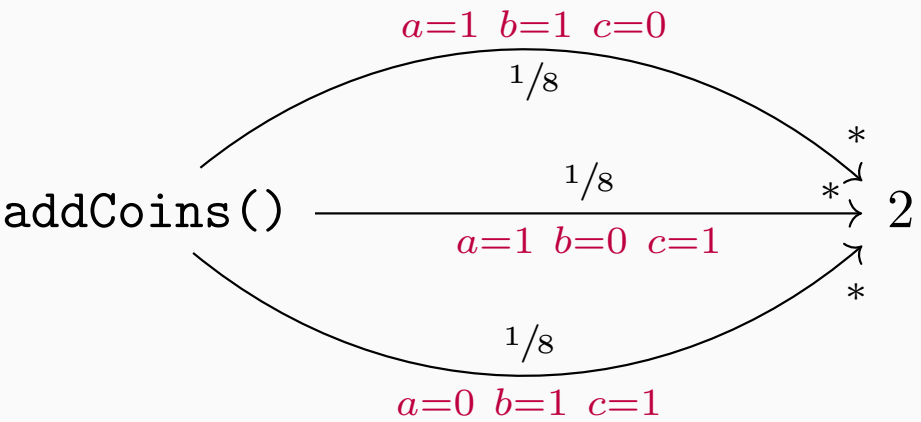
Operational Semantics on an example

(Borgström-Dal Lago-Gordon-Szymczak ICFP'16)

```
def addCoins():
  a = coin
  b = coin
  c = coin
  return (a + b + c)
```

- $(\lambda x.M)N \xrightarrow{1} M[N/x]$
- $\mathsf{coin} \xrightarrow{1/2} \underline{0}$
- $\mathsf{coin} \xrightarrow{1/2} \underline{1} \quad \ldots$

$$\texttt{addCoins()} \xrightarrow{1} \begin{array}{l}\texttt{a = coin}\\ \texttt{b = coin}\\ \texttt{c = coin}\\ \texttt{(a + b + c)}\end{array} \xrightarrow[a=0]{1/2} \begin{array}{l}\texttt{a = 0}\\ \texttt{b = coin}\\ \texttt{c = coin}\\ \texttt{(a + b + c)}\end{array} \xrightarrow[b=1]{1/2} \xrightarrow[c=1]{1/2} \xrightarrow{1\ *} 2$$



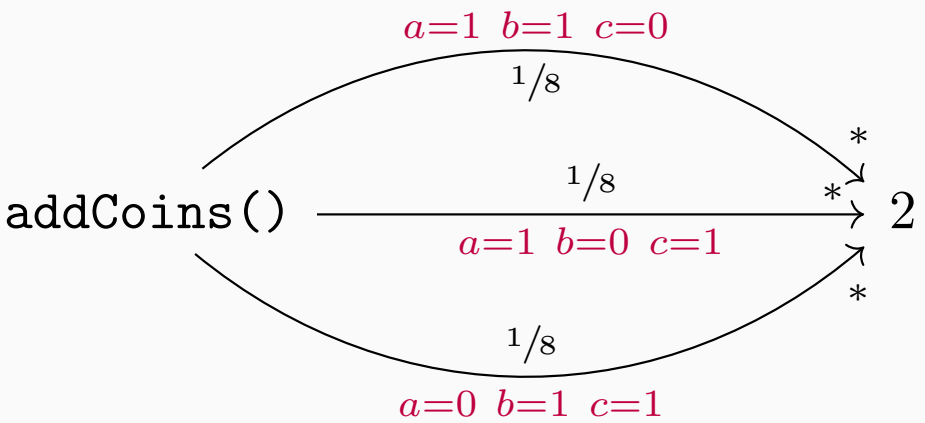
Operational Semantics on an example

(Borgström-Dal Lago-Gordon-Szymczak ICFP'16)

```
def addCoins():
  a = coin
  b = coin
  c = coin
  return (a + b + c)
```

- $(\lambda x.M)N \xrightarrow{1} M[N/x]$
- $\mathsf{coin} \xrightarrow{1/2} \underline{0}$
- $\mathsf{coin} \xrightarrow{1/2} \underline{1} \quad \ldots$

$$\texttt{addCoins()} \xrightarrow{1} \begin{array}{l}\texttt{a = coin}\\ \texttt{b = coin}\\ \texttt{c = coin}\\ \texttt{(a + b + c)}\end{array} \xrightarrow[a=0]{1/2} \begin{array}{l}\texttt{a = 0}\\ \texttt{b = coin}\\ \texttt{c = coin}\\ \texttt{(a + b + c)}\end{array} \xrightarrow[b=1]{1/2} \xrightarrow[c=1]{1/2} \xrightarrow{1\ *} 2$$



$$\textbf{Proba}^{\infty}(\texttt{addCoins()}, 2) = \frac{3}{8}$$

Operational Semantics

$\mathbf{Proba}^{\infty}(M, N)$ is the proba. that M reduces to N in *any* number of steps

Behavioral equivalence:

$$M_1 \simeq M_2 \quad \textbf{iff} \quad \forall C[\,],\ \mathbf{Proba}^{\infty}(C[M_1], \underline{0}) = \mathbf{Proba}^{\infty}(C[M_2], \underline{0})$$

```
1  def addCoins1():
2      a = coin
3      b = coin
4      c = coin
5      return (a + b + c)
```

```
1  def addCoins2():
2      b = coin
3      a = coin
4      c = coin
5      return (a + b + c)
```

```
1  def infer1(f0, B0):
2      rejection(guesser, checker(f0, B0)):
3
4  def infer2(f0, B0):
5      metropolis(f0, B0, 1000)
```

" The developers of probabilistic programming languages need to ensure that the implementation of compilers, optimizers, and inference algorithms do not have bugs." (van de Meent-Paige-Yang-Wood 2018)

Denotational semantics allows to define the mathematical meaning of every probabilistic program.

