Semantics of Reactive Probabilistic Programming

JFLA 2024

Guillaume Baudart <guillaume.baudart@inria.fr>
Louis Mandel <louis.mandel@us.ibm.com>
Christine Tasson <christine.tasson@isae-supaero.fr>

isae SUPAERO

Introduction

Probabilistic semantics

Probabilistic Progamming

A probabilistic program is a random variable: it can be sampled and models a probability distribution, like a dice or gaussian.

Las Vegas algorithm

Given $\mathrm{t} : \mathrm{int} \to \mathrm{int}$ and $\mathrm{n} : \mathrm{int}$, find i such that $1 \leq i \leq n$ and $t(i) = 0$

```
let rec f () =
  let x = sample(uniform_int(1, n)) in
  if t(x) = 0 then x else f()
```

Assuming $\exists i_0,\ 1 \leq i_0 \leq n$ and $t(i_0) = 0$,

- does it terminates ?
- what probabilistic model does it represents ?

Probabilistic Progamming

A probabilistic program is a random variable: it can be sampled and models a probability distribution, like a dice or gaussian.

Las Vegas algorithm

Given $\mathrm{t} : \mathrm{int} \to \mathrm{int}$ and $\mathrm{n} : \mathrm{int}$, find i such that $1 \leq i \leq n$ and $t(i) = 0$

$$
\begin{array}{l}
\mathrm{let\ rec\ f}\ () = \\
\quad \mathrm{let\ x} = \ \mathrm{sample}(\mathrm{uniform\_int}(1, \mathrm{n}))\ \mathrm{in} \\
\quad \mathrm{if\ t(x)} = 0\ \mathrm{then\ x\ else\ f}()
\end{array}
$$

Assuming $\exists i_0,\ 1 \leq i_0 \leq n \quad \text{and} \quad t(i_0) = 0$,

- does it terminates ? yes ! **almost surely**.
- what probabilistic model does it represents ? it depends on the **evaluation strategy:**
  - Call-by-name: $\mathrm{uniform\_int}(1, \mathrm{n})$
  - Call-by-value: $\mathrm{uniform}\{i \mid t(i) = 0\}$

Probabilistic Semantics might be tricky

Types: Measurable space: carrier X and a family of measurables Σ_X

Programs: if $\Gamma \vdash e : X$ then $\forall \gamma$, $[\![e]\!]_\gamma : \Sigma_X \to \mathbb{R}^+$ is a measure on X,
i.e. $[\![e]\!] : \Gamma \times \Sigma_X \to \mathbb{R}^+$ is a kernel.

Higher-Order?

Probabilistic Semantics might be tricky

Types: Measurable space: carrier X and a family of measurables Σ_X

Programs: if $\Gamma \vdash e : X$ then $\forall \gamma$, $[\![e]\!]_\gamma : \Sigma_X \to \mathbb{R}^+$ is a measure on X,
i.e. $[\![e]\!] : \Gamma \times \Sigma_X \to \mathbb{R}^+$ is a kernel.

Higher-Order? The category of measurable space and kernels is monoidal but not closed 😰, as there is no measurable structure on function space $\mathbb{R} \to \mathbb{R}$ such that evaluation is measurable.

📚 *Aumann. Borel structures for function spaces. 1961*

Probabilistic Semantics might be tricky

Types: Measurable space: carrier X and a family of measurables Σ_X

Programs: if $\Gamma \vdash e : X$ then $\forall \gamma$, $[\![e]\!]_\gamma : \Sigma_X \to \mathbb{R}^+$ is a measure on X,
i.e. $[\![e]\!] : \Gamma \times \Sigma_X \to \mathbb{R}^+$ is a kernel.

Higher-Order? The category of measurable space and kernels is monoidal but not closed 😰, as there is no measurable structure on function space $\mathbb{R} \to \mathbb{R}$ such that evaluation is measurable.

Aumann. Borel structures for function spaces. 1961

Yet other models can be defined, where the function type is not interpreted as measures on functions but as a **distribution transformer**, like stochastic matrix or bayesian networks.

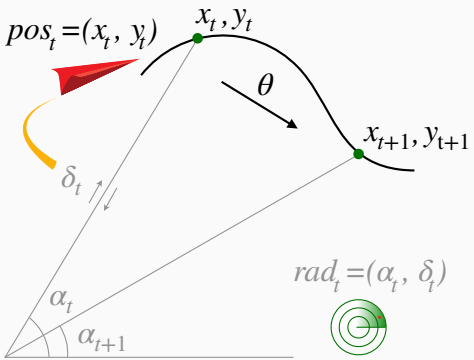
Heunen & al. A convenient category for Higher-Order Probability Theory. 2017

Ehrhard & al. Measurable cones and Stable, Measurable Functions. 2018

Introduction

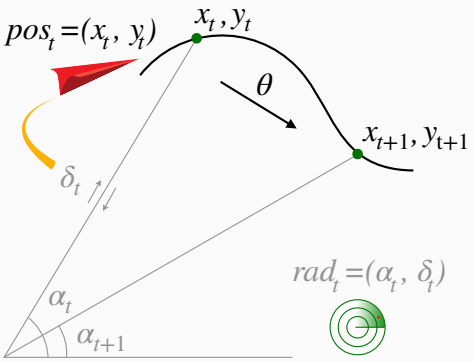
Semantics of Synchronous Probabilistic Programming

Tracker example



Chopin & Papaspiliopoulos. An introduction to sequential Monte Carlo. 2020

Tracker example

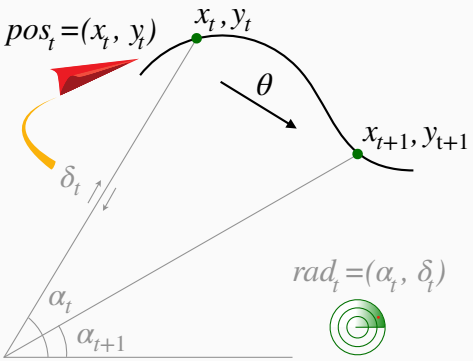


Linear Movement model

$$z = 10\,000\ m$$

$$\text{pos}_{t+1} \sim \mathcal{N}(\text{pos}_t + \theta, s_p)$$

Tracker example



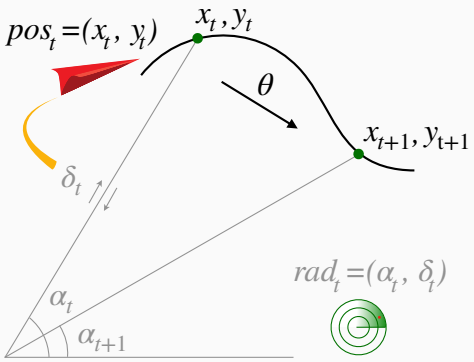
Linear Movement model

$$
\begin{aligned}
z &= 10\,000\ m \\
\text{pos}_{t+1} &\sim \mathcal{N}(\text{pos}_t + \theta, s_p)
\end{aligned}
$$

Radar: angle and delay

$$
\begin{aligned}
\text{rad}_t &= g(\text{pos}_t) \\
\alpha_t &= \text{atan}(y_t / x_t) \\
\delta_t &= 2\sqrt{x_t^2 + y_t^2} / c_{\text{light}}
\end{aligned}
$$

Tracker example



Linear Movement model

$$
\begin{aligned}
z &= 10\,000\ m \\
\text{pos}_{t+1} &\sim \mathcal{N}(\text{pos}_t + \theta, s_p)
\end{aligned}
$$

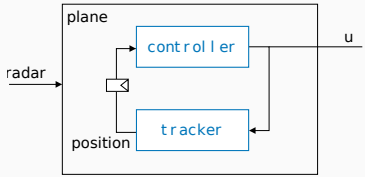
Radar: angle and delay

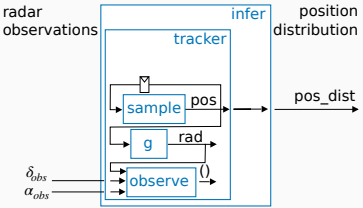
$$
\begin{aligned}
\text{rad}_t &= g(\text{pos}_t) \\
\alpha_t &= \text{atan}(y_t / x_t) \\
\delta_t &= 2\sqrt{x_t^2 + y_t^2} / c_{\text{light}}
\end{aligned}
$$

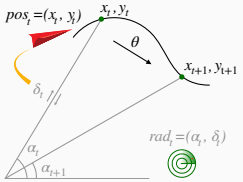
Noisy observation

$$
\text{rad\_obs}_t \sim \mathcal{N}(\text{rad}_t, s_r)
$$

Programming the tracker








```
1  proba tracker(rad_obs) = pos where
2    rec  init pos = pos_init
3    and pos = sample(gaussian(f(last pos), s_p))
4    and rad = g(pos)
5    and () = observe(gaussian(rad, s_r), rad_obs)
6
7  node main(rad_obs) = u where
8    rec pos_dist = infer (tracker (rad_obs))
9    and u = controller(pos_dist)
```

Notice that equations correspond to arrows in block diagrams and are unordered.