Problem: Measurable sets and measurable functions are not suitable to interpret higher order functional probabilistic programming languages.

The evaluation map $\mathrm{ev} : \mathcal{F}(\mathbb{R}, \mathbb{R}) \times \mathbb{R} \to \mathbb{R}$ with $\mathrm{ev}(f, r) = f(r)$ is not measurable whatever measurable sets we put on the set $\mathcal{F}(\mathbb{R}, \mathbb{R})$ of measurable functions between reals endowed with borel sets.

(Aumann 1961)

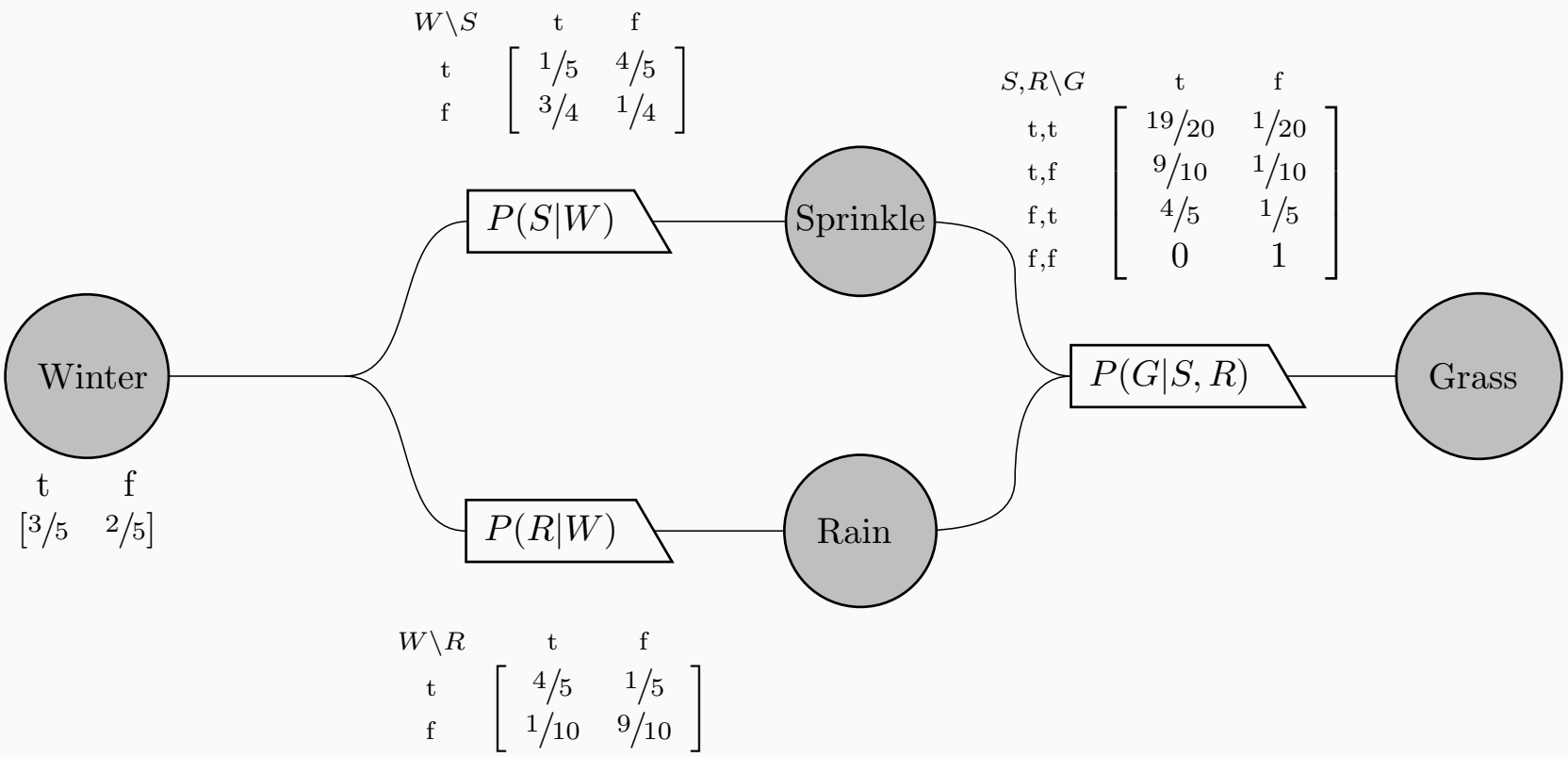
Semantics for HOPPL with continuous probability

- Quasi Borel Spaces
  (Kammar-Staton+Heunen-Yang LICS'17, +Vakar POPL'19)
- Measurable postive Cones and Stable maps
  (Ehrhard-Pagani-T. POPL'18)
- Ordered Banach Spaces and Regular maps
  (Dahlqvist-Kozen POPL'20)