Deterministic Synchronous Semantics

Coiterative Semantics

Deterministic co-iterative semantics (under causality assumption)

State Machine

Allocation: $[\![e]\!]_\gamma^{\text{init}} : M$

Transition: $[\![e]\!]_\gamma^{\text{step}} : M \to M \times V$

- State: M
- Input Environment: $\gamma \in \Gamma$
- Output Values: V

$$\left[\!\!\left[\begin{array}{l}\texttt{e where rec init x = c}\\ \qquad\texttt{and x = e\_x}\\ \qquad\texttt{and y = e\_y}\end{array}\right]\!\!\right]_\gamma^{\text{init}} = c, \left([\![e]\!]_\gamma^{\text{init}}, [\![e_x]\!]_\gamma^{\text{init}}, [\![e_y]\!]_\gamma^{\text{init}}\right)$$

$$\left[\!\!\left[\begin{array}{l}\texttt{e where rec init x = c}\\ \qquad\texttt{and x = e\_x}\\ \qquad\texttt{and y = e\_y}\end{array}\right]\!\!\right]_\gamma^{\text{step}} (p_x, (m, m_x, m_y)) = \text{let } m'_x, v_x = [\![e_x]\!]^{\text{step}}_{\gamma+[x.last \leftarrow p_x]}(m_x) \text{ in}$$

$$\text{let } m'_y, v_y = [\![e_y]\!]^{\text{step}}_{\gamma+[x.last \leftarrow p_x, x \leftarrow v_x]}(m_y) \text{ in}$$

$$\text{let } m', v = [\![e]\!]^{\text{step}}_{\gamma+[x.last \leftarrow p_x, x \leftarrow v_x, y \leftarrow v_y]}(m) \text{ in}$$

$$(v_x, (m', m'_x, m'_y)),\ v$$

Mutually recursive equations

Streams defined by causal equations

```
(x, y) where
    rec x = z+1
    and y = x
    and z = 42
```

In one time step

```
let rec x = z+1
  and y = x
  and z = 42
  in (x, y)
```

Scheduled agnostic semantics

- Inherited from block diagrams
- Standard in industry:
  Simulink, Scade

Fixpoint computation

$$\rho_0 = [x \leftarrow \bot,\ y \leftarrow \bot, z \leftarrow \bot]$$
$$\rho_1 = [x \leftarrow \bot,\ y \leftarrow \bot, z \leftarrow 42]$$
$$\rho_2 = [x \leftarrow 43,\ y \leftarrow \bot, z \leftarrow 42]$$
$$\rho_3 = [x \leftarrow 43,\ y \leftarrow 43, z \leftarrow 42]$$
$$\rho_4 = [x \leftarrow 43,\ y \leftarrow 43, z \leftarrow 42]$$

Colaço & al. A Constructive State-based Semantics and Interpreter for a Synchronous Data-flow Language with State machines. 2023.

Deterministic Synchronous Semantics

Relational Semantics

Deterministic Relational Semantics

Sequents

Evaluates expressions to streams: $G, H \vdash e \downarrow s$

Checks stream equations: $G, H \vdash E$

- G globals
- H maps all variables to streams

$$\emptyset, \begin{bmatrix} x & = & 1 \cdot 2 \cdot 3 \cdot \ldots \\ y & = & 2 \cdot 4 \cdot 6 \cdot \ldots \end{bmatrix} \vdash \begin{array}{l} \texttt{rec x = 1} \rightarrow \texttt{last x + 1} \\ \texttt{and y = 2 * x} \end{array}$$

$$\vdash \begin{array}{l} \texttt{(x, y) where rec init x = 0} \\ \qquad\qquad \texttt{and x = last x + 1} \downarrow (1,2) \cdot (2,4) \cdot (3,6) \cdot \ldots \\ \qquad\qquad \texttt{and y = 2 * x} \end{array}$$

Bourke & al. A formally verified compiler for Lustre. 2017.

Deterministic Relational Semantics

$$G, H \vdash c \downarrow c \qquad G, H \vdash x \downarrow H(x) \qquad \frac{x \notin H}{G, H \vdash x \downarrow G(x)} \qquad \frac{G, H \vdash e_1 \downarrow s_1 \qquad G, H \vdash e_2 \downarrow s_2}{G, H \vdash (e_1, e_2) \downarrow (s_1, s_2)}$$

$$\frac{G, H \vdash e \downarrow s}{G, H \vdash op(e) \downarrow op(s)} \qquad \frac{H(x.\mathrm{last}) = s}{G, H \vdash \texttt{last}\ x \downarrow s}$$

$$\frac{G, H \vdash e \downarrow s_e \qquad G(f) = \texttt{node}\ f\ x = e_f \qquad G, [x \leftarrow s_e] \vdash e_f \downarrow s}{G, H \vdash f(e) \downarrow s}$$

$$\frac{G, H + H_E \vdash E \qquad G, H + H_E \vdash e \downarrow s}{G, H \vdash e\ \texttt{where rec}\ E \downarrow s}$$

$$\frac{G, H \vdash e \downarrow H(x)}{G, H \vdash x = e} \qquad \frac{G, H \vdash e \downarrow i \cdot s \qquad H(x.\mathrm{last}) = i \cdot H(x)}{G, H \vdash \texttt{init}\ x = e} \qquad \frac{G, H \vdash E_1 \qquad G, H \vdash E_2}{G, H \vdash E_1\ \texttt{and}\ E_2}$$

Deterministic Synchronous Semantics

Equivalent semantics

Equivalence between Deterministic Coiterative and Relational Semantics

Theorem [Bourke & al. 2017]: For causal models,

For any H and γ such that $\forall k,\ \forall x \in \mathrm{FV}(e),\ \gamma_k(x) = H(x)_k$,

$$\text{if } G, H \vdash e \downarrow s, \text{ then, } \forall k,\ s_k(H) = v_k(H_{\leq k}) \quad \text{where} \quad \begin{cases} m_0 & = [\![e]\!]^{\mathrm{init}}_{\gamma_0} \\ m_k, v_k & = [\![e]\!]^{\mathrm{step}}_{\gamma_k}(m_{k-1}) \end{cases}$$