Applications

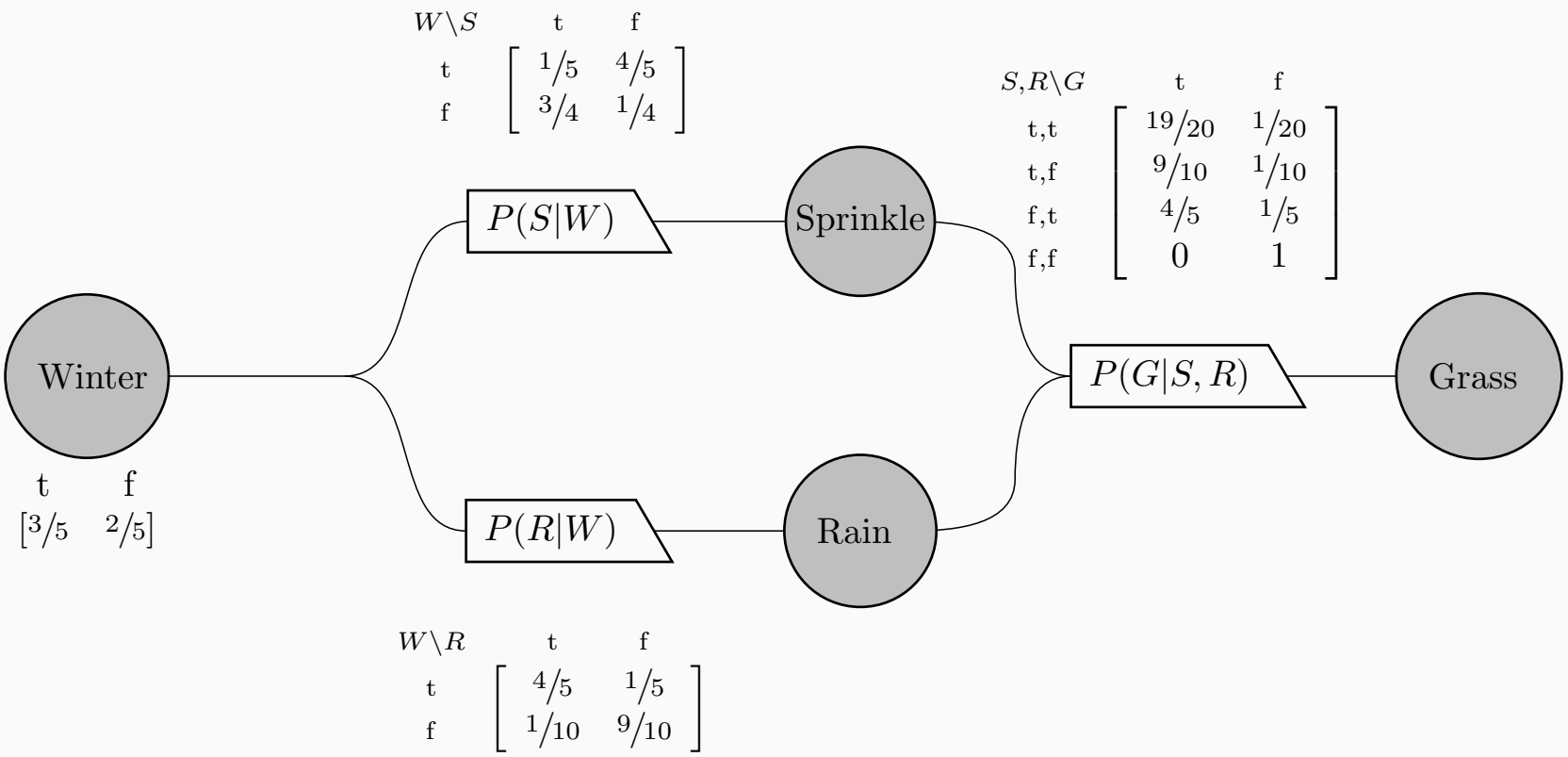
- Probabilistic programming inference via intensional semantics for FO
  (Castellan-Paquet ESOP'19)
- Well-typed inference programs are sound by construction.
  (Lew-Cusumano-Towner-Sherman-Carbin-Mansinghka POPL'20)
- Denotational semantics and program analysis for score estimators
  (Lee-Yu-Rival-Yang POPL'20)