Equivalence between Deterministic Coiterative and Relational Semantics

Theorem [Bourke & al. 2017]: For causal models,

For any H and γ such that $\forall k,\ \forall x \in \mathrm{FV}(e),\ \gamma_k(x) = H(x)_k$,

$$\text{if } G, H \vdash e \downarrow s, \text{ then, } \forall k,\ s_k(H) = v_k(H_{\leq k}) \quad \text{where} \quad \begin{cases} m_0 & = [\![e]\!]^{\mathrm{init}}_{\gamma_0} \\ m_k, v_k & = [\![e]\!]^{\mathrm{step}}_{\gamma_k}(m_{k-1}) \end{cases}$$

```
(x, y) where rec init x = 0
           and x = last x + 1
           and y = 2 * x
```

$$\vdash \quad \downarrow (1,2) \cdot (2,4) \cdot (3,6) \cdot \ldots$$

$$\left[\!\!\left[\begin{array}{l} \texttt{(x, y) where rec init x = 0} \\ \texttt{\quad\quad and x = last x + 1} \\ \texttt{\quad\quad and y = 2 * x} \end{array} \right]\!\!\right]^{\mathrm{init}}_{\gamma_0} = \overbrace{0, (((), ()), (), ())}^{m_0}$$

$$\left[\!\!\left[\begin{array}{l} \texttt{(x, y) where rec init x = 0} \\ \texttt{\quad\quad and x = last x + 1} \\ \texttt{\quad\quad and y = 2 * x} \end{array} \right]\!\!\right]^{\mathrm{step}}_{\gamma_k} \underbrace{(k-1, (((), ()), (), ()))}_{m_{k-1}} = \underbrace{k, (((), ()), (), ())}_{m_k}, \underbrace{(k,\ 2 * k)}_{v_k}$$

Deterministic Synchronous Semantics

Program Equivalence

Program Equivalence

Definition: $e^1 \cong e^2$ if for all **input** streams, e^1 and e^2 produce the same **output** streams.

Thanks to the equivalence of semantics, the equivalence in the **relational** semantics coincides with the bisimulation of state machines from the **coiterative** semantics.

Program Equivalence

Definition: $e^1 \cong e^2$ if for all **input** streams, e^1 and e^2 produce the same **output** streams.

Thanks to the equivalence of semantics, the equivalence in the **relational** semantics coincides with the bisimulation of state machines from the **coiterative** semantics.

Relational equivalence:

$$\forall H,\ s^1(H) = s^2(H) \text{ where } G, H \vdash e^i \downarrow s^i.$$

Program Equivalence

Definition: $e^1 \cong e^2$ if for all **input** streams, e^1 and e^2 produce the same **output** streams.

Thanks to the equivalence of semantics, the equivalence in the **relational** semantics coincides with the bisimulation of state machines from the **coiterative** semantics.

Relational equivalence:

$$\forall H,\ s^1(H) = s^2(H) \text{ where } G, H \vdash e^i \downarrow s^i.$$

Coiterative equivalence:

There is a bisimulation $\cong$, i.e. a relation on states such that:

$$\forall k\ \forall \gamma_k\ m_0^1 \cong m_0^2 \qquad \text{where } m_0^i = [\![e^i]\!]_{\gamma_0}^{\text{init}}$$

$$m_{k+1}^1 \cong m_{k+1}^2 \quad \text{and} \quad v_{k+1}^1 = v_{k+1}^2 \qquad \text{where } m_{k+1}^i, v_{k+1}^i = [\![e^i]\!]_{\gamma_{k+1}}^{\text{step}}(m_k^i)$$

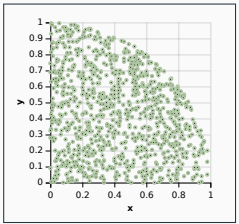
Probabilistic Synchronous Semantics

Probabilistic Programming

Sample mean and law of large numbers

Sample mean: The sample mean of a random variable is obtained by **simulation**:

- compute n **samples** denoted $x_1, \ldots, x_n$.
- compute the mean $\frac{1}{n}(x_1 + \cdots + x_n)$.



Sample mean and law of large numbers

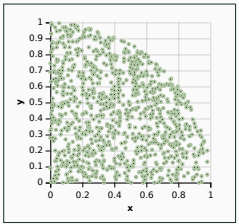
Sample mean: The sample mean of a random variable is obtained by **simulation**:

- compute n **samples** denoted $x_1, \ldots, x_n$.
- compute the mean $\frac{1}{n}(x_1 + \cdots + x_n)$.

Law of large numbers: **The sample mean approximates the mean.**

If $X_1, \ldots, X_n$ are independent identically distributed (i.i.d.) and the mean of $g(X)$ is finite, then

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \to \mathbb{E}(g(X))$$



Sample mean and law of large numbers

Sample mean: The sample mean of a random variable is obtained by **simulation**:

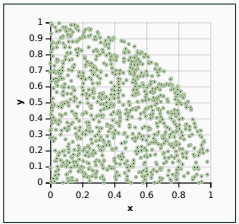
- compute n **samples** denoted $x_1, \ldots, x_n$.
- compute the mean $\frac{1}{n}(x_1 + \cdots + x_n)$.

Law of large numbers: **The sample mean approximates the mean.**

If $X_1, \ldots, X_n$ are independent identically distributed (i.i.d.) and the mean of $g(X)$ is finite, then

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \to \mathbb{E}(g(X))$$

$$\frac{1}{n}\sum_{i=1}^{n} X_i \to \mathbb{E}(X)$$



Monte-Carlo simulation and the law of large numbers

A **Probabilistic program** is the **random variable** whose values are the outcome of the program execution. If the program has no **effect**, then its **executions** are i.i.d. Then:

```
let pi4() =
    let x = sample(uniform(0., 1.)) in
    let y = sample(uniform(0., 1.)) in
    let () = assume(distance(x,y) < 1.) in
    1.
```

Monte-Carlo simulation and the law of large numbers

A **Probabilistic program** is the **random variable** whose values are the outcome of the program execution. If the program has no **effect**, then its **executions** are i.i.d. Then:

Law of large numbers:

- Run n times the probabilistic program
- Store the outputs $x_1, \ldots, x_n$.
- Compute $\frac{x_1 + \cdots + x_n}{n} \to_\infty \mathbb{E}(X) = \int \mu_X(dx)$ and $\frac{g(x_1) + \cdots + g(x_n)}{n} \to_\infty \mathbb{E}(g(x)) = \int g(x)\mu_X(dx)$

```
let pi4() =
    let x = sample(uniform(0., 1.)) in
    let y = sample(uniform(0., 1.)) in
    let () = assume(distance(x,y) < 1.) in
    1.
```

Monte-Carlo simulation and the law of large numbers

A **Probabilistic program** is the **random variable** whose values are the outcome of the program execution. If the program has no **effect**, then its **executions** are i.i.d. Then:

Law of large numbers:

- Run n times the probabilistic program
- Store the outputs $x_1, \ldots, x_n$.
- Compute $\frac{x_1 + \cdots + x_n}{n} \to_\infty \mathbb{E}(X) = \int \mu_X(dx)$ and $\frac{g(x_1) + \cdots + g(x_n)}{n} \to_\infty \mathbb{E}(g(x)) = \int g(x)\mu_X(dx)$

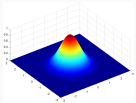
Monte-Carlo Simulation: histograms approximate distributions

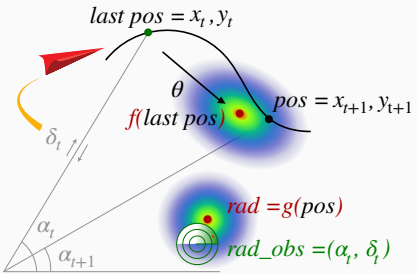
- $\frac{1}{n}\#\{i | x_i = x\}$ approximates $\mathbb{E}(\mathbb{1}_{X=x}) = \mathbb{P}(X = x)$
- $\frac{1}{n}\#\{i | a \leq x_i \leq b\}$ approximates $\mathbb{E}(\mathbb{1}_{a \leq X \leq b}) = \mathbb{P}(a \leq X \leq b)$

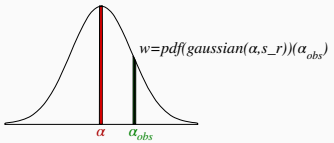
```
let pi4() =
    let x = sample(uniform(0., 1.)) in
    let y = sample(uniform(0., 1.)) in
    let () = assume(distance(x,y) < 1.) in
    1.
```

ProbZelus

```
1  proba tracker(rad_obs) = pos where
2    rec  init pos = pos_init
3    and pos = sample(gaussian(f(last pos), s_p))
4    and rad = g(pos)
5    and () = factor(pdf(gaussian(rad, s_r))(rad_obs))
6    (* () = observe(gaussian(rad, s_r), rad_obs) *)
7
8  node main(rad_obs) = u where
9    rec pos_dist = infer (tracker (rad_obs))
10   and u = controller(pos_dist)
```





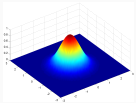


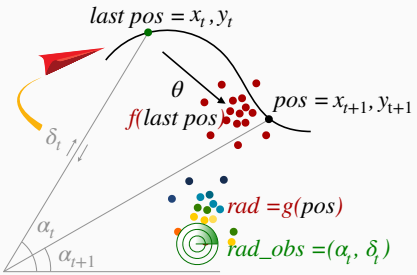
```
1  proba tracker(rad_obs) = pos where
2    rec  init pos = pos_init
3    and pos = sample(gaussian(f(last pos), s_p))
4    and rad = g(pos)
5    and () = factor(pdf(gaussian(rad, s_r))(rad_obs))
6    (* () = observe(gaussian(rad, s_r), rad_obs) *)
7
8  node main(rad_obs) = u where
9    rec pos_dist = infer (tracker (rad_obs))
10   and u = controller(pos_dist)
```

sample: ●●●●●●● $[(pos^0,\ 1\ ), \ldots, (pos^n,\ 1\ )]$

observe: ●●●●●●● $[(pos^0, w^0), \ldots, (pos^n, w^n)]$

categorical distribution





Probabilistic Synchronous Semantics

Probabilistic Semantics

Co-iterative kernel semantics

State Machine

Allocation: $\{\!|\, e \,|\!\}_\gamma^{\text{init}} : M$

Transition: $\{\!|\, e \,|\!\}_\gamma^{\text{step}} : M \to \Sigma_{M \times V} \to \mathbb{R}^+$

- State: M
- Input Environment: $\gamma \in \Gamma$
- Output Values: V

$$[\![\texttt{infer}(e)]\!]_\gamma^{\text{step}} : M \text{ dist} \to M \text{ dist} \times V \text{ dist}$$

$$[\![\texttt{infer}(e)]\!]_\gamma^{\text{init}} = [\![e]\!]_\gamma^{\text{init}}$$

$$[\![\texttt{infer}(e)]\!]_\gamma^{\text{step}}(\sigma) = \textit{let } \nu = \int \sigma(dm)\, \{\!|\, e \,|\!\}_\gamma^{\text{step}}(m) \textit{ in let } \overline{\nu} = \nu/\nu(\top) \textit{ in } (\pi_{1*}(\overline{\nu}), \pi_{2*}(\overline{\nu}))$$

Baudart & al. Reactive Probabilistic Programming. 2020

Co-iterative kernel semantics

GoTo Deterministic

State Machine

Allocation: $\{\!|\, e \,|\!\}_\gamma^{\text{init}} : M$

Transition: $\{\!|\, e \,|\!\}_\gamma^{\text{step}} : M \to \Sigma_{M \times V} \to \mathbb{R}^+$

- State: M
- Input Environment: $\gamma \in \Gamma$
- Output Values: V

$$\left\{\!\!\left[\begin{array}{l}\texttt{e where rec init x = c}\\ \qquad\texttt{and x = e\_x}\\ \qquad\texttt{and y = e\_y}\end{array}\right]\!\!\right\}_\gamma^{\text{step}} (p_x, (m, m_x, m_y)) = \int \{\!|\, e_x \,|\!\}_{\gamma+[x.last \leftarrow p_x]}^{\text{step}} (m_x)\ (dm'_x, dv_x)$$

$$\int \{\!|\, e_y \,|\!\}_{\gamma+[x.last \leftarrow p_x, x \leftarrow v_x]}^{\text{step}} (m_y)\ (dm'_y, dv_y)$$

$$\int \{\!|\, e \,|\!\}_{\gamma+[x.last \leftarrow p_x, x \leftarrow v_x, y \leftarrow v_y]}^{\text{step}} (m)\ (dm', dv)$$

$$\delta_{(v_x, (m', m'_x, m'_y))}$$

Baudart & al. Reactive Probabilistic Programming. 2020

Probabilistic mutually recursive equations

GoTo Deterministic

Streams defined by causal equations

```
(x, y) where
    rec x = sample(gaussian(42, 1))
    and y = x
```

In one time step

```
let rec x = sample(gaussian(42, 1))
  and y = x
  in (x, y)
```

Scheduled semantics:

$\rho_{x,y} = \int \delta_{x,x} * \mathcal{N}(42, 1)(dx)$

Scheduled agnostic semantics

- Inherited from block diagrams
- Standard in industry: Simulink, Scade

Fixpoint and order on measures

- Fixpoint of $\mu \mapsto$ $\int \int (\mathcal{N}(42, 1) \otimes \delta_x)(dx', dy') * \mu(dx, dy) * \delta_{x',y'}$
- Kleene:
  - $\perp$ is the null measure
  - $\mu \leq \nu$ iff for all U, $\mu(U) \leq \nu(U)$

Jones & Plotkin. A Probabilistic Powerdomain of Evaluations. 1998

Probabilistic mutually recursive equations

GoTo Deterministic

Streams defined by causal equations

```
(x, y) where
    rec x = sample(gaussian(42, 1))
    and y = x
```

In one time step

```
let rec x = sample(gaussian(42, 1))
  and y = x
  in (x, y)
```

Scheduled semantics:

$\rho_{x,y} = \int \delta_{x,x} * \mathcal{N}(42, 1)(dx)$

Scheduled agnostic semantics

- Inherited from block diagrams
- Standard in industry: Simulink, Scade

Fixpoint and order on measures

- Fixpoint of $\mu \mapsto$ $\int \int (\mathcal{N}(42, 1) \otimes \delta_x)(dx', dy') * \mu(dx, dy) * \delta_{x',y'}$
- Kleene: 😢
  - $\bot$ is the null measure
  - $\mu \leq \nu$ iff for all U, $\mu(U) \leq \nu(U)$

The null measure is the least fixpoint.

Jones & Plotkin. A Probabilistic Powerdomain of Evaluations. 1998

From kernel to density semantics

A **measure** $\mu : \Sigma \to \mathbb{R}^+$ with **density** $\mathrm{pdf}_\mu : [0,1] \to \mathbb{R}^+$ can be expressed with respect to the Lebesgues measure: (change of variable formula)

$$\mu = \int \delta_x * \mathrm{pdf}_\mu(dx) = \int_{[0,1]} \delta_{icdf_\mu(r)} * \mathrm{pdf}_\mu(icdf_\mu(dr)) = \int_{[0,1]} \delta_{icdf_\mu(r)} * dr$$

Kernel Semantics $\{\!|e|\!\}_\gamma : \Sigma_V \to \mathbb{R}^+$ and **Density Semantics** $(\!|e|\!)_\gamma : [0,1]^p \to V \times \mathbb{R}^+$

$$(\!|e|\!)_\gamma(r) = (v(r), w(r)) \text{ with } \begin{cases} p \text{ number of samples} \\ r \in [0,1]^p \text{ sample seeds} \\ v(r) \text{ output value} \\ w(r) \text{ weight} \end{cases}$$

are related by **Lebesgues measure** $\lambda_{[0,1]^p}$

$$\forall \gamma \in \Gamma,\ \{\!|e|\!\}_\gamma = \int_{[0,1]^p} \delta_{v(r)} * w(r)\, dr$$

Probabilistic Synchronous Semantics

Co-iterative Density Semantics

Co-iterative density semantics

GoTo Kernel

State Machine

Allocation: $(\!|e|\!)_\gamma^{\text{init}} : M \times \mathbb{N}$

Transition: $(\!|e|\!)_\gamma^{\text{step}} : M \times [0,1]^p \to M \times V \times \mathbb{R}^+$