Semantics of a Bayesian Network



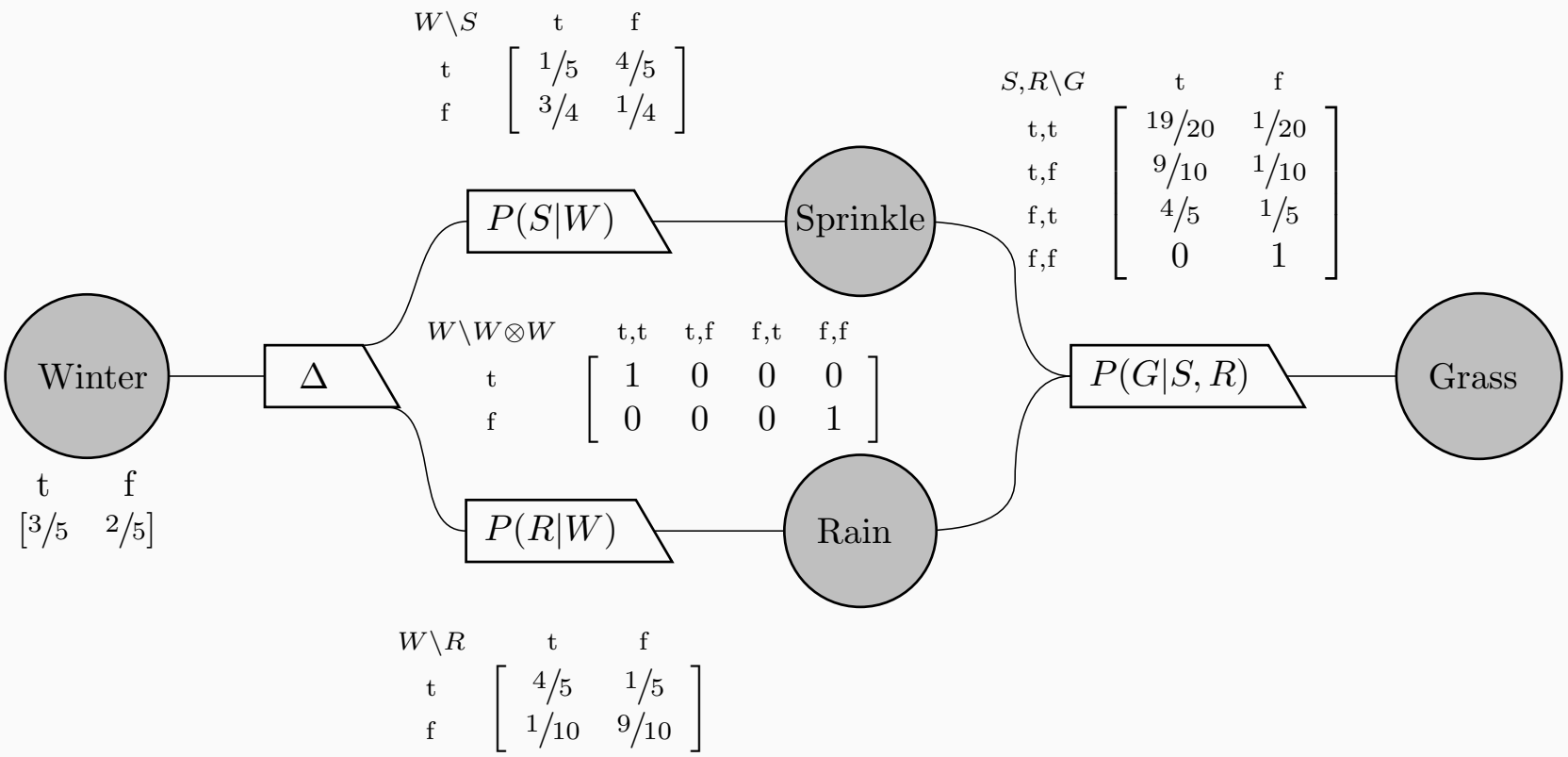
$$p(S) = \left(\sum_{a \in \{\mathrm{t},\mathrm{f}\}} P(S|W)_{a,b} \cdot p(W)_a \right)_{b \in \{\mathrm{t},\mathrm{f}\}}$$

Semantics of a Bayesian Network



$$\begin{aligned} p(W)\ P(S|W) &= p(S) \\ p(W)\ P(R|W) &= p(R) \end{aligned} \quad \text{and} \quad (p(S) \otimes p(R))\ P(G|S,R) = p(G)$$

Semantics of a Bayesian Network



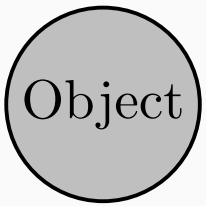
$$p(W)\ \Delta\ (P(S|W) \otimes P(R|W))\ P(G|S,R)\ =\ p(G)$$

(Jacobs-Kissinger-Zanasi FOSSACS'19)

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.



$(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

closure: $\mathbf{P}(\mathbf{X})^{\perp\perp} = \mathbf{P}(\mathbf{X})$ where
$\forall P \subseteq (\mathbb{R}^+)^{|X|},\ P^{\perp} = \{v \in (\mathbb{R}^+)^{|X|}\ ;\ \forall u \in P,\ \sum_{a \in |X|} u_a v_a \leq 1\}$

bounded covering:

$$\forall a \in |X|,\ \exists v \in P(X)\ ;\ v_a \neq 0 \quad \text{and} \quad \exists p > 0,\ ;\ \forall v \in \mathrm{P}(X),\ v_a \leq p.$$

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.

Object

$(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

Type

$A, B ::= \mathtt{nat} \mid A \to B \mid \ldots$ are interpreted by objects $[\![A]\!] = (|A|, \mathrm{P}(A))$ defined by induction on A.

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.

Object $(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

Type

$A, B ::= \mathtt{nat} \mid A \to B \mid \ldots$ are interpreted by objects $[\![A]\!] = (|A|, \mathrm{P}(A))$ defined by induction on A.

- unit type 1: $|1| = \{()\}$ and $\mathrm{P}(1) = [0, 1]$

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.

Object $(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

Type

$A, B ::= \mathtt{nat} \mid A \to B \mid \ldots$ are interpreted by objects $[\![A]\!] = (|A|, \mathrm{P}(A))$ defined by induction on A.

- unit type 1: $|1| = \{()\}$ and $\mathrm{P}(1) = [0, 1]$
- $\mathcal{B} = 1 \oplus 1$: $|\mathcal{B}| = \{\mathrm{t}, \mathrm{f}\}$ and $\mathrm{P}(\mathcal{B}) = \{x \cdot \mathrm{t} + y \cdot \mathrm{f} \mid x + y \leq 1\}$

$$p(W) = \left[3/5, 2/5\right] \in \mathrm{P}(\mathcal{B}).$$

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.

Object $(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

Type

$A, B ::= \mathtt{nat} \mid A \to B \mid \ldots$ are interpreted by objects $[\![A]\!] = (|A|, \mathrm{P}(A))$ defined by induction on A.