- State: M Environment: $\gamma \in \Gamma$
- Output Values: V
- Number of random sites p

Given random seeds, computation is deterministic.

$$
\begin{array}{lcl}
(\!|e|\!)_\gamma^{\text{init}} & = & [\![e]\!]_\gamma^{\text{init}}, 0 \\
(\!|e|\!)_\gamma^{\text{step}}(m, []) & = & \textit{let } m', v = [\![e]\!]_\gamma^{\text{step}}(m) \textit{ in } m', v, 1 \qquad \text{if } e \text{ is deterministic}
\end{array}
$$

$$
\begin{array}{lcl}
(\!|\texttt{sample}(e)|\!)_\gamma^{\text{init}} & = & \textit{let } m = [\![e]\!]_\gamma^{\text{init}} \textit{ in } m, 1 \\
(\!|\texttt{sample}(e)|\!)_\gamma^{\text{step}}(m, [r]) & = & \textit{let } m', \mu = [\![e]\!]_\gamma^{\text{step}}(m) \textit{ in } m', \textit{icdf}_\mu(r), 1
\end{array}
$$

$$
\begin{array}{lcl}
(\!|\texttt{factor}(e)|\!)_\gamma^{\text{init}} & = & \textit{let } m = [\![e]\!]_\gamma^{\text{init}} \textit{ in } m, 0 \\
(\!|\texttt{factor}(e)|\!)_\gamma^{\text{step}}(m, []) & = & \textit{let } m', v = [\![e]\!]_\gamma^{\text{step}}(m) \textit{ in } m', (), v
\end{array}
$$

Co-iterative density semantics

GoTo Kernel

State Machine

Allocation: $(\!|e|\!)_\gamma^{\text{init}} : M \times \mathbb{N}$

Transition: $(\!|e|\!)_\gamma^{\text{step}} : M \times [0,1]^p \to M \times V \times \mathbb{R}^+$

- State: M Environment: $\gamma \in \Gamma$
- Output Values: V
- Number of random sites p

Given random seeds, computation is deterministic.

$$[\![\texttt{infer}(e)]\!]_\gamma^{\mathrm{D}\,\text{init}} = \textit{let } m, p = (\!|e|\!)_\gamma^{\text{init}} \textit{ in } \delta_m, p$$

$$[\![\texttt{infer}(e)]\!]_\gamma^{\mathrm{D}\,\text{step}}(\sigma, p) = \textit{let } \psi(m) = \int_{[0,1]^p} \textit{let } m', v, w = (\!|e|\!)_\gamma^{\text{step}}(m, r) \textit{ in } w * \delta_{(m',v)} \, dr \textit{ in}$$
$$\textit{let } \nu = \int \sigma(dm)\ \psi(m) \textit{ in let } \overline{\nu} = \nu/\nu(\top) \textit{ in}$$
$$(\pi_{1*}(\overline{\nu}), p), \pi_{2*}(\overline{\nu})$$

Co-iterative density semantics

GoTo Kernel

State Machine

Allocation: $(\!|e|\!)_\gamma^{\text{init}} : M \times \mathbb{N}$

Transition: $(\!|e|\!)_\gamma^{\text{step}} : M \times [0,1]^p \to M \times V \times \mathbb{R}^+$

- State: M Environment: $\gamma \in \Gamma$
- Output Values: V
- Number of random sites p

One big integral on random seeds instead of nested integrals at each synchronous step.

$$\left(\!\left|\begin{array}{l}\texttt{e where rec init x = c}\\ \qquad\texttt{and x = e\_x}\\ \qquad\texttt{and y = e\_y}\end{array}\right|\!\right)_\gamma^{\text{step}} (M), [r, r_x, r_y]) =$$

$$\begin{array}{l}\text{let } m'_x, v_x, w_x = (\!|e_x|\!)_\gamma^{\text{step}} (m_x, r_x)\\ \text{and } m'_y, v_y, w_y = (\!|e_y|\!)_\gamma^{\text{step}} (m_y, r_y)\\ \text{and } m', v, w \ = (\!|e|\!)_\gamma^{\text{step}} (m, r) \text{ in}\end{array}$$

$$(v_x, (m', m'_x, m'_y)), v, w \cdot w_x \cdot w_y$$

Co-iterative density semantics

GoTo Kernel

State Machine

Allocation: $(\!|e|\!)_\gamma^{\text{init}} : M \times \mathbb{N}$

Transition: $(\!|e|\!)_\gamma^{\text{step}} : M \times [0,1]^p \to M \times V \times \mathbb{R}^+$

- State: M Environment: $\gamma \in \Gamma$
- Output Values: V
- Number of random sites p

One big integral on random seeds instead of nested integrals at each synchronous step.

$$\int_{[0,1]^3} \text{let } M', V', W' = \left(\begin{array}{l}\texttt{e where rec init x = c}\\ \texttt{\quad and x = e\_x}\\ \texttt{\quad and y = e\_y}\end{array}\right)_\gamma^{\text{step}} (M), [r, r_x, r_y]) \text{ in } \delta_{M'} \otimes \delta_{V'} * W' \, dr dr_x dr_y =$$

$$\int_{[0,1]^3} \begin{array}{l}\text{let } m'_x, v_x, w_x = (\!|e_x|\!)_\gamma^{\text{step}} (m_x, r_x)\\ \text{and } m'_y, v_y, w_y = (\!|e_y|\!)_\gamma^{\text{step}} (m_y, r_y)\\ \text{and } m', v, w \ = (\!|e|\!)_\gamma^{\text{step}} (m, r) \text{ in}\end{array}$$

$$(\delta_{(v_x,(m',m'_x,m'_y))} \otimes \delta_v) * w \cdot w_x \cdot w_y \, dr dr_x dr_y$$

Co-iterative density semantics

GoTo Kernel

State Machine

Allocation: $(\!|e|\!)_\gamma^{\text{init}} : M \times \mathbb{N}$

Transition: $(\!|e|\!)_\gamma^{\text{step}} : M \times [0,1]^p \to M \times V \times \mathbb{R}^+$

- State: M Environment: $\gamma \in \Gamma$
- Output Values: V
- Number of random sites p

One big integral on random seeds instead of nested integrals at each synchronous step.

$$\int_{[0,1]^3} \text{let } M', V', W' = \left(\!\left|\begin{array}{l}\texttt{e where rec init x = c}\\ \texttt{\quad and x = e\_x}\\ \texttt{\quad and y = e\_y}\end{array}\right|\!\right)^{\text{step}}_{\gamma}(M), [r, r_x, r_y]) \text{ in } \delta_{M'} \otimes \delta_{V'} * W' \, dr dr_x dr_y =$$

$$\left\{\!\left|\begin{array}{l}\texttt{e where rec init x = c}\\ \texttt{\quad and x = e\_x}\\ \texttt{\quad and y = e\_y}\end{array}\right|\!\right\}^{\text{step}}_{\gamma}(M))$$

Probabilistic Mutual Recursive equations

GoTo Kernel

Streams defined by causal equations

```
(x, y) where
    rec x = sample(gaussian(42, 1))
    and y = x
```

Scheduled semantics

$$\int \delta_{x,x} * \mathcal{N}(42,1)(dx)$$

Fixpoint given random seed

$$\rho_0(r) = [x \leftarrow \bot,\ y \leftarrow \bot]$$
$$\rho_1(r) = [x \leftarrow \mathit{icdf}_{\mathcal{N}(42,1)}(r),\ y \leftarrow \bot]$$
$$\rho_2(r) = [x \leftarrow \mathit{icdf}_{\mathcal{N}(42,1)}(r),\ y \leftarrow \mathit{icdf}_{\mathcal{N}(42,1)}(r)]$$
$$\rho_\infty(r) = [x \leftarrow \mathit{icdf}_{\mathcal{N}(42,1)}(r),\ y \leftarrow \mathit{icdf}_{\mathcal{N}(42,1)}(r)]$$

In one time step

```
let rec x = sample(gaussian(42, 1))
    and y = x
    in (x, y)
```

Density semantics

$$\int_{[0,1]} \mathit{let}\ x, y = \rho_\infty(r)(x,y)\ \mathit{in}\ \delta_{(x,x)} dr$$
$$= \int_{[0,1]} \mathit{let}\ x = \mathit{icdf}_{\mathcal{N}(42,1)}(r)\ \mathit{in}\ \delta_{(x,x)} dr$$
$$= \int \delta_{(x,x)} * \mathcal{N}(42,1)(dx)$$

(change of variable formula)

Co-iterative density semantics

$$[\![\texttt{infer}(e)]\!]_\gamma^{\mathsf{D}\,\mathrm{init}} = \textit{let } m, p = (\!|e|\!)_\gamma^{\mathrm{init}} \textit{ in } \delta_m, p$$

$$[\![\texttt{infer}(e)]\!]_\gamma^{\mathsf{D}\,\mathrm{step}}(\sigma, p) = \textit{let } \psi(m) = \int_{[0,1]^p} \textit{let } m', v, w = (\!|e|\!)_\gamma^{\mathrm{step}}(m, r) \textit{ in } w * \delta_{(m',v)} \; dr \textit{ in}$$
$$\textit{let } \nu = \int \sigma(dm)\ \psi(m) \textit{ in let } \overline{\nu} = \nu/\nu(\top) \textit{ in } (\pi_{1*}(\overline{\nu}), p), \pi_{2*}(\overline{\nu})$$

$$\left[\!\!\left[\texttt{infer}\left(\begin{array}{l}\texttt{(x, y) where}\\ \texttt{\ \ rec x = sample(bernoulli(0.2))}\\ \texttt{\ \ and y = 1 - x}\end{array}\right)\right]\!\!\right]^{\mathsf{D}\,\mathrm{init}} = \delta_{((),()),(),()}, 1$$

$$\left[\!\!\left[\texttt{infer}\left(\begin{array}{l}\texttt{(x, y) where}\\ \texttt{\ \ rec x = sample(bernoulli(0.2))}\\ \texttt{\ \ and y = 1 - x}\end{array}\right)\right]\!\!\right]^{\mathsf{D}\,\mathrm{step}} = (\delta_{((),()),(),()}, 1),\ \int_{[0,1]} (\mathbb{1}_{r \le 0.2} * \delta_{(1,0)} + \mathbb{1}_{r \ge 0.2} * \delta_{(0,1)}) dr$$
$$= (\delta_{((),()),(),()}, 1),\ [(1,0) \mapsto 0.2,\ (0,1) \mapsto 0.8]$$

Probabilistic Synchronous Semantics

Relational Density Semantics

Relational density semantics

GoTo Deterministic

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$\frac{G, H \vdash e \downarrow s}{G, H, [] \vdash e \Downarrow (s, 1)} \qquad \frac{G, H \vdash e \downarrow s_\mu}{G, H, [R] \vdash \texttt{sample}(e) \Downarrow (icdf_{s_\mu}(R), 1)}$$

$$\frac{G, H \vdash e \downarrow w}{G, H, [] \vdash \texttt{factor}(e) \Downarrow ((), w)} \qquad \frac{G, H + H_E, R_E \vdash E : w_E \qquad G, H + H_E, R_e \vdash e \Downarrow (s, w)}{G, H, [R_e : R_E] \vdash e \texttt{ where rec } E \Downarrow (s, w * w_E)}$$

$$\frac{G, H, R \vdash e \Downarrow (H(x), w)}{G, H, R \vdash x = e : w}$$

Relational density semantics

GoTo Deterministic

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$\frac{p = \mathrm{RV}(e) \qquad \Big[G, H, R \vdash e \Downarrow (s, w) \qquad \overline{w} = \prod\ w\Big]_{R \in ([0,1]^\omega)^p}}{G, H \vdash \texttt{infer}(e) \downarrow \mathit{integ}_p\ \overline{w}\ s}$$

$$\mathit{integ}_p\ (\overline{w}_n \cdot \overline{ws})\ (v \cdot vs) = \\ \big(\mathit{let}\ \mu = \textstyle\int \overline{w}_n(H,R)\delta_{v(H,R)}\ \lambda_\omega^p(dR)\ \mathit{in}\ \mu/\mu(\top)\big) \cdot (\mathit{integ}\ \overline{ws}\ vs)$$

Relational density semantics

GoTo Deterministic

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$[0.1 \cdot 0.33 \cdot 0.12 \cdot \ldots] \vdash$$

```
(x, y) where
  rec x = sample(bernoulli(0.2))
  and y = 1 - x
```

$$\downarrow \begin{cases} s = & (1,0) & \cdot & (0,1) & \cdot & (1,0) & \cdot & \ldots \\ w = & 1 & \cdot & 1 & \cdot & 1 & & \ldots \end{cases}$$

Relational density semantics

GoTo Deterministic

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$[0.72 \cdot 0.08 \cdot 0.14 \cdot \ldots] \vdash$$

```
(x, y) where
  rec x = sample(bernoulli(0.2))
  and y = 1 - x
```

$$\downarrow \begin{cases} s = & (0,1) & \cdot & (1,0) & \cdot & (1,0) & \cdot & \ldots \\ w = & 1 & \cdot & 1 & \cdot & 1 & \ldots \end{cases}$$

Relational density semantics

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$[0.72 \cdot 0.08 \cdot 0.14 \cdot \ldots] \vdash$$

```
(x, y) where
  rec x = sample(bernoulli(0.2))
  and y = 1 - x
```

$$\downarrow \begin{cases} s = & (0,1) & \cdot & (1,0) & \cdot & (1,0) & \cdot & \ldots \\ w = & 1 & \cdot & 1 & \cdot & 1 & & \ldots \end{cases}$$

$$integ_p\ (\overline{w}_n \cdot \overline{ws})\ (v \cdot vs) = \left(\int_{[0,1]^\omega} \delta_{v(H,R)} dR\right) \cdot integ_p\ 1\ vs = \mu \cdot \mu \cdot \ldots$$

where μ is the categorical distribution $[(1,0) \mapsto 0.2,\ (0,1) \mapsto 0.8]$

Relational density semantics

GoTo Deterministic

Sequents

Evaluates expressions to streams: $G, H, R \vdash e \downarrow s, w$

Checks stream equations: $G, H, R \vdash E, W$

- G globals
- H maps all variables to streams
- R random array streams

λ_ω^p is the Lebesgues measure on the cube of random streams $([0,1]^\omega)^p$.

$$integ_p \ \ (\overline{w}_n \cdot \overline{ws}) \ \ (v \cdot vs) \ \ = \ \ \left(\int_{[0,1]^\omega} \delta_{v(H,R)} dR \right) \cdot integ_p \ \ 1 \ \ vs \ \ = \ \ \mu \cdot \mu \cdot \ldots$$

where μ is the categorical distribution $[(1,0) \mapsto 0.2, \ (0,1) \mapsto 0.8]$

$$\vdash \quad \texttt{infer} \left(\begin{array}{l} \texttt{(x, y) where} \\ \quad \texttt{rec x = sample(bernoulli(0.2))} \\ \quad \texttt{and y = 1 - x} \end{array} \right) \quad \downarrow \quad \mu \cdot \mu \cdot \mu \cdot \ldots$$