- unit type 1: $|1| = \{()\}$ and $\mathrm{P}(1) = [0, 1]$
- $\mathcal{B} = 1 \oplus 1$: $|\mathcal{B}| = \{\mathrm{t}, \mathrm{f}\}$ and $\mathrm{P}(\mathcal{B}) = \{x \cdot \mathrm{t} + y \cdot \mathrm{f} \mid x + y \leq 1\}$

  $p(W) = \left[3/5, 2/5\right] \in \mathrm{P}(\mathcal{B})$.
- $\mathtt{nat} = 1 \oplus \mathtt{nat}$: $|\mathtt{nat}| = \mathbb{N}$ and $\mathrm{P}(\mathtt{nat})$ sub-proba distrib. over $\mathbb{N}$

Denotational Semantics:

(Danos-Ehrhard 2011)

Probabilistic Coherent Spaces (**Pcoh**) an adequate model of probabilistic functional programming with discrete probability.

Object $(|X|, \mathrm{P}(X))$

- the universe $|X|$ is a (potentially infinite) set of final states
- a set of vectors $\mathrm{P}(X) \subseteq (\mathbb{R}^+)^{|X|}$

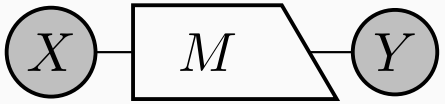
Type

$A, B ::= \mathtt{nat} \mid A \to B \mid \ldots$ are interpreted by objects $[\![A]\!] = (|A|, \mathrm{P}(A))$ defined by induction on A.

- unit type 1: $|1| = \{()\}$ and $\mathrm{P}(1) = [0, 1]$
- $\mathcal{B} = 1 \oplus 1$: $|\mathcal{B}| = \{\mathrm{t}, \mathrm{f}\}$ and $\mathrm{P}(\mathcal{B}) = \{x \cdot \mathrm{t} + y \cdot \mathrm{f} \mid x + y \leq 1\}$
  $p(W) = \left[\frac{3}{5}, \frac{2}{5}\right] \in \mathrm{P}(\mathcal{B})$.
- $\mathtt{nat} = 1 \oplus \mathtt{nat}$: $|\mathtt{nat}| = \mathbb{N}$ and $\mathrm{P}(\mathtt{nat})$ sub-proba distrib. over $\mathbb{N}$
- $\mathcal{B}^* = 1 \oplus (\mathcal{B} \otimes \mathcal{B}^*)$: $|\mathcal{B}^*| = \{\epsilon\} \cup \{b_1 \cdot \cdots \cdot b_n \mid n \in \mathbb{N},\ b_i \in |\mathcal{B}|\}$ and $\mathrm{P}(\mathcal{B}^*)$ sub-probability distribution over words of booleans.

Semantics: Probabilistic Coherent Spaces

(Danos-Ehrhard 2011)

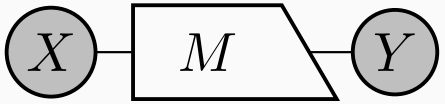


$\in (\mathbb{R}^+)^{|X|\times|Y|}$ is a matrix

$$\forall x \in \underset{\subseteq(\mathbb{R}^+)^{|X|}}{\mathrm{P}(X)}\ ,\ M\cdot x = \left(\sum_{a\in|X|} M_{a,b}\, x_a\right)_{b\in|Y|} \in \underset{\subseteq(\mathbb{R}^+)^{|Y|}}{\mathrm{P}(Y)}$$

Semantics: Probabilistic Coherent Spaces

(Danos-Ehrhard 2011)



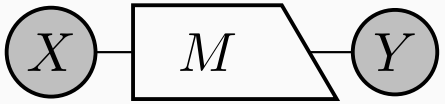
$\in (\mathbb{R}^+)^{\mathcal{M}_{\mathrm{fin}}(|X|)\times|Y|}$ is a matrix

$$\forall x \in \underset{\subseteq(\mathbb{R}^+)^{|X|}}{\mathrm{P}(X)}\ ,\ M(x) = \left(\sum_{m\in\mathcal{M}_{\mathrm{fin}}(|X|)} M_{m,b} \prod_{a\in m} x_a^{m(a)} \right)_{b\in|Y|} \in \underset{\subseteq(\mathbb{R}^+)^{|Y|}}{\mathrm{P}(Y)}$$

Semantics: Probabilistic Coherent Spaces

(Danos-Ehrhard 2011)

Morphism



$\in (\mathbb{R}^+)^{\mathcal{M}_{\mathrm{fin}}(|X|)\times|Y|}$ is a matrix

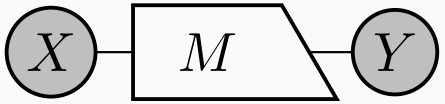
$$\forall x \in \underset{\subseteq(\mathbb{R}^+)^{|X|}}{\mathrm{P}(X)}\ ,\ M(x) = \left(\sum_{m\in\mathcal{M}_{\mathrm{fin}}(|X|)} M_{m,b} \prod_{a\in m} x_a^{m(a)} \right)_{b\in|Y|} \in \underset{\subseteq(\mathbb{R}^+)^{|Y|}}{\mathrm{P}(Y)}$$

Program

$M, N ::= x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid \underline{n} \mid \mathrm{coin} \mid \ldots$

are interpreted by morphisms, by induction on M

Semantics: Probabilistic Coherent Spaces (Danos-Ehrhard 2011)

Morphism  $\in (\mathbb{R}^+)^{\mathcal{M}_{\text{fin}}(|X|)\times|Y|}$ is a matrix

$$\forall x \in \underset{\subseteq(\mathbb{R}^+)^{|X|}}{\mathrm{P}(X)}\ ,\ M(x) = \left(\sum_{m\in\mathcal{M}_{\text{fin}}(|X|)} M_{m,b} \prod_{a\in m} x_a^{m(a)} \right)_{b\in|Y|} \in \underset{\subseteq(\mathbb{R}^+)^{|Y|}}{\mathrm{P}(Y)}$$

Program

$M, N ::= x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid \underline{n} \mid \text{coin} \mid \dots$
are interpreted by morphisms, by induction on M

- if $M : A$, then $[\![M]\!] \in \mathrm{P}(A)$

$$[\![\underline{n}]\!] = (0, \dots, \underset{n}{1}, 0, \dots) \qquad [\![\text{coin}]\!] = (\underset{0}{\tfrac{1}{2}}, \underset{1}{\tfrac{1}{2}}, 0, \dots)$$

- if $M : A \to B$, then $[\![M]\!] : \mathrm{P}(A) \to \mathrm{P}(B)$ is a Taylor series

Semantics: Probabilistic Coherent Spaces

(Danos-Ehrhard 2011)

Morphism

$\in (\mathbb{R}^+)^{\mathcal{M}_{\text{fin}}(|X|)\times|Y|}$ is a matrix

$$\forall x \in \underset{\subseteq(\mathbb{R}^+)^{|X|}}{\mathrm{P}(X)}\ ,\ M(x) = \left(\sum_{m\in\mathcal{M}_{\text{fin}}(|X|)} M_{m,b} \prod_{a\in m} x_a^{m(a)} \right)_{b\in|Y|} \in \underset{\subseteq(\mathbb{R}^+)^{|Y|}}{\mathrm{P}(Y)}$$

Program

$M, N ::= x \mid \lambda x^A.M \mid (M)N \mid \mathbf{fix}(M) \mid \underline{n} \mid \text{coin} \mid \dots$
are interpreted by morphisms, by induction on M

- if $M : A$, then $[\![M]\!] \in \mathrm{P}(A)$

$$[\![\underline{n}]\!] = (0, \dots, \underset{n}{1}, 0, \dots) \qquad [\![\text{coin}]\!] = (\underset{0}{\tfrac{1}{2}}, \underset{1}{\tfrac{1}{2}}, 0, \dots)$$

- if $M : A \to B$, then $[\![M]\!] : \mathrm{P}(A) \to \mathrm{P}(B)$ is a Taylor series
  if $M : 1 \to 1$, then $[\![M]\!]$ is smooth real function from $[0,1]$ to $[0,1]$
  if $M : \mathtt{nat} \multimap \mathtt{nat}$, then $[\![M]\!]$ is a sub-stochastic matrix

Probabilistic Coherent Spaces

Sound: Deterministic case: if $M \to N$, then $[\![M]\!] = [\![N]\!]$.

$$[\![M]\!] = \sum_N \mathbf{Proba}(M, N)[\![N]\!]$$

Adequate: If M close term of type $\mathtt{nat}$, then $[\![M]\!]_n = \mathbf{Proba}^{\infty}(M, \underline{n})$

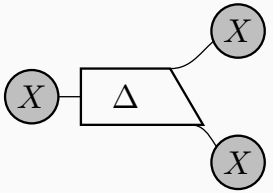
(Danos-Ehrhard 2011) $[\![M]\!]$ sub-proba distrib. on $\mathbb{N}$.

Fully abstract: $[\![M]\!] = [\![N]\!]$ iff $M \simeq N$

(Ehrhard-Pagani-T. POPL'14) Based on Taylor series

This Full Abstraction result generalizes to quantum programming.

(Clairambault-De Visme POPL'20)

CBPV: For every algebraic type X, the duplication 

is valid.

(Ehrhard-T. 2019)

Differential Programming

Semantics

Semantics of Differential Programming

Basic ingredient in **Pcoh** is that $[\![M]\!]$ is a Taylor series. This is actually the case in many quantitative semantics stemming from linear logic account of resource consumption. (Girard 1987)

Zoology of topological vector spaces that are semantics of HOPL:

Köthe spaces (Ehrhard 2002), Finiteness spaces (Ehrhard 2005), Convenient vector spaces (Blute-Ehrhard-T. 2012), Mackey-complete vector spaces (Kerjean-T. 2018).

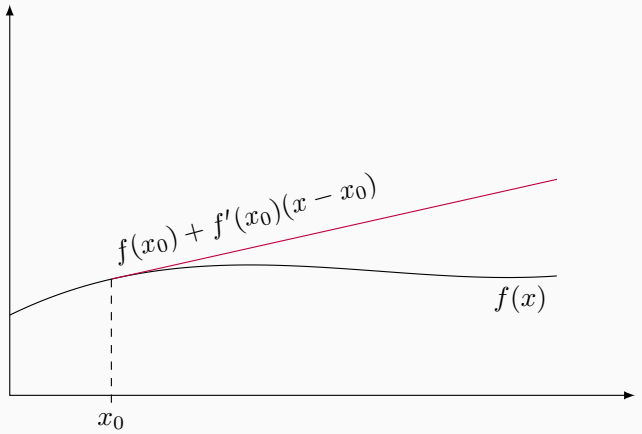
Ingredients

- Programs are smooth maps,
- Programs are Taylor series
- Derivative operator is a map in the model

From semantics to syntax

Differential lambda-calculus

(Ehrhard-Regnier 2003)



$D(\lambda x.M)N \to \lambda x.(\frac{\partial M}{\partial x} N)$

Linearization of the application.