Relational density semantics rules

$$\frac{G, H \vdash e \downarrow s}{G, H, [] \vdash e \Downarrow (s, 1)} \qquad \frac{G, H \vdash e \downarrow s_\mu}{G, H, [R] \vdash \texttt{sample}(e) \Downarrow (icdf_{s_\mu}(R), 1)} \qquad \frac{G, H \vdash e \downarrow w}{G, H, [] \vdash \texttt{factor}(e) \Downarrow ((), w)}$$

$$\frac{G, H, R_e \vdash e \downarrow (s_e, w_e) \qquad G(f) = \texttt{proba}\ f\ x = e_f \qquad G, [x \leftarrow s_e], R_f \vdash e_f \Downarrow (s, w)}{G, H, [R_e : R_f] \vdash f(e) \Downarrow (s, w * w_e)}$$

$$\frac{G, H + H_E, R_E \vdash E : w_E \qquad G, H + H_E, R_e \vdash e \Downarrow (s, w)}{G, H, [R_e : R_E] \vdash e\ \texttt{where rec}\ E \Downarrow (s, w * w_E)} \qquad \frac{G, H, R \vdash e \Downarrow (H(x), w)}{G, H, R \vdash x = e : w}$$

$$\frac{G, H, R \vdash e \Downarrow (i \cdot s, w_i \cdot w) \qquad H(x.\mathrm{last}) = i \cdot H(x)}{G, H, R \vdash \texttt{init}\ x = e : w_i \cdot 1} \qquad \frac{G, H, R_1 \vdash E_1 : w_1 \qquad G, H, R_2 \vdash E_2 : w_2}{G, H, [R_1 : R_2] \vdash E_1\ \texttt{and}\ E_2 : w_1 * w_2}$$

$$\frac{p = \mathrm{RV}(e) \qquad [G, H, R \vdash e \Downarrow (s, w) \qquad \overline{w} = \Pi\ w]_{R \in ([0,1]^\omega)^p}}{G, H \vdash \texttt{infer}(e) \downarrow integ_p\ \overline{w}\ s}$$

Probabilistic Synchronous Semantics

Equivalent semantics

Equivalence Co-iterative vs Relational density semantics

Corollary (for a causal model)

For any R, H and γ such that $\forall k,\ \forall x \in \mathrm{FV}(e),\ \gamma_k(x) = H(x)_k$,

$$G, H, R \vdash e \downarrow s, w \text{ iff } \forall k,\ s_k(H,R) = v_k(H_{\leq k} R_{\leq k}) \text{ and } \prod_{i=1}^{k} w_i(H,R) = \overline{w}_k(H_{\leq k} R_{\leq k})$$

$$\text{where } \begin{cases} m_0 & = (\!|e|\!)^{\mathrm{init}}_{\gamma_0} \\ m_k, v_k, w_k & = (\!|e|\!)^{\mathrm{step}}_{\gamma_k}(m_{k-1}, R_k) \end{cases}$$

Theorem (relational correctness) For a causal model e, and for any H,

$$\text{if } G, H \vdash \texttt{infer}(e) \downarrow \mu \text{ then } \begin{cases} \sigma_0, p & = [\![\texttt{infer}(e)]\!]^{\mathrm{D\,init}}_{\gamma_0} \\ \forall k > 0, (\sigma_{k+1}, p), \mu_{k+1} & = [\![\texttt{infer}(e)]\!]^{\mathrm{D\,step}}_{\gamma_{k+1}}(\sigma_k, p). \end{cases}$$

Probabilistic Synchronous Semantics

Program Equivalence

Program equivalence

Definition: $e_1 \cong e_2$ if for all **input** contexts, e_1 and e_2 produce the same **output** streams.

Program equivalence

Definition: $e_1 \cong e_2$ if for all **input** contexts, e_1 and e_2 produce the same **output** streams.

Relational equivalence:

Probabilistic expressions $e_1 \cong e_2$ with $\mathrm{RV}(e_1) = p_1$ and $\mathrm{RV}(e_2) = p_2$ if $\forall H,\ \forall k > 0$,

$$\int \overline{w_1}_k(H, R_1) \ * \ \delta_{s_{1_k}(H,R_1)} \quad d\lambda_\omega^{p_1}(R_1) \quad = \quad \int \overline{w_2}_k(H, R_2) \ * \ \delta_{s_{2_k}(H,R_2)} \quad d\lambda_\omega^{p_2}(R_2)$$

where $G, H, R_1 \vdash e_1 \Downarrow (s_1, w_1)$ and $G, H, R_2 \vdash e_2 \Downarrow (s_2, w_2)$.

Program equivalence

Definition: $e_1 \cong e_2$ if for all **input** contexts, e_1 and e_2 produce the same **output** streams.

Relational equivalence:

Probabilistic expressions $e_1 \cong e_2$ with $\mathrm{RV}(e_1) = p_1$ and $\mathrm{RV}(e_2) = p_2$ if $\forall H$, $\forall k > 0$,

$$\int \overline{w_1}_k(H, R_1) \ * \ \delta_{s_{1k}(H,R_1)} \ d\lambda_\omega^{p_1}(R_1) \quad = \quad \int \overline{w_2}_k(H, R_2) \ * \ \delta_{s_{2k}(H,R_2)} \ d\lambda_\omega^{p_2}(R_2)$$

where $G, H, R_1 \vdash e_1 \Downarrow (s_1, w_1)$ and $G, H, R_2 \vdash e_2 \Downarrow (s_2, w_2)$.

Proposition: if there is $f : [0,1]^{p_1} \to [0,1]^{p_2}$ measurable, that preserves uniform distribution

$e_1 \cong e_2$ when $\forall H$ and $\forall R$, $\forall k > 0$, $\begin{cases} s_{1k}(H, R) = s_{2k}(H, f(R)) \\ w_{1k}(H, R) = w_{2k}(H, f(R)) \end{cases}$,

where $G, H, R \vdash e_1 \Downarrow (s_1, w_1)$ and $G, H, f(R) \vdash e_2 \Downarrow (s_2, w_2)$.

Program equivalence, an example

$$\texttt{sample}(e_1) + \texttt{sample}(e_2) \cong x + y \texttt{ where rec } x = \texttt{sample}(e_2) \texttt{ and } y = \texttt{sample}(e_1)$$

Program equivalence, an example

$$\texttt{sample}(e_1) + \texttt{sample}(e_2) \cong x + y \texttt{ where rec } x = \texttt{sample}(e_2) \texttt{ and } y = \texttt{sample}(e_1)$$

$$\frac{H, R_1 \vdash \texttt{sample}(e_1) \Downarrow (s_1, w_1) \qquad H, R_2 \vdash \texttt{sample}(e_2) \Downarrow (s_2, w_2)}{H, [R_1 : R_2] \vdash \texttt{sample}(e_1) + \texttt{sample}(e_2) \Downarrow (s_1 + s_2, w_1 w_2)}$$

Program equivalence, an example

$$\texttt{sample}(e_1) + \texttt{sample}(e_2) \cong x + y \texttt{ where rec } x = \texttt{sample}(e_2) \texttt{ and } y = \texttt{sample}(e_1)$$

$$\frac{H, R_1 \vdash \texttt{sample}(e_1) \Downarrow (s_1, w_1) \qquad H, R_2 \vdash \texttt{sample}(e_2) \Downarrow (s_2, w_2)}{H, [R_1 : R_2] \vdash \texttt{sample}(e_1) + \texttt{sample}(e_2) \Downarrow (s_1 + s_2, w_1 w_2)}$$

$$\frac{H + H_E, [] \vdash x + y \Downarrow (s_2 + s_1, 1) \qquad \dfrac{\dfrac{H + H_E, R_2 \vdash \texttt{sample}(e_2) \Downarrow (s_2, w_2)}{H + H_E, R_2 \vdash x = \texttt{sample}(e_2) : w_2} \quad \dfrac{H + H_E, R_1 \vdash \texttt{sample}(e_1) \Downarrow (s_1, w_1)}{H + H_E, R_1 \vdash y = \texttt{sample}(e_1) : w_1}}{H + H_E, [R_2 : R_1] \vdash x = \texttt{sample}(e_2) \texttt{ and } y = \texttt{sample}(e_1) : w_1 w_2}}{H, [R_2 : R_1] \vdash x + y \texttt{ where rec } x = \texttt{sample}(e_2) \texttt{ and } y = \texttt{sample}(e_1) \Downarrow (s_2 + s_1, w_1 w_2)}$$

where $H_E = [x \leftarrow s_2, y \leftarrow s_1]$.