Taylor expansion

(Ehrhard-Regnier 2006)

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n$$

λ-calculus	$\xrightarrow{\textbf{TE}}$	Resource calculus
$M\,N$		$< s > [t_1, \ldots, t_n]$

Application

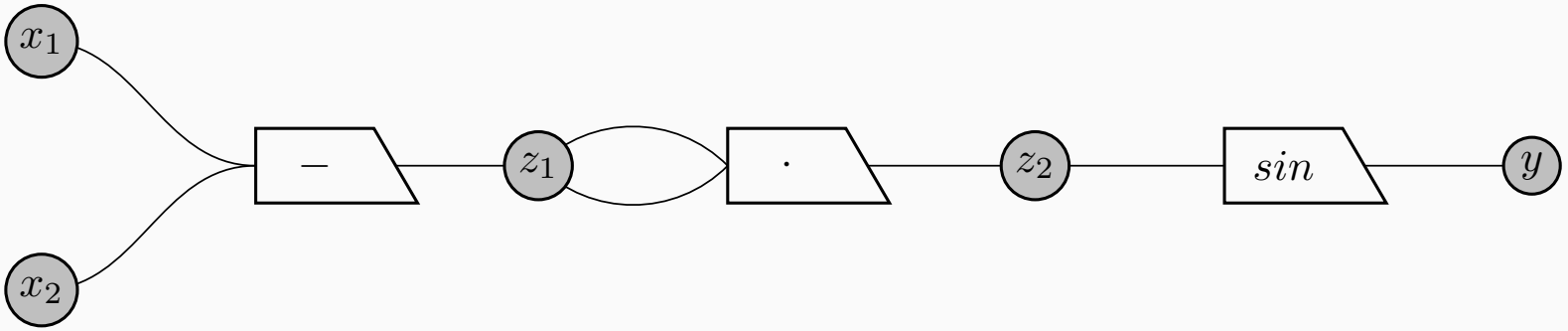
(Barbarossa-Manzonetto POPL'20)

A theory of approximation of programs based on resource consumption.

Differential Programming

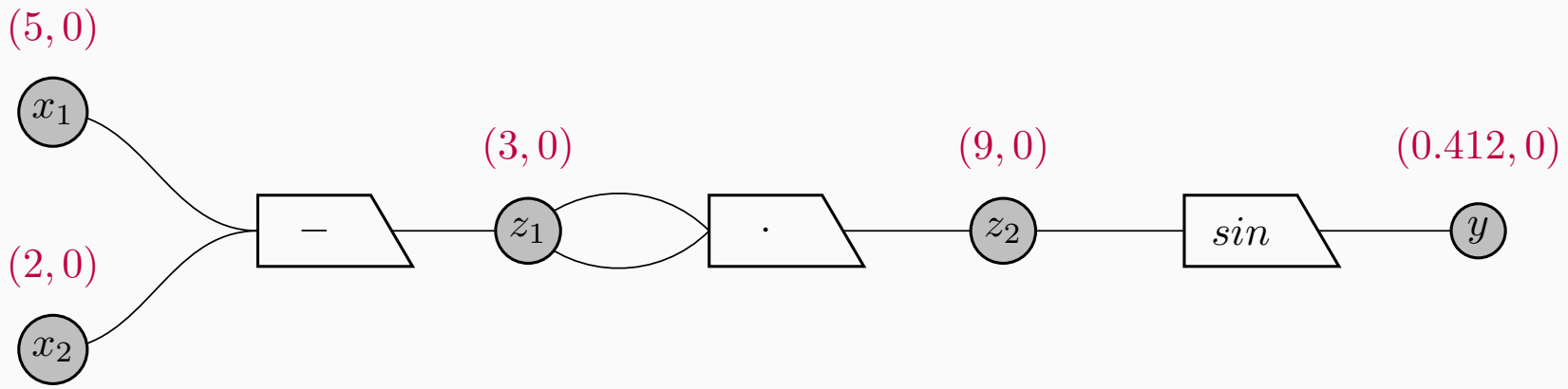
Automatic Differentiation

Automatic Differentiation on computational graphs



$[\![G]\!](x_1, x_2) = sin((x_1 - x_2)^2)$

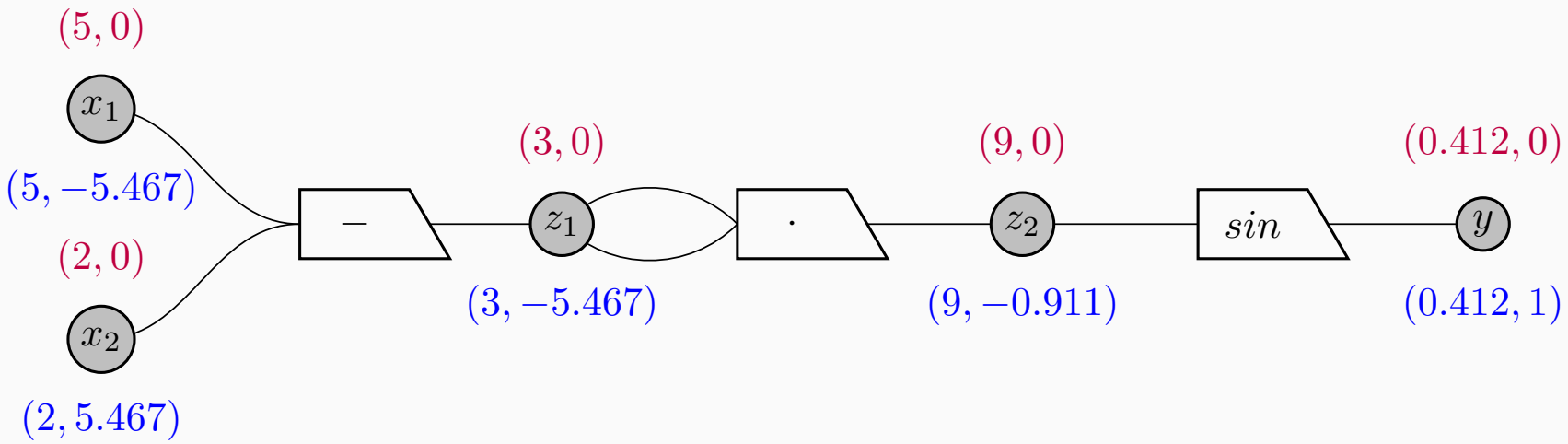
Automatic Differentiation on computational graphs



$[\![G]\!](x_1, x_2) = sin((x_1 - x_2)^2)$ $[\![G]\!](5, 2) = 0.412$

Forward propagation: compute $(s_i, 0)$ by evaluation and composition.

Automatic Differentiation on computational graphs



$[\![G]\!](x_1, x_2) = sin((x_1 - x_2)^2) \quad [\![G]\!](5, 2) = 0.412 \quad \nabla[\![G]\!] = (-5.467, 5.467)$

Forward propagation: compute $(s_i, 0)$ by evaluation and composition.

Backward propagation: compute (s_i, α_i) using chain rule

$$\frac{\partial f(v_1, \ldots, v_n)}{\partial x} = \sum_{i=1}^{n} \frac{\partial f}{\partial v_i} \cdot \frac{\partial v_i}{\partial x} \quad \text{and} \quad z_i = \left(s_i, \beta_i + \frac{\partial y}{\partial z_i} \cdot \alpha_i \right)$$

$z_2 = (9, 0 + cos(9) \cdot 1) \quad z_1 = (3, 0 + 3 \cdot -0.911) \quad z_1 = (3, 3 \cdot -0.911 + 3 \cdot -0.911)$

Compositional backpropagation (Brunel-Mazza-Pagani POPL'20)

Linear substitution calculus: (Accatoli 2012)

- HOPPL with explicit linear substitution
- well suited for fine grain complexity analysis
- no recursion or conditional

Linear negation of `real`: $\frac{\partial y}{\partial v}$ is a linear map from $\mathbb{R} \to \mathbb{R}$.

- delimited continuations (Wang et al ICFP'19)
- backpropagators (Pearlmutter-Siskind 2008)

Backpropagation

- program transformation on programs of type `real`
- correspond to usual algorithm on computational graphs

Backpropagation is sound, efficient and compositional.

Formalized Reverse derivative

(Abadi-Plotkin POPL'20)

Language: first order with fixpoints, conditions and reverse derivatives

$$A, B \quad ::= \quad \texttt{real} \mid 1 \mid A \times B$$

$$M, N, L \quad ::= \quad x \mid r \mid f(M) \mid \mathbf{fix}(f) \mid M \cdot \texttt{rd}_L(x.N) \mid \texttt{if } B \texttt{ then } M \texttt{ else } N \mid .$$

Traces with no application, fixpoints, conditions or reverse derivatives

Operational Semantics formalizing trace based differentiation

- Transformation of programs into trace programs
- Reverse derivative as an operator on traces
- Evaluation

Denotational Semantics: `if (x<0) then 0 else x`

- Types as ordered sets with properties from domain theory
- Programs as differentiable partial functions defined on open domains

Sound and adequate model

Differential Programming

Mixing with Probabilistic Programming

Derivative of probabilistic Programs

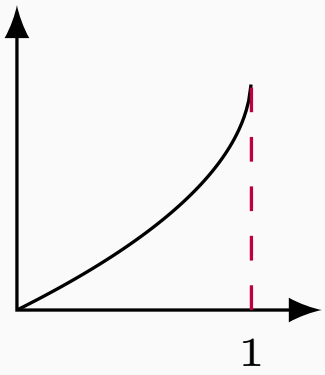
(Ehrhard FSCD'19)

Real functions are maps from 1 to 1 in **Pcoh**:

$$\mathtt{F} = \mathtt{fix\ f}^{1\to 1}\ \lambda \mathtt{x}^{1}\ \mathtt{if\ coin\ then\ ()\ else\ x; f(x)}$$

Taylor series $\phi = [\![F]\!] : [0,1] \to [0,1]$

$$\forall x \in [0,1],\ \phi(x) = \sum_{n=0}^{\infty} a_n\ x^n$$



$\phi(x) = \frac{1}{2} + \frac{1}{2}x\phi(\phi(x)) = 1 - \sqrt{1-x}.$

Analysis of compilation a_n the probability that $F()$ uses its argument exactly n times to produce an output.

Derivative $\phi'(1) = \sum_{n=0}^{\infty} n a_n$ is the expectation of the number of times F will use its argument for producing its input.

$F(1)$ converges almost surely with an ∞ expected computation time.

Conclusion

Summary

Formalizing compilers is a crucial challenge to avoid generating bugs.

Semantics allows to prove that program transformations at play in probabilistic and differential programming are correct.

Future works

- How to characterize inference approximations.
- Use the semantics tools for certification and proofs.

Related works

- Semantics of derivation potentially mixed with probability.
- Probabilistic distributed computing.

PIHOC-PPS-DIAPASoN workshop **Paris, Feb 26 to 28, 2020**

Program semantics and formal methods for probabilistic programming, statistical learning, differential and approximate computing.

(organized by Dal Lago-Ehrhard-Pagani)