Soundness of a Program Transformation

APF

Remember JFLA'23

Filtrer sans s'appauvrir : inférer les paramètres constants des modèles réactifs probabilistes

Guillaume Baudart,[1,2] Grégoire Bussone [2,3]
Louis Mandel,[4] et Christine Tasson[3]

[1] Inria Paris
[2] École normale supérieure – PSL university
[3] Sorbonne Université
[4] IBM Research

Résumé

ProbZelus étend le langage synchrone Zelus pour permettre de décrire des modèles probabilistes synchrones. Là où la programmation synchrone implémente des fonctions de suites, un langage probabiliste synchrone permet de décrire des modèles qui calculent des suites de distributions. On peut par exemple estimer la position d'un objet en mouvement à partir d'observations bruitées ou estimer l'incertitude d'un capteur à partir d'une suite d'observations. Ces problèmes mêlent des flots de variables aléatoires – les *paramètres d'état* qui changent au cours du temps – et des variables aléatoires constantes – les *paramètres constants*.

Pour estimer les paramètres d'état, les algorithmes d'inférence bayésienne de Monte Carlo séquentiels reposent sur des techniques de filtrage. Le filtrage est une méthode approchée qui consiste à perdre volontairement de l'information sur l'approximation actuelle pour recentrer les estimations futures autour de l'information la plus significative. Malheureusement, cette perte d'information est dommageable pour l'estimation des paramètres constants qui n'évoluent pas au cours du temps. Ce phénomène s'appelle l'appauvrissement.

Inspirés de la méthode d'inférence *Assumed Parameter Filter* (APF), nous proposons (1) une analyse statique, (2) une passe de compilation et (3) une nouvelle méthode d'inférence modulaire pour ProbZelus qui évite l'appauvrissement pour les paramètres constants tout en gardant les performances des algorithmes de Monte Carlo séquentiels pour les paramètres d'états.

1 Introduction

La programmation synchrone flot de données [9] est un paradigme dans lequel les fonctions, appelées nœuds, manipulent des suites infinies, appelées flots. Ces flots progressent au même rythme, de manière synchrone. L'expressivité des langages synchrones est volontairement restreinte, ce qui permet à des compilateurs spécialisés de générer du code efficace et correct par construction avec de fortes garanties sur le temps d'exécution et sur l'utilisation de la mémoire [10]. Le langage industriel Scade est notamment utilisé pour la conception de systèmes embarqués critiques [13].

Les langages probabilistes [6, 14, 17, 18] étendent des langages de programmation généralistes avec des constructions probabilistes. Suivant la méthode bayésienne, un *modèle* probabiliste décrit une distribution de probabilité (la distribution *a posteriori*) à partir d'une croyance initiale (la distribution *a priori*) et d'observations concrètes (les entrées).

Dans la même lignée de langages, ProbZelus [2] est une extension probabiliste du langage synchrone flot de données Zelus [9]. Ce langage permet de décrire des modèles réactifs probabilistes. Là où la programmation synchrone implémente des fonctions de suites, un langage probabiliste synchrone permet de décrire des modèles qui calculent des suites de distributions.

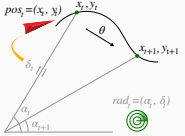
Problem: Implementation and Correction of a transformation of probZelus programs in order to apply APF

Semantics tools: Kernel co-iterative semantics

Correction:

- 😱 Base case : ✓: 8 pages
- 🥵 General case: ✗ Far too many nested integrals and tedious (even perhaps impossible)

Program transformation




```
proba f(pre_x) = pre_x + theta where
  rec init theta = sample(gaussian(zeros, st))
  and theta = last theta

proba tracker(rad_obs) = pos where
  rec  init pos = pos_init
  and pos = sample(gaussian(f(last pos), sp))
  and rad = g(pos)
  and () = observe(gaussian(rad, sr), rad_obs)

node main(rad_obs) = u where
  rec pos_dist = infer (tracker (rad_obs))
  and msg = controller(pos_dist)
```

```
let f_prior = gaussian(zeros, st)
proba f_model(theta, pre_pos) = pre_pos + theta

let tracker_prior = f_prior
proba tracker_model(theta, rad_obs) = pos where
  rec init pos = pos_init
  and pos = sample(gaussian(f_prior(theta, last pos), sp))
  and rad = g(pos)
  and () = observe(gaussian(rad, sr), rad_obs)

node main(rad_obs) = msg where
  rec pos_dist = APF.infer(tracker_model, tracker_prior, rad_obs)
  and msg = controller(pos_dist)
```

$$\texttt{APF.infer}(f.\text{model}, f.\text{prior}, e) \triangleq \texttt{infer}(f.\text{model}(\theta, e) \texttt{ where rec init } \theta = \texttt{sample}(f.\text{prior}))$$

Soundness of a Program Transformation

Soundness proof

Soundness of a Program Transformation

Theorem

$$G, H \vdash \texttt{infer}(f(e)) \downarrow d \quad \text{iff} \quad G^{+}, H \vdash \texttt{APF.infer}(f.\text{model}, \; f.\text{prior}, \; e) \downarrow d$$

Co-iterative density semantics

- 😌 Base case : ✓
- 🤔 General case: $0.1\checkmark + 0.9\times$: 8 pages (missing step: bisimulation from one time step to the next one, interaction integral and states)

Relational density semantics

- 😌 Base case : ✓
- 🥳 General case: ✓: 2 pages !

Take home

Motto: Every programmer can perform data analysis by describing models as programs and key operations.


Van de Meent & al. Introduction to Probabilistic Programming. 2018

Semantics

- Might be useful
- Two equivalent density semantics for ProbZelus
- Proof of a program transformation for an optimized inference scheme

Future

- Denotational validation of inference in ProbZelus
- Higher order probabilistic synchronous programming
- Probabilistic control with differential equations

The category of measurable space and measurable functions is symmetric monoidal but not closed.

By contradiction: Assume Meas is an SMCC: $\forall X, Y, \text{ev} : Y^X \otimes X \to Y$ is measurable.

Measurable spaces X is $\mathbb{R}$ with $\Sigma_X = \mathcal{P}(X)$ any parts and Y is $\mathbb{R}$ with the Σ-algebra generated by countable and cocountable subsets (closed by complement and countable unions and intersections).

Diagonal function: $h : \begin{cases} (\mathbb{R} \times \mathbb{R}, \mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})) & \to \{0,1\} \\ (x,y) & \mapsto 1 \text{ if } x = y \\ & \mapsto 0 \text{ otherwise} \end{cases}$,

$\Lambda(h) : (\mathbb{R}, \mathcal{P}(\mathbb{R})) \to (\{0,1\}^{\mathbb{R}}, \Sigma_{2^Y})$ is **measurable**

$h = \text{ev} \circ \Lambda(h)$ is **measurable** as the composition of measurable functions

$\Delta = \{(x,y) \in \mathbb{R}^2 \mid x = y\} = h^{-1}(1)$ is measurable in $\mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$.

The category of measurable spaces and measurable functions is not closed.

By contradiction: Assume Meas is an SMCC: $\forall X, Y, \mathrm{ev} : Y^X \otimes X \to Y$ is measurable.

We deduced that $\Delta = \{(x, y) \in \mathbb{R}^2 \mid x = y\} = h^{-1}(\{1\})$ is **measurable** in $\mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$.

If $W \in \mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$, then there is $B \subseteq \mathbb{R}$ countable such that:

$$\text{if there is } (x, y) \in W \text{ such that } y \notin B, \text{ then } \forall z \notin B, (x, z) \in W.$$

Proof: satisfied by basic measurable subsets and closed by countable intersection and unions

Consequence: Δ satisfies this property: there is B countable such that there is $(x, x) \in \Delta$ and $x \notin B$. Yet, for any $z \notin B$ and $z \neq x$, $(x, z) \notin \delta$.

✗ **CONTRADICTION**